

Efficient Automatic Engineering Design Synthesis via Evolutionary Exploration

Thesis by
Cin-Young Lee

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy



California Institute of Technology
Pasadena, California

2002

(Defended May 22, 2002)

© 2002

Cin-Young Lee

All rights Reserved

Acknowledgements

There are a great many people that I need to thank and acknowledge for their part in this thesis. First, I would like to thank my advisor, Professor Erik K. Antonsson, whose laissez-faire approach to graduate guidance was greatly appreciated because it allowed me to explore freely, which meant, equivalently, that I could learn about the things I enjoy. Learning became fun for me once again and, in my opinion, that's what education should be. Unfortunately, it took me some two decades to regain this childhood joy.

My parents deserve a special thanks. Without their love and support I could never have come so far. I hope these two simple sentences can convey the magnitude of my gratitude. I also thank them deeply for giving me Cin-Ty, my brother. He has taught me more than he knows, as he has been a model to which I have often aspired.

Additional thanks go to my friends and peers who have imparted their wisdom and wit upon my psyche. They all know who they are, so they shall remain nameless, but not unthanked.

Foreword

Everything has a story and this is the story of how this thesis came to be. It is my personal opinion that much can be learned through history to clarify the present. My hope is that by describing the events that culminated in the completion of this thesis that the work will have some context and relevance. Because, in contrast to the cliché, it's not only about where you get, but how you get there. After all, results are useless if they cannot be replicated.

Throughout the early years of my tenure at Cal, I did not have any idea what I wanted to do. Sure, I was a mechanical engineering and materials science major, but, like most undergraduates, I didn't know what I was getting into when I chose those majors. By the time I did know, it was too late to change. So, in the summer of my junior year, I went off to Sandia National Labs to work with Dr. Robert W. Schwartz, now a professor at Clemson University, on pyroelectric PZT thin films. I decided then that, as a future career, I would like to be in the microelectronics field since it seemed to be a hot and lucrative field. Unfortunately, my education was not well suited to the choice I had made.

After returning to Cal for my senior year, I began working with Prof. George C. Johnson and his student Peter T. Jones. To my great surprise and joy, our research focused on MEMS - a popular new microelectronics technology that was ideally suited to study by mechanical engineers and material scientists. That decided it. My graduate career would focus on MEMS research. With this goal in mind, I went to Michigan to study under Prof. Liwei Lin who, at the time, was an up and coming star of the MEMS arena. Michigan was a great place and I met some amazing people, but I could not handle the cold winters of Michigan and I missed sunny California. So, I asked Prof. Erik Antonsson if he would consider re-admitting me to Caltech. Thankfully, he said yes and I was on my way to Caltech to study MEMS.

Upon arriving at Caltech, I learned that the MEMS research being conducted by Erik was predominantly computational. Being a traditional mechanical engineer, I was presented with the problem of being rather uneducated in the ways of computation. I was forced to pick up C and C++ to

continue our current research directions.

A senior graduate student, by the name of Hui “Frank” Li, laid the groundwork of our current MEMS research efforts by developing a mask-layout synthesis tool that implemented a genetic algorithm (in fact, an earlier student, Ted Hubbard, was the first in Erik’s group to study MEMS in some manner). As I was to be continuing Frank’s work, I had to understand how genetic algorithms worked. At first, I was a bit apprehensive (I mean what does a mechanical engineer know about biology?), but as I began learning more about genetic algorithms and evolutionary computation in general, my interest in the subject grew.

I along with another student, Lin Ma, who was my senior, were given the task of improving the shortcomings of Frank’s work. At least initially, Lin and I worked on two separate lines of research. His research emphasized the introduction of multiple processes into the mask-layout synthesis application; whereas I tried to remove the restriction of fixed number of polygon vertices.

This began my interest in the tool rather than the application, since the application, it seemed to me, was only a specialized instance of the tool. Of course, by tool I mean evolutionary computation. So, I began studying evolutionary computation in earnest and tried to develop a general variable length chromosome representation. This fortuitously led to my work with non-coding segments and linkage learning, which further led to the development of the speciation mechanisms in this thesis. In fact, if one has a keen eye, they should no doubt see the relations amongst all of these before I describe the issue near the end of my thesis.

There are two “morals” I hope the reader will take from this story. The first is that my choices were guided in some manner, a selection if you will. Second, there were chance encounters that led me to my fortunate coincidences. In fact, one might say that this thesis evolved from earlier ideas.

The format of this thesis keeps with the spirit of this foreword in that not only are the chapters sequentially ordered in a logical manner, but also in chronological order that clearly shows the evolution of the ideas developed in this thesis.

Abstract

The evolution of designs in nature has been the inspiration for this thesis, which seeks to develop a framework for efficient automatic engineering design synthesis based on evolutionary methods.

The design synthesis process is equated to an evolutionary process. Because of this, the same formalization for evolution, the evolution algorithm, is used as a design synthesis formalism. Implementation of the evolution algorithm on a computer allows evolution of non-biological systems, and, hence, automatic engineering design synthesis. The early and canonical versions of such evolutionary computation are bare bones evolution tools that neglect several key aspects of evolutionary systems. Three aspects are identified as being universal in all designs, evolved by nature or by man. These are variable complexity, modularity, and speciation.

Framed in an evolutionary context, each of these characteristics are requisites for being able to evolve in correspondence with a dynamic environment. Those that are most evolvable will survive. After all, if a species cannot evolve quickly enough with changes in the environment, it will perish. In a design context, this indicates that the characteristics are vital for efficiency and shorter design cycles.

An integrated framework is developed to address all three aspects individually or in any combination thereof, which has not been done heretofore. The Because of the poor theoretical foundations of evolutionary computation, the effectiveness of the developed approach is determined through computer experimentation on several test and design problems. Results are promising as all three aspects were successfully achieved in comparison to canonical evolutionary computation.

Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 1 |
| 1.1 | Motivation | 1 |
| 1.2 | Background | 1 |
| 1.3 | Thesis Contributions | 2 |
| 1.4 | Thesis Overview | 3 |
| 2 | Darwinian Evolution and Engineering Design | 5 |
| 2.1 | Formalizing Design | 6 |
| 2.2 | Evolutionary Computation Details | 8 |
| 2.2.1 | Information Encoding | 9 |
| 2.2.2 | Variation | 9 |
| 2.2.3 | Initialization | 10 |
| 2.2.4 | Evaluation | 11 |
| 2.2.5 | Selection | 11 |
| 2.2.6 | Termination | 12 |
| 2.3 | Evolutionary Computation Theory | 12 |
| 3 | Evolvable Designs | 15 |
| 4 | Variable Length Representations | 18 |
| 4.1 | Introduction | 18 |
| 4.2 | Previous Work | 19 |
| 4.3 | Development of Variable Length Chromosomes | 19 |
| 4.4 | Test Problems | 23 |
| 4.4.1 | Proof of Concept | 23 |
| 4.4.2 | Trade-Off Problem | 24 |

| | | |
|----------|--|-----------|
| 4.5 | Design Problems | 26 |
| 4.5.1 | Design of a Pattern Classifier | 26 |
| 4.5.2 | Design of 2-D Polygons | 36 |
| 4.6 | Summary and Discussion | 42 |
| 5 | Modularity and Linkage Learning | 43 |
| 5.1 | Introduction | 43 |
| 5.2 | Previous Work | 44 |
| 5.2.1 | Non-Coding DNA Segments | 45 |
| 5.3 | Development of Linkage Learning and Modularity | 46 |
| 5.3.1 | Non-Coding Segments | 46 |
| 5.3.2 | Coding Segments | 48 |
| 5.4 | Test Problem | 49 |
| 5.4.1 | Royal Road Problem | 49 |
| 5.5 | Design Problems | 54 |
| 5.5.1 | Design of Artificial Neural Networks | 54 |
| 5.5.2 | Design of 2-D Truss Structures | 85 |
| 5.6 | Summary and Discussion | 91 |
| 6 | Speciation | 93 |
| 6.1 | Introduction | 93 |
| 6.2 | Previous Work | 94 |
| 6.2.1 | Speciation by Topological Isolation | 95 |
| 6.2.2 | Speciation by Separation | 95 |
| 6.3 | Development of Speciation | 96 |
| 6.3.1 | Discussion | 98 |
| 6.4 | Test Problems | 98 |
| 6.4.1 | Multimodal Functions | 99 |
| 6.4.2 | 1-D Ising Model | 108 |
| 6.5 | Design Problem | 111 |
| 6.5.1 | Design of Artificial Neural Networks | 111 |
| 6.6 | Summary and Discussion | 112 |

| | |
|--|------------|
| 7 Conclusion | 114 |
| 7.1 Summary | 114 |
| 7.2 Discussion and Future Work | 115 |
| A Source Code | 118 |
| B Table of Abbreviations | 126 |
| Bibliography | 127 |

List of Figures

| | | |
|------|--|----|
| 2.1 | Diagram of the evolution algorithm. | 6 |
| 2.2 | Diagram of evolutionary computation – a modified evolution algorithm. | 8 |
| 4.1 | Test Problem 1 with $L = 50$. Proof of concept test case for variable length search. | 24 |
| 4.2 | Test Problem 2 with $L = 50$. Trade-off test case for variable length search. | 25 |
| 4.3 | Non-touching cluster data set and corresponding partitions. | 32 |
| 4.4 | Non-touching cluster data set and typical evolved partitions. | 32 |
| 4.5 | Actual (–) and typical evolved partitions (- -) for non-touching case. Actual (×) and typical evolved (⊙) cluster centers. | 33 |
| 4.6 | Touching cluster data set and corresponding partitions. | 34 |
| 4.7 | Touching cluster data set and typical evolved partitions. | 34 |
| 4.8 | Actual (–) and evolved partitions (- -) for touching case. Actual (×) and typical evolved (⊙) cluster centers. | 35 |
| 4.9 | Target Shape and Best Shape for Generation 1 | 40 |
| 4.10 | Best Shapes for Generation 15 and 25 | 40 |
| 4.11 | Best shapes at generations 65 and 105. | 41 |
| 5.1 | Diagram of the non-coding representation. | 47 |
| 5.2 | Diagram of the parallel linkage characteristics that overlapping range representations can exhibit. | 49 |
| 5.3 | Royal Road function modules. | 50 |
| 5.4 | Gene indices for the first chromosome that matches a module of the given Royal Road module level. | 53 |
| 5.5 | Average generation at which each level is discovered for population size of 100. | 54 |
| 5.6 | Example of a feedforward artificial neural network. | 56 |
| 5.7 | (a) Connection cycle. (b) 2-D index ranges needed to encode connection topology of (a). | 58 |

| | | |
|------|--|----|
| 5.8 | (a) Connection cycle with additional node. (b) 2-D index ranges needed to encode connection topology of (a). Required range of added node is shaded. | 59 |
| 5.9 | Connection cycle with 2 additional nodes that cannot be encoded by any combination of 2-D index ranges. | 60 |
| 5.10 | Parent 1. (a) Network topology. Connections are determined from overlapping index ranges shown in (b) and weights are proportional to signed p_1 difference. (b) Index ranges for each node. | 61 |
| 5.11 | Parent 2. (a) Positions and network topology. (b) Index ranges for each node. | 63 |
| 5.12 | Overlay of Parents 1 and 2. (a) Positions and network topology. Parent 1 denoted by gray nodes and integers. Parent 2 denoted by black nodes and letters. (b) Index ranges for each node. Crossover range denoted by the bold rectangle. | 64 |
| 5.13 | Offspring 1. (a) Positions and network topology. (b) Index ranges for each node. | 65 |
| 5.14 | Offspring 2. (a) Positions and network topology. (b) Index ranges for each node. | 66 |
| 5.15 | Initial best topology for Function 1 using uniform range crossover. | 70 |
| 5.16 | Final evolved topology for Function 1 using uniform range crossover. | 71 |
| 5.17 | Function 1 data, initial best approximation, and final evolved approximation using uniform range crossover. | 72 |
| 5.18 | Initial best topology for Function 2 using uniform range crossover. | 73 |
| 5.19 | Final evolved topology for Function 2 using uniform range crossover. | 74 |
| 5.20 | Function 2 data, initial best approximation, and final evolved approximation using uniform range crossover using uniform range crossover. | 75 |
| 5.21 | Initial best topology for Function 3 using uniform range crossover. | 76 |
| 5.22 | Final evolved topology for Function 3 using uniform range crossover. | 77 |
| 5.23 | Function 3 data, initial best approximation, and final evolved approximation using uniform range crossover. | 78 |
| 5.24 | Initial best topology for Function 4 using uniform range crossover. | 79 |
| 5.25 | Final evolved topology for Function 4 using uniform range crossover. | 80 |
| 5.26 | Function 4 data, initial best approximation, and final evolved approximation using uniform range crossover. | 81 |
| 5.27 | Initial best topology for Function 5. | 82 |
| 5.28 | Final evolved topology for Function 5. | 83 |
| 5.29 | Function 5 data, initial best approximation, and final evolved approximation. | 84 |

| | | |
|------|---|-----|
| 5.30 | Best truss solution in initial population. | 89 |
| 5.31 | Final evolved truss solution using uniform range crossover. | 90 |
| 6.1 | Function 1: Unimodal test function. | 100 |
| 6.2 | Function 2: Multimodal test function with equal peaks. | 100 |
| 6.3 | Function 3: Multimodal test function with different peaks. | 101 |
| 6.4 | Function 1: Typical evolved solutions for standard EC. | 102 |
| 6.5 | Function 1: Typical evolved solutions for speciation EC. | 102 |
| 6.6 | Function 1: Typical evolved index ranges for speciation EC. | 103 |
| 6.7 | Function 2: Typical evolved solutions for standard EC. | 104 |
| 6.8 | Function 2: Typical evolved solutions for speciation EC. | 105 |
| 6.9 | Function 2: Typical evolved index ranges for speciation EC. | 105 |
| 6.10 | Function 3: Typical evolved solutions for standard EC. | 106 |
| 6.11 | Function 3: Typical evolved solutions for speciation EC. | 107 |
| 6.12 | Function 3: Typical evolved index ranges for speciation EC. | 107 |
| 6.13 | Qualitative diagram of the 1-D Ising model. | 109 |
| 6.14 | Ising Model: Typical evolved index ranges. | 110 |

List of Tables

| | | |
|-----|---|-----|
| 4.1 | Iterations to convergence for Test Problem 1 and $L = 100$ for different operator combinations. | 24 |
| 4.2 | Iterations to convergence for Test Problem 2 and $L = 25$ for different operator combinations. | 25 |
| 4.3 | Generations to length convergence and final fitness values for different initial ranges. . . . | 41 |
| 5.1 | Generation convergence results for Royal Road test function. | 52 |
| 5.2 | Nodal parameters and initial ranges for neural network encodings. | 62 |
| 5.3 | Neural network test functions. | 68 |
| 5.4 | Best evolved neural network fitness after 15,000 generations. | 69 |
| 6.1 | Speciation test functions. | 99 |
| 6.2 | Convergence results for Function 1. | 101 |
| 6.3 | Convergence results for Function 2. | 104 |
| 6.4 | Convergence results for Function 3. | 106 |
| 6.5 | Convergence results for 1-D Ising model. | 110 |
| 6.6 | Neural network test functions. | 111 |
| 6.7 | Best evolved neural network fitness after 15,000 generations. | 112 |
| B.1 | Table of Abbreviations. | 126 |

Chapter 1

Introduction

The evolution of designs in nature has been the inspiration for this thesis, which seeks to develop a framework for efficient automatic engineering design synthesis based on evolutionary methods.

1.1 Motivation

Designs are procedures or plans for creating new instances of the same object. The generation of such designs is called design synthesis and has historically been a human endeavor that required human ingenuity and skill. However, many have formalized design synthesis (particularly in engineering fields) in an attempt to reduce the reliance on human resources, which leads to shorter design cycles and often more robust design solutions. Taken in the context of the computer age, formal design methodologies, with algorithmic descriptions, can be used to achieve automatic design synthesis computationally [3]. Computational implementations have the further benefit of being able to tackle problems that are not amenable to solution by man.

1.2 Background

Design is often considered an art, even in the engineering fields where formalism and algorithmic implementations are expected. The primary reason for this is that design is largely considered a creative endeavor, which is difficult, if not impossible, to formalize. In this work, formal engineering design synthesis is the structured process of synthesizing new engineering designs. A structured algorithmic description of design synthesis implies that computational solutions can be developed. This further implies that fully automated systems can be created to generate good designs. Consequently, the problem of formalizing engineering design synthesis is equivalent, in some sense, to

finding an algorithmic approach for automating the design process.

A remarkably simple automatic design system has been in use for at least 3.5 billion years. This system, of course, is biological evolution. However, it was not until the mid-nineteenth century that a plausible theory on the mechanisms of evolution was discovered and elucidated. The theory called Darwinian evolution in tribute to its originator, Charles Darwin, is also known as natural selection or “survival of the fittest” evolution¹. Darwin’s theory is embodied in the evolution algorithm that states that evolutionary change is a result of the repeated and combined action of transmission, variation, and selection of individuals with different traits. Evidently, nature’s design synthesis algorithm, the evolution algorithm, has been exceedingly successful at generating novel and complex designs. As everyone knows, success breeds imitation (an evolutionary selection in itself) and many have sought to replicate the evolution algorithm on a computational substrate. This field of study is called evolutionary computation (EC). Automatic design synthesis has long been a pursuit of evolutionary computationalists and enough literature exists on the subject to have engendered several books and book chapters [8, 9, 59].

1.3 Thesis Contributions

Although many have drawn parallels between evolution and design synthesis, to the author’s knowledge, there has been no formalization of design synthesis as an evolutionary process. This thesis provides a first attempt at equating the two and formalizing design synthesis in the process.

In addition to formalizing design, several key characteristics of “good” designs are identified and discussed in relation to why they are good. Framed within an evolutionary context, it is seen that designs with such traits are good because they are more evolvable. In other words, they can be modified more quickly to generate better designs than designs without said characteristics. As such, it becomes beneficial to introduce these concepts into an evolutionary computation implementation because they should lead to more efficient and shorter design cycles.

An EC framework was developed to achieve three of the identified characteristics. These are variable complexity, modularity, and speciation. Because of the poor theoretical foundations of EC, much of the work is empirical in nature, with the effectiveness of each approach verified through numerous computational experiments. During the development of these three approaches, it became

¹Although there are other types of evolution, notably Lamarckian, it will be understood that Darwinian evolution is meant in the sequel.

apparent that a single, unified model could be used to achieve each characteristic separately or in any combination thereof, which had not been accomplished heretofore.

The thesis contributions are explicitly listed in the following.

1. Engineering design synthesis is equated to an evolutionary process and formalized as the evolution algorithm. See Chapter 2.
2. Key concepts of good design are identified and remarked upon based on the evolutionary framework. These universal aspects are shown to have two major benefits over canonical evolutionary computation. The benefits are shortened design cycles and knowledge of the final evolved solution. See Chapter 3.
3. A variable complexity representation and corresponding operator are developed for general application in evolutionary computation. See Chapter 4.
4. The variable complexity representation is extended to evolve correct linkage characteristics and modularity. See Chapter 5.
5. An unified approach to variable complexity, linkage learning, and speciation in evolutionary computation is developed. See Chapter 6.

1.4 Thesis Overview

This thesis broadly follows the outline given by the thesis contributions. Chapter 2 is an introduction to the evolution algorithm. After this brief review, design synthesis is formalized as the evolution algorithm. Computational implementation of the evolution algorithm then leads to the development of evolutionary computation that can be used as an automatic design synthesis tool.

Some universal aspects of good designs are identified in Chapter 3. The reasons for the ubiquity of such characteristics are attributed to evolvability, which is introduced and discussed in this chapter. Three of the mentioned characteristics, variable complexity, modularity, and speciation are identified for further study and are the topic of the remaining chapters.

Chapter 4 describes in more detail the problem of variable complexity. An unique approach is developed for enabling variable complexity search, which is then tested on a variety of test and design problems. Closer inspection of the developed approach and its results revealed that modularity was unknowingly built into the representation. Chapter 5 takes this observation and extends it. The extension allows simultaneous evolution of both modular and variable complexity solutions.

Chapter 6 introduces the concept of speciation and population diversity preservation. A novel method that extends the developments in the Chapters 4 and 5 is presented in detail. Again, the developed speciation approach is tested on a variety of test and design problems. Finally, Chapter 7 concludes the thesis with a summary and discussion of work done along with an examination of possible future work.

Although not referenced anywhere, the software code written to test the developed evolutionary computation framework is included in Appendix A.

Chapter 2

Darwinian Evolution and Engineering Design

Evolution is the process of change and it is evident that most things evolve; after all, time marches on. However, the mechanisms of evolution differ from one system to another. The Grand Canyon reached its present state as a result of the continued uplift of the Colorado Plateau and incessant erosion by the Colorado River. Organisms, in contrast, reached their present states as a result of the prolonged effects of natural selection.

In his seminal work, *On the Origin of Species*, Darwin described the process of natural selection and introduced the foundations for what would come to be known as the evolution algorithm [18]. The evolution algorithm describes how any information system, be it biological, cultural or otherwise, may evolve over time¹. Simply stated, information systems evolve through the repeated action of transmission, variation, and selection. For example, in biological evolution, individuals reproduce and transmit their genetic code to offspring with some variation after which the combined pool of individuals is subjected to a selective environment that removes unfit individuals such that only the fittest individuals are left to repeat the process of transmission, variation, and selection. Broadly speaking, transmission ensures that good traits are retained in future generations while variation enables the discovery of better traits. Selection guides evolution by biasing the survival of certain traits (that are then considered “good” because they have survived). Figure 2.1 shows the iterative and cyclical properties of the evolution algorithm.

¹The application of the principles of Darwinian evolution to non-biological systems is often referred to as “Universal Darwinism” as originated by Dawkins [20].

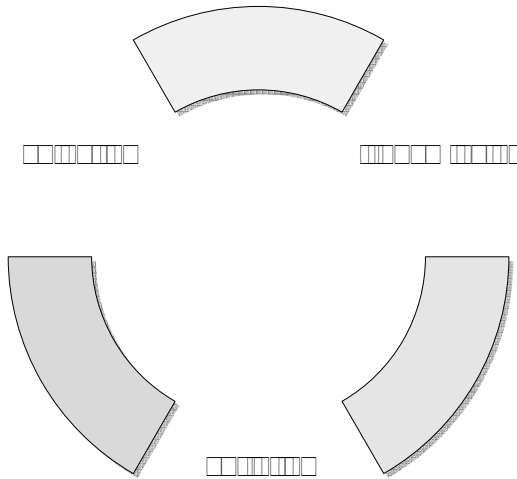


Figure 2.1: Diagram of the evolution algorithm.

2.1 Formalizing Design

Designs found in nature are undoubtedly the most remarkable known to man; however, nature cannot be considered a designer in the traditional sense because nature has no desired goal. Rather, designs manifested in nature are a result, not a goal, of Darwinian evolution (shortened to evolution in the sequel). Mankind on the other hand has subverted evolution to achieve his own goals. The earliest examples involved selective breeding for the production of more fecund plants and livestock. More recently scientists have introduced biased, as opposed to random, variations into genetic lines. These observations point to the effectiveness of evolution for goal-oriented design. But, can evolution be applied to non-biological design problems?

Revisiting the evolution algorithm, any information system can be evolved and guided through appropriate choice of the selection mechanism. Unfortunately, the problem of how to evolve non-biological systems is hardly trivial (*i.e.*, how to achieve reproduction, variation, and selection). In fact, Darwin faced a similar problem in that he did not know how traits were transmitted with variation. Gregor Mendel, in his brilliant work with peas, answered Darwin's problems [69]. Ironically, though Mendel published a few years after Darwin's *On the Origin of Species*, it was not until the mid-twentieth century that evolution and genetics (the foundations of which Mendel developed)

were reconciled in the “evolutionary synthesis”. The great breakthrough was the realization that genetic material is the transmitted information and that variation in genetic material arises primarily as a result of the action of mutation and recombination. With the inner workings of biological evolution revealed and the advent of computers, everything was in place for developing non-biological evolutionary systems, and indeed this did occur shortly after the synthesis. Several researchers independently and simultaneously extended the concepts of genetics and the evolution algorithm to non-biological problems. These extensions all came to be known as evolutionary computation methods. Today, there are four popular variants of evolutionary computation (EC). These are genetic algorithms (GA), evolution strategies (ES), evolutionary programming (EP), and genetic programming (GP), which is a derivative of GAs. GAs were introduced by Holland [45] and later popularized by Goldberg [39]. ES’s were developed in Europe by Schwefel and Rechenberg [75, 76, 79], while EP, which bears a strong resemblance to ES, was developed by Fogel [29, 30]. EP lay dormant for some time and re-emerged with Fogel’s son in the early 1990s [26]. GP is quite new, having been invented in the 1990s by Koza [52]. A good review of the seminal literature on evolutionary computation can be found in [27].

Evolutionary computation is an extension of the evolution algorithm such that non-biological systems can be evolved in a suitable manner. EC requires the addition of three steps. These are initialization, evaluation, and termination. Initialization precedes the iterative evolution loop and is necessary to create a substrate from which to begin the evolution. After initialization, the iterative transmission, variation, and selection loop begins. However, an evaluation step is required before selection. Evaluation determines the fitness of each individual, which can then be used by the selection step to bias individual survival or reproduction rates. Lastly, because human beings have finite life spans and patience, evolution cannot be allowed to continue indefinitely and must be terminated upon reaching specified conditions. The modified evolution algorithm is illustrated in Figure 2.2. See Figure 2.1 for comparison.

Just as nature has evolved remarkable designs so too can EC, except now designs can be generated with specific goals in mind (through appropriate choice of the evaluation and selection functions). Hence, this formal, structured methodology for EC also constitutes a formal description of the design process. Coincidentally, or perhaps not depending on the viewpoint, this EC formalism bears a marked resemblance to design methodologies adhered to by engineers. Typically, the engineering design process is described as an iterative process where the designer starts from an initial design (initialization), modifies it (transmission and variation), analyzes the new design to

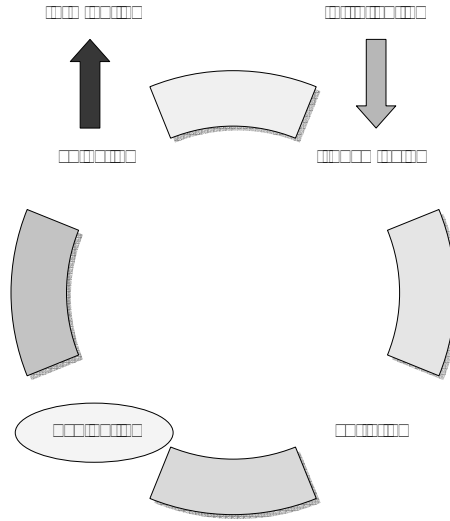


Figure 2.2: Diagram of evolutionary computation – a modified evolution algorithm.

determine performance characteristics (evaluation), chooses whether or not to keep the new design (selection), then repeats if necessary (termination). Given that the EC formalism and previous design formalisms are nearly equivalent, the EC formalism will be adopted as the design formalism in what follows.

2.2 Evolutionary Computation Details

Although the EC algorithm is a good methodology for optimization (of which design is a subset), the actual implementation of EC is still left unstandardized. As mentioned previously, EC sprung forth shortly after the evolutionary synthesis where evolution and genetics were comingled, so it is not surprising that EC implementation details are all strikingly similar to genetics. This also explains the preponderance of terminology borrowed from genetics that are introduced subsequently.

The EC framework clearly defines what details need to be fleshed out for implementation. These are, what is to be transmitted? how is variation accomplished? how is the information evaluated? how is selection implemented? how is the initial substrate or population generated? and how or when should evolution be terminated? Each is answered in turn. Of course, some, if not all, of the details are problem dependent and this is where human intervention may always be necessary. For

clarity, canonical EC implementations are described in the following.

2.2.1 Information Encoding

The goal of EC is almost always to evolve a good solution to a given problem. As such, the information being transmitted from generation to generation (or each iteration of the EC loop) must somehow encode solutions to the problem. Most EC practitioners encode each variable of feasible solutions as a single unit called, imaginatively, a gene. The set of all genes, or solution variables, can then be concatenated into a string called a chromosome. The variables are typically encoded directly as real numbers or bit strings that can be decoded into the appropriate value. So, for example, if the problem were to maximize the number of zeroes in a bit string, one could choose each gene to be a single bit with a chromosome containing as many genes as there are bits.

2.2.2 Variation

Variation is a tricky subject because a careful balance must be struck between exploration and exploitation. Exploitation refers to transmission or inheritance of previously well evolved traits. In other words, knowledge of good solutions is exploited through retention. Exploration opposes exploitation in that novel solutions are actively searched rather than retaining old knowledge. However, if the variations and modifications are too large, then offspring (transmittee) do not resemble parents (transmitter) and no exploitation can occur. Likewise, if there is little or no variation, there is too much exploitation and not enough exploration such that innovation cannot occur. This results in stagnant solutions that have prematurely converged to metastable states. It is important to note that the effect of variation operators is closely related to how the solutions are encoded.

The primary variation operators used in EC parallel the primary genetic operators and, consequently, are called mutation and recombination (also known as crossover). Mutation perturbs the value of each gene with some small probability. In bit string representations, mutation is often a bit flip, while in real number encodings mutation most often takes on the form of some addition of a normally distributed random number. Mutation acts as an explorative operator by replacing old elements with new ones in chromosomes. Conversely, recombination is chiefly an exploitative operator. Recombination swaps blocks of genes between two or more chromosomes. This has two desired effects. First, if any gene is the same in all parents, then it will remain unchanged in the offspring. If the parents are well-evolved, it is highly probable that the genes common to the parents are responsible for their high fitness; thus, these features are exploited in recombination. Second, if

any of the genes differ between parents, exploration is achieved because new combinations of genes are tested.

The necessity and utility of both operators has been widely disputed, with one camp claiming the necessity and sufficiency of mutation and another camp trumpeting the much greater importance of recombination over mutation. Making the dispute more interesting is that both have corroborating evidence in nature. Asexual organisms evolve solely through the action of mutation (disregard for the moment transmission and selection), while sexual organisms take advantage of recombination with mutation occurring infrequently in comparison. The reason for this discrepancy is that evolution occurs in a dynamic environment so organisms that evolve more quickly have a distinct advantage over creatures that evolve more slowly. This ability to evolve, or evolvability, lies at the crux of why crossover has come into being, which is illustrated in a simple example.

Imagine that in a population of identical individuals that there are two possible beneficial mutations that are additive (*i.e.*, having both mutations compounds the beneficial effect of each). Then, in an asexual population, for an individual to gain both mutations, two mutation events would have to occur serially. In contrast, crossover has the benefit of parallel evolution in which two individuals can separately obtain one of the beneficial mutations and recombine to generate an offspring with both in the following generation.

2.2.3 Initialization

Initialization refers to the initialization of individuals for the evolutionary substrate (*i.e.*, the starting point for evolution). In most instances, no bias in the starting point is desired such that initial solutions are randomly generated from an uniform distribution over the search or design space. Nonetheless, there are cases where experts have knowledge of good solutions and their expertise can be used to seed the initial population. This leads to more rapid convergence than random initialization because solutions have already been partially evolved (in the Darwinian sense) by expert selection. It should be noted that initialization is of multiple solutions, or a population of individuals. Thus, evolution occurs on a population, as in nature, rather than a single individual. Several benefits are accrued from having a population. The most notable are that crossover becomes applicable and that large populations are robust against genetic drift and premature convergence to local optima.

2.2.4 Evaluation

Evaluation is used to determine the fitness or performance of an individual solution. The selection mechanism can then bias reproduction and survival to those individuals with the best fitnesses. Evaluation is problem dependent and is nearly always specified by the human designer who determines which facets of the solution need to be maximized or minimized.

2.2.5 Selection

Selection is significantly more general and less problem specific than evaluation. The only requirement for selection is that better solutions be preferred for survival and reproduction over worse solutions. There exists a variety of selection methods, the most popular of which are roulette wheel (or proportional), rank, tournament, elitist, and truncation selection schemes. Each of these selection schemes is briefly described in relation to fitness maximization problems.

Proportional selection equates the probability of selection to an individual's fitness contribution to the aggregate fitness of the entire population. Mathematically, $p_i = f_i / \sum_{j=1}^N f_j$, where p_i is the probability of selecting individual i , f is the fitness value, and N is the population size. Proportional selection has several disadvantages. During the early stages of evolution when most individuals have poor fitness values, if a single individual has a markedly better fitness value, it will dominate the next generation because its proportion of the total fitness is much larger than that of the other individuals. This could and often does lead to premature convergence to initially lucky individuals. Similarly, near the end of an evolutionary run when the population has converged, all individuals will have nearly equal fitness values. This results in reduced selection pressures for choosing the best individuals, and evolution effectively grinds to a halt.

Rank selection was introduced to combat the shortcomings of proportional selection, which was introduced at the inception of genetic algorithms. In rank selection, each individual receives a certain number of offspring that is fixed according to the individual's fitness rank. For example, the best individual would generate five offspring, the second best four, *etc.* Though rank selection has proven more robust than proportional selection, it has the drawback that the fixed number of offspring per rank must be specified *a priori* and may not provide appropriate selective pressure.

A more robust selection method is **tournament selection**. Rather than depending on the absolute fitness values, tournament selection determines survival and reproduction based on relative fitness. In tournament selection, a subset of the entire population is chosen and the fittest individual

from this subset, or tournament, is selected for survival. Such an algorithm avoids the pitfalls of proportional selection and also allows *in situ* modification of how many offspring are generated per individual. However, the size of the subset needs to be chosen *a priori*, which leads to different selective pressures.

Elitist selection is not really a selection scheme, but a modification to other selection schemes. Elitist selection simply requires that the best individual always survives. Empirical results have shown that elitist selection schemes often outperform non-elitist selection. Thus, it is not uncommon to see the addition of explicit elitism into tournament and proportional selection schemes.

Truncation selection methods are more widely used in evolution strategies and evolutionary programming. The truncation schemes are often referred to as (μ, λ) or $(\mu + \lambda)$ selection. As opposed to the previously mentioned selection schemes, truncation culls survivors from the offspring pool, rather than the parent pool. Thus, truncation involves reproduction of more offspring than parents. μ indicates the number of parents, or population size, while λ denotes the number of offspring. In truncation selection, μ parents generate $\lambda > \mu$ offspring, some of which are truncated to reestablish the original population size. In (μ, λ) selection, survivors are culled only from the offspring pool. Modification of “,” to “+” leads to $(\mu + \lambda)$ selection where survivors are culled from the combined pool of parents and offspring – an elitist selection scheme.

2.2.6 Termination

Termination of evolution can be achieved in a variety of manners. Often a maximum limit on the number of generations is set, or some convergence criteria are specified and if met, evolution is terminated. These criteria range from low population diversity, little change in the best solution, *etc.* Unfortunately, no theoretical work to the author’s knowledge has been able to identify optimal conditions for termination when the best known solution is not known *a priori*.

2.3 Evolutionary Computation Theory

The simplicity of EC implementations and their subsequent empirical successes have led to bold claims of EC effectiveness in problem solving. These claims have not been confirmed by theoretical work, which is reviewed and discussed in this section.

A large part of the theoretical work on evolutionary computation focuses on genetic algorithms and their use of bit string representations. Nevertheless, discussion of such work is foregone here in

favor of more modern theoretical advances that are clearly true for all EC, including binary string genetic algorithms.

The convergence characteristics of EC can be determined by considering EC as a Markov chain process as first presented in [22]. Markov chain processes are memoryless processes in which the current state is sufficient to determine the following state. Moreover, state transitions are probabilistically defined. Thus, if there are a finite number of states, then a transition matrix can be defined to determine expected convergence characteristics. The difficulty lies in determination of the state transition matrix. For EC, states correspond to possible population configurations. Transition probabilities are then governed by the chosen variation and selection operators with the relationship often being too obtuse to obtain. In addition, as the number of states grows, the transition matrix grows exponentially. As an example of the intractability of EC evaluation through Markov chain analysis, take a 10 bit problem with a population size of 10. Such a simple problem already requires calculation of $2^{10} \cdot 10$ or 10,240 transition probabilities.

Two conclusions can be made from the preceding discussion. The first is that EC should employ elitist selection schemes where the best individual(s) are always kept in the population. The reason is that the EC should converge to a fixed point, hopefully the global optimum, from which it cannot depart. The only way to guarantee this is through elitist selection. The other conclusion to be made is that the effectiveness of EC implementations cannot generally be predicted in advance; thus, there is a great need for computer experimentation and the reliance of EC claims on empirical evidence is somewhat justified. The successes of EC are further tempered by Wolpert and Macready's No Free Lunch (NFL) theorem [88].

The NFL theorem basically states that no single optimization algorithm is the best over all possible problems. This is rather intuitive and exhaustive search nicely illustrates this property. Optimization of delta functions (*i.e.*, searching for a needle in a haystack) is best done through exhaustive search. However, if there is any type of fitness correlation between neighboring points in the search space, exhaustive search should never be used because the correlations can be learned and exploited. At first glance, the NFL theorem contradicts the claims of the general applicability of EC to problem solving. Indeed, this is true. EC is not universally applicable to all problems – as it will almost certainly perform worse on randomly generated fitness landscapes than exhaustive search. The catch is that the NFL theorem pertains to all problems including both those for which nothing is known about the search space and those for which something is known. This cautions against the blind application of EC to problem solving because it is likely that EC will not perform

as expected. But, if something is known about the search space, then the EC implementation can be tailored to the problem to obtain good performance (*i.e.*, fast convergence). Furthermore, since many problems have similar fitness landscapes, these customized EC implementations are widely applicable. It turns out that this tailoring of EC is equivalent to making more evolvable solutions, which is discussed in the following chapter.

A great effort has been made here to show why certain obvious intuitions are true. The key points are that computer experimentation is necessary to determine performance characteristics, EC should not be applied blindly to any problem, and EC requires tailoring of the evolvability of solutions. These issues will resurface later in the thesis when highly evolvable solutions are developed for design problems and their effectiveness is determined empirically.

Chapter 3

Evolvable Designs

One of the greatest disadvantages of EC, as with most automated optimization algorithms, is that nothing can be said about the final evolved solution other than it works and works well. Frequently, designers would like to know what the crucial elements of the solution are and whether there are other solutions. To some extent, these issues and others can be resolved by looking at the “fossil record” or the entire evolutionary process. This presents difficulties in itself because determination of the proper relationships can get tricky with millions of data points. A cursory glance of biological evolutionary systems reveals that evolution has designed genetic codes that are modular and has also evolved multiple solutions to the problem of life. This implies that there is little need to pore over the fossil record because much of the necessary information is already encoded in the present population. An immediate question that comes to mind is whether or not similar results can be achieved within the EC framework. Indeed, they can, but there is a deeper question, that is, why should species evolve in such a way? This sets the scene for some interesting observations on evolutionary characteristics that have parallels in engineering design. These parallels should not be surprising given that engineering design is an evolutionary process. Nonetheless, it is important to point out such universal aspects of good design/designers. The most prominent of these are as follows

Variable Complexity: Designers by necessity need to design in variable complexity spaces. Evidence in nature can be seen by the variation of chromosome lengths from species to species. Similarly, computer programs are not all of the same length nor are all automobiles as complex as Formula One race cars.

Modularity and Re-Use: There is a distinct hierarchical modularity in nature with compartmentalization of the genome into chromosomes, chromosomes into genes, and genes into codons.

The compound eyes of insects provide one of the more outstanding examples of re-use in nature. Object oriented programming was developed with the intent of modularization and re-use and similar concepts can be seen in automobile design where tires are designed independently and typically re-used in the final design.

Speciation: There is almost always a plenitude of solutions to real design problems. The proliferation of species is an example of the number of different solutions that exist to the problem of life. Likewise, the great variety of word processing software and automobiles reflect the possible solutions of their respective problems.

Redundancy: The genetic code is rife with redundancy, particularly for diploid organisms, or those with double stranded DNA. Here, exactly half of the genetic code is redundant. The reasons for redundancy in design are always for safety or error correction. For example, redundant DNA strands enable self-repair mechanisms. Similarly, in engineering fields, structures are designed with redundant parts so that they have acceptable factors of safety. Redundancy plays a major role in noisy environments in which there are uncontrolled variations.

The reasons such characteristics arise are easily understood when framed in an evolutionary context; after all, only traits that increase fitness will survive when subjected to prolonged selective pressures. The increased survivability of species endowed with these traits is a result of dynamic and noisy environments. Because of changing environments, the fittest species are those that can evolve in concert with the environment. If the environment changes quickly, so too must species evolve quickly if they are to survive. This ability to evolve is termed evolvability and is a key concept in evolution. Although survival is still to the fittest, the fittest are those that evolve most ably with the dynamic environment. Returning to the context of design synthesis, evolvable designs are those that are more easily modified in accordance to shifting consumer demands, safety constraints, *etc.*.

In addition, dynamic environments in nature have resulted in the evolution of adaptability, or the ability of an individual to adapt itself to an environment *within* its life span. Human beings are the prime example of adaptable organisms; but, to a large extent, it can be said that human beings do not adapt to environments, rather they modify environments to suit themselves. Regardless of this issue, adaptation seems to be a final venue of evolution where evolutionary change has slowed to such an extent that without adaptation, the species would not proliferate.

The following chapters build a framework for implementing variable complexity, modularity, and speciation into evolutionary computation and automatic design. Redundancy is not directly ad-

dressed because evolution will naturally evolve redundant structures where needed¹. Also, evolution of adaptation, while an interesting subject, is not addressed in this work.

There are two goals for adding to the evolutionary computation framework. First, as alluded to at the outset of this chapter, evolution of modularity and speciation will reveal some of the structure of the final evolved designs. Second, evolution of more evolvable representations will lead to shorter and more efficient design cycles. Both of these goals are met in the following chapters through a single approach that is able to evolve variable complexity, modularity, and speciation in any combination. This unified approach to all three points to a more global approach to evolvability that is not addressed in previous work. Although, each aspect has been individually answered on many occasions and some have implemented two within the same evolutionary computation framework. Such related work is cited in the “Previous Work” sections in the remaining chapters.

¹Diploid chromosomes are avoided because replication errors on computers are nearly non-existent.

Chapter 4

Variable Length Representations

4.1 Introduction

Design problems often have solution spaces that are of variable complexity or dimensionality. In other words, there is an unlimited number of design parameters. It is up to the designer and the design process to determine the appropriate number. Take for example the design of automobiles. Any number of wheels could have been chosen, but for stability and cost issues, most are of the four wheeled variety. This illustrates the well known trade-off that must be made between functionality and complexity, which is often referred to as the principle of parsimony or Occam's razor.

Although most design problems have variable complexity search spaces, nearly all EC implementations have fixed length representations or chromosomes. What this means is that the structure of the optimal design is somehow known *a priori*; yet it is almost always the case that design functionality is dependent on both complexity and parameter values. Thus, without knowing the optimal parameter values, the optimal number of parameters is also unknown and vice-versa. This implies that complexity and the ensuing parameter values must be evolved simultaneously for effective search.

A general approach, as first seen in [55] by the author, is developed here for implementing variable complexity search within an EC framework. The basic premise is to tag genes with an identifier to achieve length changes through crossover operations. The remainder of this chapter is organized as follows. First, a brief review is given of previous work, after which the focus of this chapter, variable length representations, will be introduced and developed. A set of test problems is then generated to determine whether the developed implementation can actually achieve the desired goals. In these sections, detailed descriptions are given such that the computer experiments can be

repeated for outside verification. When the work has been shown to achieve the desired goals on these aforementioned test problems, it will be applied to more design oriented problems with real world applications. The chapter is concluded with a discussion of the results and a summary of the work done.

4.2 Previous Work

Because most search and optimization problems have variable complexity search spaces, it is not surprising that a tremendous amount of work has been done on developing variable length chromosomes. Many of the developments in the evolution strategy (ES) and evolutionary programming (EP) fields, which rely on mutation as the primary variation operator, implement an analogous point operator to achieve length changes [36]. In these cases, genes are randomly inserted or deleted from chromosomes. But, in genetic algorithm camps, where crossover is the favored operator, length changes are enacted through the swap of chromosome blocks of different length. A particularly brilliant idea for variable length crossover is that found in most genetic programming (GP) applications¹. In GP, computer programs are represented as LISP-like trees, and crossover swaps subtrees between parent chromosomes. Because of the differing sizes of subtrees, length changes often result. Several good references on genetic programming can be found in [6, 52, 53]. There are many other variable length chromosomes, too numerous to cite. A few of these are messy GAs, Virtual Virus, *etc* [11, 37, 43, 44].

4.3 Development of Variable Length Chromosomes

Crossover is adopted as the primary length changing operator, although insertion and deletion operators are also used. Insertion and deletion are simple operators that insert or delete a randomly chosen gene. Illustrated below are insertion and deletion at the seventh bit for a 10 bit chromosome. Each bit is a gene in this case.

Parent : 1100101011

Insertion : 11001011011

Deletion : 110010011

¹Genetic programming is considered a subset of genetic algorithms where the goal is to evolve software code. Although there are linear variants of GP, tree structures are referred to here.

Both of these operators are straightforward extensions of the point-wise mutation operators that perturb single genes. In a similar fashion, canonical two point crossover is extended to enable variable length representations. Recall that in canonical two point crossover, a range of gene indices is chosen over which swapping of genes occurs between two parents. Shown below is a crossover operation with selected endpoints of 2 and 5 (*i.e.*, genes between the second and fifth bits, inclusive, are swapped). Crossover points are demarcated by ‘|’.

Parent 1: 1|1001|01011

Parent 2: 0|1111|11100

Offspring 1: 1|1111|01011

Offspring 2: 0|1001|11100

These representations implicitly code gene index via a gene’s position. The gene indices can be made explicit by adding another string to the chromosome as shown.

Genes : 1 1 0 0 1 0 1 0 1 1

Indices : 1 2 3 4 5 6 7 8 9 10

Convention has restricted indices to integer values, but a real number valued index system is equivalent as long as appropriate real valued ranges are chosen for crossover. In addition, the indices of each gene position need not remain fixed. For example, two randomly initialized parents with differing indices can be written as

Parent 1: *Genes* : | 1 1 0 0 1 | 0 1 0 1
Indices : | 0.1 0.2 0.3 0.4 0.5 | 0.6 0.7 0.8 0.9

Parent 2: *Genes* : | 1 1 0 | 1
Indices : | 0.15 0.25 0.45 | 0.85

Notice that these parents are also of differing length. Thus, when a crossover range is chosen, it is likely that the length of the offspring will be different than that of the parents’. Crossover of the

range $[0.1, 0.5]$ results in the following offspring (crossover points are again demarcated by '|')

$$\begin{array}{l} \text{Offspring 1:} \\ \text{Genes : } | \quad 1 \quad 1 \quad 0 \quad | \quad 0 \quad 1 \quad 0 \quad 1 \\ \text{Indices : } | \quad 0.15 \quad 0.25 \quad 0.45 \quad | \quad 0.6 \quad 0.7 \quad 0.8 \quad 0.9 \end{array}$$

$$\begin{array}{l} \text{Offspring 2:} \\ \text{Genes : } | \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \quad | \quad 1 \\ \text{Indices : } | \quad 0.1 \quad 0.2 \quad 0.3 \quad 0.4 \quad 0.5 \quad | \quad 0.85 \end{array}$$

In the crossover shown above, the lengths of each parent have changed from 9 to 7 and 4 to 6 respectively. It should be apparent that total length is conserved in this operation. Furthermore, the range of possible offspring lengths is easily calculated as $[0, l_1 + l_2]$ where l_i is the length of parent i . This implies that initial chromosome lengths should be large so that a larger space of lengths can be explored during the early portion of the evolutionary process.

Although the index representation is capable of variable length search, a nearly equivalent representation based on “separations” is more useful. The separation representation encodes the difference or separation between adjacent indices. This permits normalization of indices into the range $[0, 1]$, as will be seen shortly. Normalization accomplishes two things. First, it eliminates the possibility that index ranges of parents do not overlap, which, if this occurred, would negate crossover. Second, normalization standardizes chromosome index ranges such that crossover ranges are always defined on the fixed interval $[0, 1]$. For example, a gene with both index and separation representations is shown below

$$\begin{array}{l} \text{Genes :} \quad 1 \quad 1 \quad 0 \quad 0 \quad 1 \\ \text{Indices :} \quad 0.15 \quad 0.25 \quad 0.45 \quad 0.6 \quad 0.72 \\ \text{Separations :} \quad 0.15 \quad 0.1 \quad 0.2 \quad 0.15 \quad 0.12 \quad 0.28 \end{array}$$

The additional separation value is necessary for normalization, otherwise without it the final gene index value would always be 1 after normalization. The relationship between indices and separations is then given by

$$I_k = \frac{\sum_{i=1}^k S_i}{\sum_{j=1}^{l+1} S_j} \quad (4.1)$$

where I_k is the index of gene k , S_k is the separation between genes k and $k+1$, and l is chromosome length. In words, the numerator indicates that gene indices are equal to the sum of all preceding

separations. The denominator is then needed to normalize index values such that they lie in the range $[0, 1]$. Additionally, $S_k > 0 \forall k$ holds.

The difference between the separation and index representations manifests itself in application of the crossover operator. If separations are inherited rather than indices, then index values are not maintained after crossover and normalization is required yet again. This is demonstrated in the following example with a crossover range of $[0.2, 0.7]$. Note that the parents are normalized so the sum of all separations is equal to 1.

$$\begin{array}{l}
 \text{Genes : } 1 \quad | \quad 1 \quad 0 \quad 0 \quad | \quad 1 \\
 \text{Parent 1: Separations : } 0.15 \quad | \quad 0.1 \quad 0.2 \quad 0.15 \quad | \quad 0.12 \quad 0.28 \\
 \text{Indices : } 0.15 \quad | \quad 0.25 \quad 0.45 \quad 0.6 \quad | \quad 0.72
 \end{array}$$

$$\begin{array}{l}
 \text{Genes : } 1 \quad [\quad 1 \quad 0 \quad 0 \quad] \quad 1 \\
 \text{Parent 2: Separations : } 0.05 \quad [\quad 0.2 \quad 0.2 \quad 0.15 \quad] \quad 0.12 \quad 0.28 \\
 \text{Indices : } 0.05 \quad [\quad 0.25 \quad 0.45 \quad 0.6 \quad] \quad 0.72
 \end{array}$$

$$\begin{array}{l}
 \text{Genes : } 1 \quad [\quad 1 \quad 0 \quad 0 \quad] \quad 1 \\
 \text{Offspring 1: Separations : } 0.15 \quad [\quad 0.2 \quad 0.2 \quad 0.15 \quad] \quad 0.12 \quad 0.28 \\
 \text{Indices : } 0.15 \quad [\quad 0.35 \quad 0.55 \quad 0.7 \quad] \quad 0.82
 \end{array}$$

$$\begin{array}{l}
 \text{Genes : } 1 \quad | \quad 1 \quad 0 \quad 0 \quad | \quad 1 \\
 \text{Offspring 2: Separations : } 0.05 \quad | \quad 0.1 \quad 0.2 \quad 0.15 \quad | \quad 0.12 \quad 0.28 \\
 \text{Indices : } 0.05 \quad | \quad 0.15 \quad 0.35 \quad 0.5 \quad | \quad 0.62
 \end{array}$$

After crossover, the indices within the swapped regions (differentiated by square brackets and vertical bars for clarity) change while the separations remain constant. This becomes more of an issue in the subsequent chapter and is discussed there.

So, recapping, a variable length chromosome has been developed by making gene indices explicit and allowing them to take on real values. The mechanics of two point crossover are retained by extending crossover ranges to the real valued indices. The developed representation is now tested on a variety of test problems to determine its validity and effectiveness in variable complexity search.

4.4 Test Problems

Two manufactured problems are used as initial tests of the developed variable length representation, henceforth referred to as VLR. Both test problems are solved with the same type of EC implementation, which is described subsequently.

Because the problems act on binary strings, initialization of starting individuals is done by randomly selecting a binary value for each gene. Chromosome lengths are initialized using an uniform distribution from 5 to 50. Indices are chosen uniformly at random from the range $[0, 1]$, and then transformed into separation values. Mutation is chosen as the bit flip operator and occurs with probability $1/l$ where l is chromosome length. A $(\mu + \lambda)$ selection scheme is used where the number of parents, μ , is 15 and the number of offspring, λ , is 90. In every generation, each parent generates six offspring. The variation operators for generating offspring are insertion, deletion, and crossover. Insertion and deletion add or remove a gene at random. The 15 best individuals from the combined pool of parents and offspring are then selected for survival to the next generation. Termination occurs when the global optimum is reached. Each test problem is presented and discussed separately.

4.4.1 Proof of Concept

The first problem is maximization of the function

$$f = L - |L - l| \quad (4.2)$$

where f is the fitness value, L is the optimal length, and l is the chromosome length. Figure 4.1 shows the function for $L = 50$. This problem is an unimodal function of length and serves as a proof of concept tool where only the correct length needs to be evolved.

Results for $L = 100$ are shown in Table 4.1 and were averaged over 30 runs each. The **Operators** heading indicates which variation operators were used to create the 6 offspring. The first number indicates the number of offspring created through insertion, the second deletion, and the third crossover. As an example, (0, 0, 6) indicates that only crossover was used to produce offspring. Each combination of variation operators was able to converge to the correct length even though the initial length range was set to $[5, 50]$. Use of insertion and deletion operators results in decreased performance when compared to strict use of crossover. This is because crossover is able to accommodate a larger variety of length changes than insertion or deletion. Although multi-

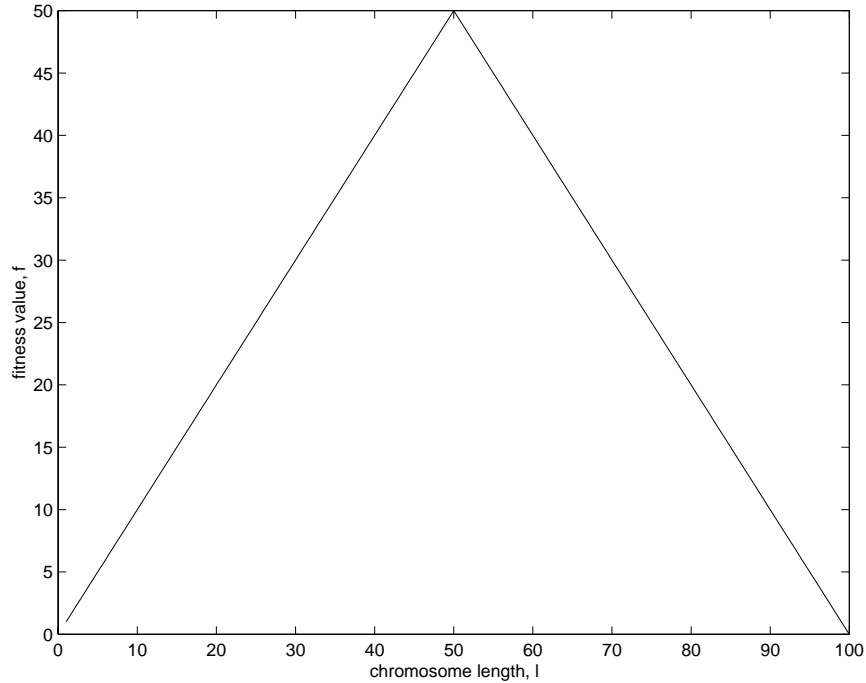


Figure 4.1: Test Problem 1 with $L = 50$. Proof of concept test case for variable length search.

| Operators | μ | σ | Median |
|------------------|-------|----------|---------------|
| 1,1,3 | 16.37 | 1.38 | 16 |
| 0,0,6 | 14.83 | 0.99 | 15 |
| 3,3,0 | 53.17 | 2.12 | 53 |

Table 4.1: Iterations to convergence for Test Problem 1 and $L = 100$ for different operator combinations.

ple insertions and deletions can be used to create a single offspring, the number typically needs to be specified in advance, which is often difficult to do. In contrast, crossover can adapt its length changes as evolution proceeds by evolving strings of certain lengths. This is a useful characteristic and requires more confirmation than this simple length matching function.

4.4.2 Trade-Off Problem

The second test problem seeks to maximize the following function

$$f = \begin{cases} n & \text{if } l \leq L \\ n - 2(l - L) & \text{otherwise} \end{cases} \quad (4.3)$$

| Operators | μ | σ | Median |
|-----------|-------|----------|--------|
| 1,1,3 | 10.27 | 1.87 | 10 |
| 0,0,6 | 9.37 | 1.75 | 9 |
| 3,3,0 | 27.03 | 7.28 | 26 |

Table 4.2: Iterations to convergence for Test Problem 2 and $L = 25$ for different operator combinations.

where n is the number of ones in the bit string and the other variables are as defined in Equation 4.2. The case for $L = 25$ is shown in Figure 4.2. This is a more realistic problem than the proof of concept because there is a functional dependency on both chromosome length and gene values.

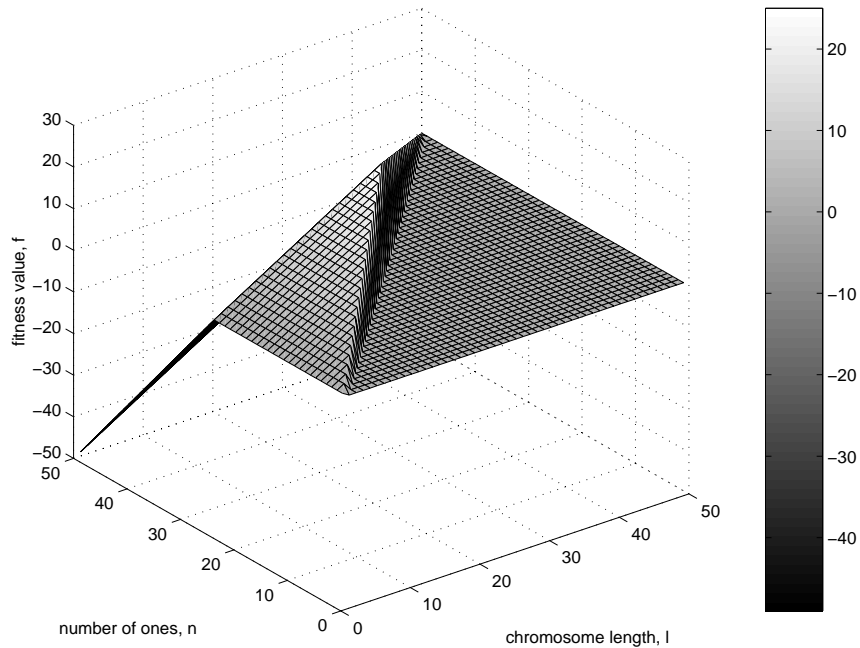


Figure 4.2: Test Problem 2 with $L = 50$. Trade-off test case for variable length search.

Results for $L = 25$ are shown in Table 4.2 and were averaged over 30 runs each. As in the proof of concept case, VLR was able to effectively evolve correct lengths. In addition, the optimal bit strings (all 1s) were evolved. Results with crossover as the only variation operator were again more rapid at converging to the correct solution. These results and those for the first test function indicate that crossover as the length changing operator are useful in cases in which the optimal length is not constrained to a small interval. The pointwise operation of insertion and deletion is incapable of rapid exploration of varying complexity search spaces.

4.5 Design Problems

Two interesting design problems with unambiguously defined variable dimensional search spaces are selected for further investigation of the developed VLR. One is the design of pattern classifiers and the other is design of 2-D polygons. Details and results of the VLR on both design tasks are presented as follows.

4.5.1 Design of a Pattern Classifier

INTRODUCTION

Pattern classification is the process of inferring a class given a set of observations [78]. Often the classes are known ahead of time as in the case of handwriting recognition. For handwriting recognition, the classes are letters of the alphabet and the data observations are digital captures of handwritten letters. However, in just as many instances, the classes are not *a priori* known and need to be discovered from the observed data. The discovered patterns can then be used for classification. For example, imagine that a census was taken on the heights and weights of people in the U.S. and in the U.K. in English units and metric units respectively. But, by some sort of unfortunate mishap, all knowledge of where the data came from was lost. Well, if each individual's height and weight were plotted, a distinct clustering would be seen with the metric units clustered together and the English units in another cluster. Thus, any new data from an unknown location can be classified according to the most similar cluster. The problem boils down to how to discovering the patterns or cluster centers for classification purposes.

Although there are many different types of pattern discovery and classification techniques, vector quantization (often referred to as partitional clustering as well) is addressed here. The name derives from the compression of the observed data vectors, $(\vec{x}_1, \dots, \vec{x}_n)$, into a set of representative vectors, $(\vec{c}_1, \dots, \vec{c}_m)$, with $m < n$. These quantized vectors are the class archetypes and any observed data vector is classified as belonging to the closest quantized vector. This classification scheme results in a Voronoi tessellation of the data space in which the space is partitioned into regions of closest similarity to the quantized vector (also known as the cluster center)². Thus, the design problem is to determine the optimal quantized vectors according to a distance metric. In addition, because the relationships and classes of the observed data are unknown in advance, the

²The partition boundaries are then equidistant to two cluster centers while partition vertices are equidistant to three or more cluster centers.

number of classes, or quantized vectors, is unknown as well. So, the design problem requires both the design of number of vectors and vector locations.

The work in this section was reported first in [54] and adheres to the structure of that paper. First, a brief review of previous work is given. This is followed by a detailed description of the EC implementation using the VLR. Results and two test cases are shown and discussed. The section concludes with comments on future research.

PREVIOUS WORK

The most common approach to finding cluster centers, equivalently quantized vectors, is the k-means clustering algorithm, which is a gradient descent type search algorithm. In addition to having the drawback of susceptibility to local optima, the k-means algorithm cannot be used to determine the number of classes. The number is assumed to be *a priori* known.

Early evolutionary approaches as reported in [40] remediated problems of susceptibility to local optima, but retained the property of fixed numbers of classes. This was also the case for Franti's work in [32]. Later work in [36] introduced variable number of classes in an evolutionary programming implementation. The number of classes is modified by simple insertion and deletion operators that add or remove a random number of cluster centers. Results of the algorithm were promising, but the number of clusters was relatively small (≤ 5). Moreover, the presence of crossover as a search operator is absent. Reviews of pattern recognition systems can be found in [16, 78].

IMPLEMENTATION DETAILS

The proposed EC implementation is an evolution strategy (ES) that was developed for the design of optimal cluster centers for 2-D spatial data. Although the developed ES is quickly generalized to higher dimensional spaces, 2-D spaces are graphically easy to visualize and interpret.

Encoding Scheme

In accordance with the problem's 2-D nature, each gene or cluster center is an ordered pair of real numbers denoted by (x, y) . Individual genes are concatenated into a chromosome and are ordered within the chromosome by ascending x value. So, for example, a three class chromosome with $x_1 \leq x_2 \leq x_3$ would be

$$[(x_1, y_1), (x_2, y_2), (x_3, y_3)]$$

Also encoded are self-adaptive mutation parameters for each data dimension, which is typical of evolution strategies.

Initialization

Initialization of the starting population is given in pseudo code as

```

for  $i = 1 : n_{pop}$ ,
    select  $l_i \in l_{range}$ 
    for  $j = 1 : l_i$ ,
        select  $c_j \in c_{range}$ 
    end
    order  $\vec{c}$ 
end

```

where n_{pop} is population size, l_i is chromosome length or number of classes, l_{range} is possible values of l_i and is set at $[10, 35]$, c_j is cluster j , c_{range} is *a priori* specified data bounds, and \vec{c} is the chromosome or entire encoding for the vector quantization. Although the range of class numbers is specified in advance, there are no restrictions on how many or how few clusters can be evolved through the action of crossover.

Variation Operators

Mutation follows the guidelines of standard ES in which a Gaussian random variable with zero mean and standard deviation σ_x or σ_y perturbs both the x and y values of every cluster center/gene. σ_x and σ_y are the self-adaptive mutation parameters and are adjusted as described in [80].

Insertion and deletion operators are ignored because the developed VLR and corresponding crossover operator can duplicate the effects of both. The x values or first coordinate in the ordered pair serve as the gene indices. Hence, genes are swapped if their x coordinate lies within the specified crossover range. Simultaneous insertion and deletion occurs when the crossover range only covers genes in one parent.

Selection Strategy

A (10+60) ES selection strategy is used, where 10 indicates the number of parents or individuals in the population and 60 is the number of offspring produced each generation. The 10 best individuals from the combined pool of parents and offspring are selected for survival each generation (denoted by the '+'). Offspring are created by mutating and recombining each parent twice. So, a total of six offspring is generated per parent genome per generation.

Fitness Evaluation

A modification of the Mean Square Error (MSE) clustering measure is chosen as the fitness function. The MSE is expressed as

$$\text{MSE fitness} = \frac{\sum_{i=1}^l \sum_{j=1}^{m_i} d(c_i, x_j^i)}{n} \quad (4.4)$$

where l is the number of clusters, c_i is the i th cluster center, m_i is the number of data points belonging to the i th cluster, x_j^i is the j th data point belonging to the i th cluster, n is the total number of data observations, and $d(a, b)$ is the squared euclidean distance between points a and b . Data points are assigned membership to the cluster with the nearest cluster center as stated previously.

Unfortunately, the above fitness function is poorly suited for comparing clusterings that have different numbers of classes. This can be seen by imagining the relationship between optimal MSE and the number of classes. Because the optimal MSE can always be decreased by adding an observed data point as a cluster center, fitness is a monotonically decreasing function of cluster number. Obviously, the best clustering is not *simply* the data set (since no new information is gained if this is the case). The problem is that there needs to be a penalty for model complexity (*i.e.*, number of clusters). This tradeoff between model complexity and goodness of fit is well known in function approximation and learning system fields (also known as the principle of parsimony or Occam's razor) [16].

A variety of methods have been developed to address the problem of model complexity. Two of the more prevalent methods, for clustering at least, are the Davies and Bouldin index [19] and the principle of Minimum Description Length, or MDL [36]. The Davies and Bouldin index requires the calculation of the clustering scatter or variation in addition to the distances between cluster centers

and their data members. MDL also necessitates extra calculations, as seen by its formulation

$$\text{MDL}(\omega) = -\log_2(f(\mathbf{x}|\omega)) + \frac{1}{2}n \log_2 |\mathbf{x}| \quad (4.5)$$

where \mathbf{x} is the observed data, $f(\mathbf{x}|\omega)$ is the conditional likelihood function given parameter vector ω , n is the number of parameters, and $|\mathbf{x}|$ is the number of data points.

Instead of using either of the above fitness measures, a heuristic measure is used here, since both the Davies and Bouldin index and the MDL principle require too much overhead when compared to MSE. So, a heuristic MSE is chosen and is given by

$$\text{MSE heuristic fitness} = \sqrt{n+1} \frac{\sum_{i=1}^n \sum_{j=1}^{m_i} d(c_i, x_j^i)}{n} \quad (4.6)$$

The heuristic penalizes model complexity by multiplying the MSE fitness by a constant proportional to the square root of the number of clusters. The penalization factor was chosen primarily because it provided good clustering results for a variety of data sets. However, the effectiveness of the penalization term is not without basis. To see this, recall the relationship between MSE fitness and number of classes. The curve exhibits a point where the slope changes dramatically since the addition of new cluster centers after this point has little effect on the MSE fitness measure. Intuitively, the optimal clustering occurs at this ‘knee’, as it provides the best performance per unit change in cluster number. By multiplying the curve by some function of cluster number, the ‘knee’ can be made into a minimum. The chosen penalization factor seems to be one such function. However, these functions are dependent on ‘knee’ location and curvature making the choice of such a function difficult.

Termination

The evolution strategy is terminated either when the number of iterations exceeds 200 or when the average population fitness varies less than 0.001 from one generation to the next. As with the fitness penalization factor, the maximum number of iterations is chosen empirically. In most cases, after 200 generations, the self-adaptive mutation parameters are so small that significant improvements in the heuristic MSE no longer occur.

RESULTS

The proposed evolution strategy is tested on two data sets generated from noisy perturbations about known cluster centers. In both instances, twenty vectors are used to generate the test data. For each of these cluster centers, fifty data points are distributed around the vector according to a Gaussian with random variance. The first data set has no overlapping data or so-called non-touching clusters. Conversely, the second data set has overlapping or touching clusters. Although overlap may take on different definitions, the distinction between non-touching and touching is clearly illustrated by the test data as shown in Figures 4.3 and 4.6. Since these data sets are representative of nearly all clustered data sets, the performance of the proposed ES will be indicative of its effectiveness in designing vector quantizations. In each case, results are averaged over 11 runs.

Non-touching clusters

The data generated for the non-touching case are shown in Figure 4.3. Also shown in this figure is the partitioning as defined by nearest cluster center. An example of a typical evolved clustering is shown in Figure 4.4. Comparison of the known and evolved cluster centers shows good agreement as seen in Figure 4.5.

The fitness measure of the test data using the known cluster centers is 84.7 (see Equation 4.6). This compares to an average evolved fitness of 135.1 with a standard deviation of 95.2. The average number of evolved clusters was 19.4 with a standard deviation of 3.3. These results are promising. In particular, the best evolved solution had a fitness of 84.7 and 21 clusters. As was the case whenever an evolved solution had more than 20 clusters, the ‘extra’ cluster in this case was redundant having no assigned data members. Thus, if the redundant cluster is deleted, the best fitness is in fact lower than the ‘true’ fitness of 84.7.

The results are marred by a single evolved solution that performed significantly worse than the others. The fitness of this individual was 414.602 and the number of clusters was 10.

Touching clusters

Figures 4.6–4.8 are the touching data set equivalents of the non-touching data results.

The fitness measure of the test data using the known cluster centers is 245.9. The average evolved clustering fitness was 251.9, only marginally worse than the ‘true’ value, with a standard deviation of 18.4. However, the average number of clusters was 13.6, significantly lower than the

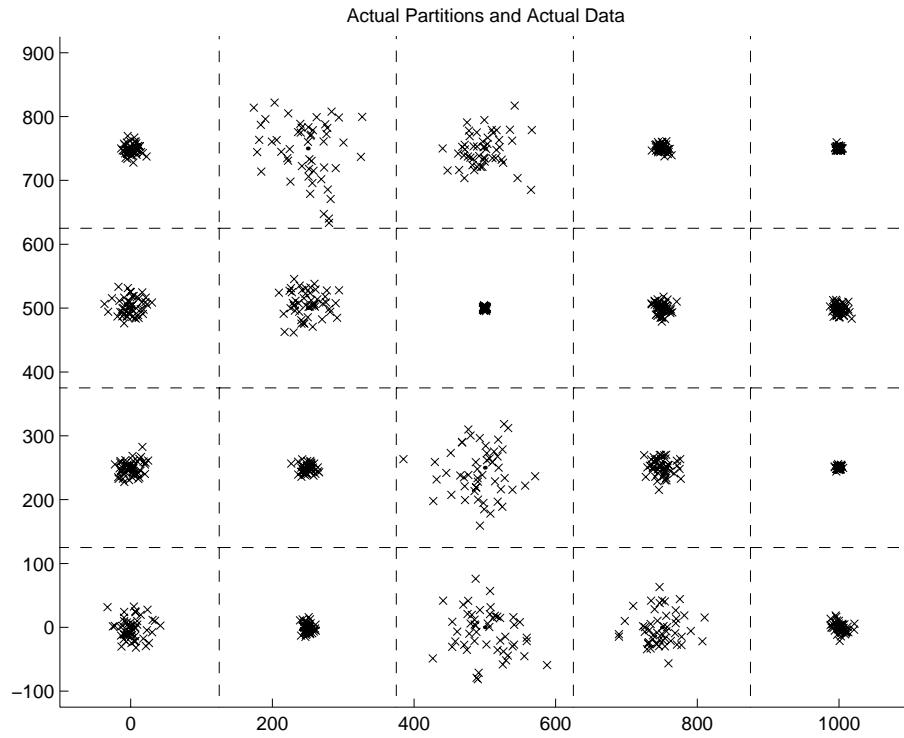


Figure 4.3: Non-touching cluster data set and corresponding partitions.

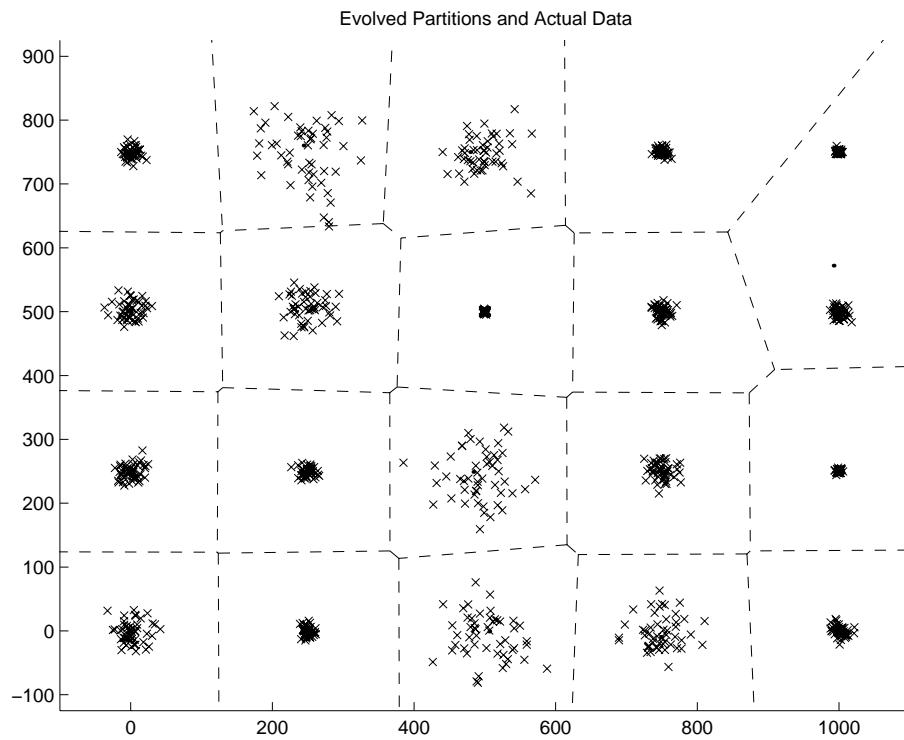


Figure 4.4: Non-touching cluster data set and typical evolved partitions.

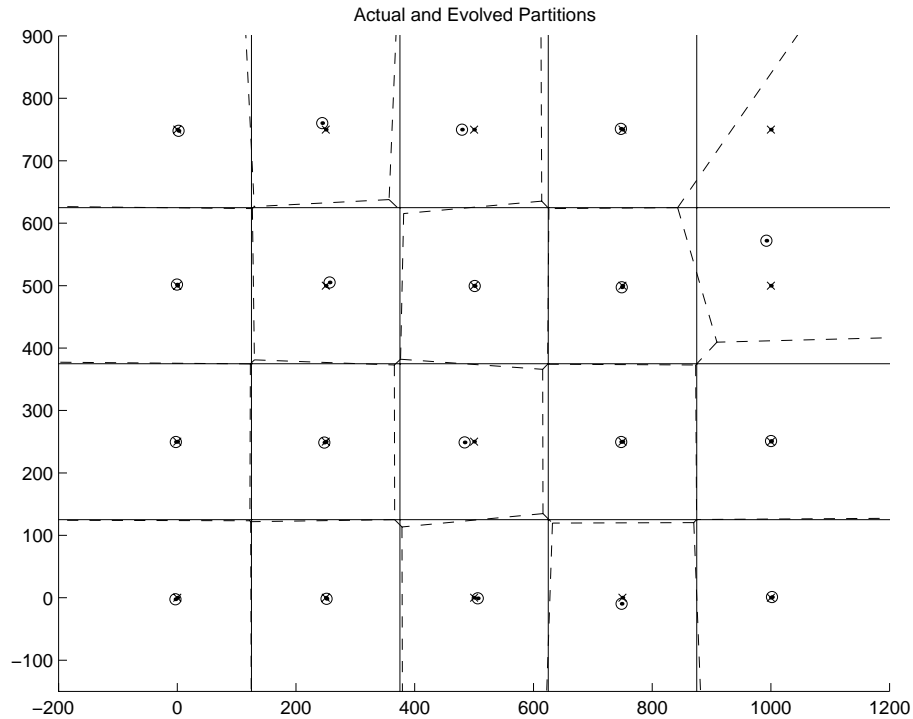


Figure 4.5: Actual (—) and typical evolved partitions (- -) for non-touching case. Actual (×) and typical evolved (○) cluster centers.

actual cluster number, with a standard deviation of 3.2. This is expected since several of the actual clusters overlap, leading to more ambiguous clusters. In any event, for more than half of the ES runs, the evolved solution was able to better the actual clustering's fitness. As compared to the non-touching case, evolved clusterings for the touching data never resulted in redundant clusters nor were there any outliers in the evolved clusterings. The best evolved clustering had 17 clusters and a fitness of 234.5. Conversely, the worst evolved solution had 7 clusters and a fitness of 285.5.

DISCUSSION

When the MSE and heuristic MSE fitness functions are minimized, the data partitioning is done in a nearest neighbor approach. So, the resultant partitioning is simply the Voronoi diagram whose Voronoi centers are the cluster centers.

The results of the evolved clusterings are highly intuitive. For example, in the non-touching case, an evolved cluster center is usually farther from an actual cluster center when that cluster has more scatter. This can be seen by comparing Figure 4.5 to Figures 4.3 or 4.4.

In the touching case, several of the actual cluster centers are easily identified by the evolution

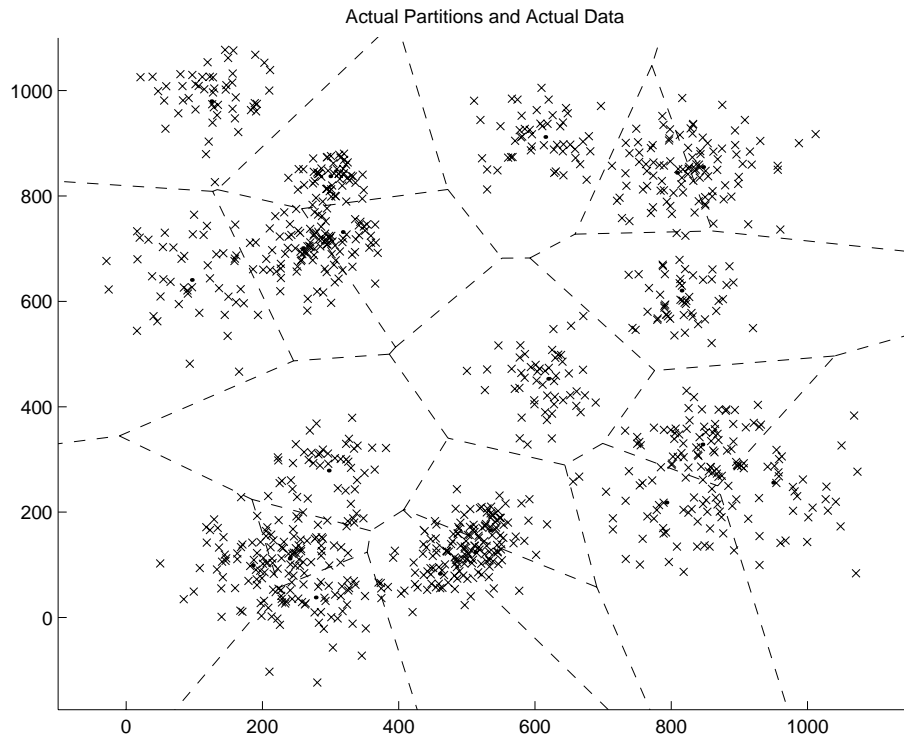


Figure 4.6: Touching cluster data set and corresponding partitions.

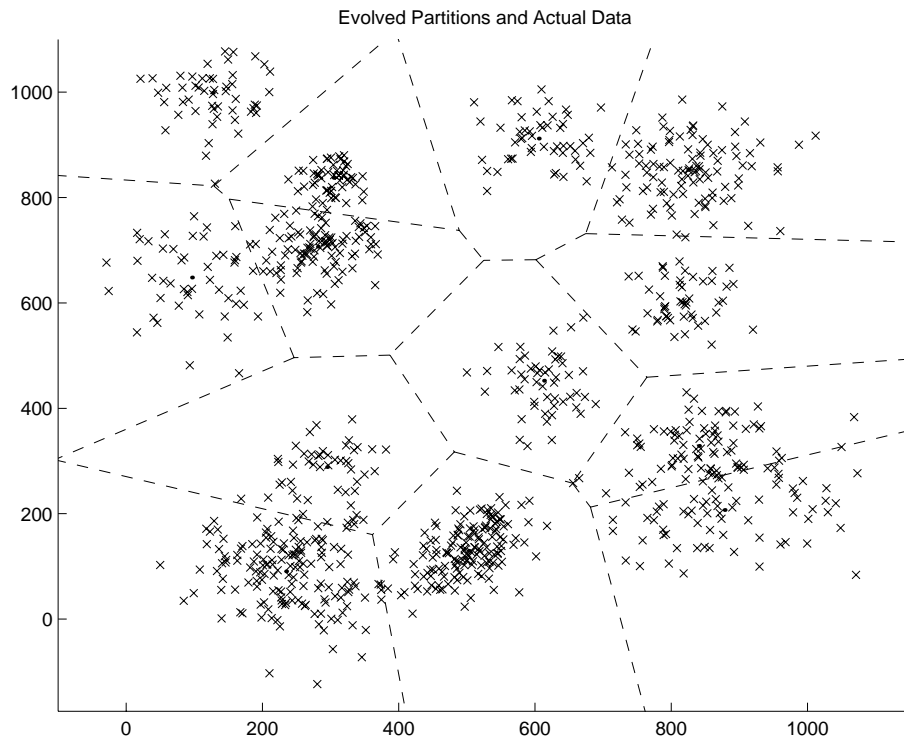


Figure 4.7: Touching cluster data set and typical evolved partitions.

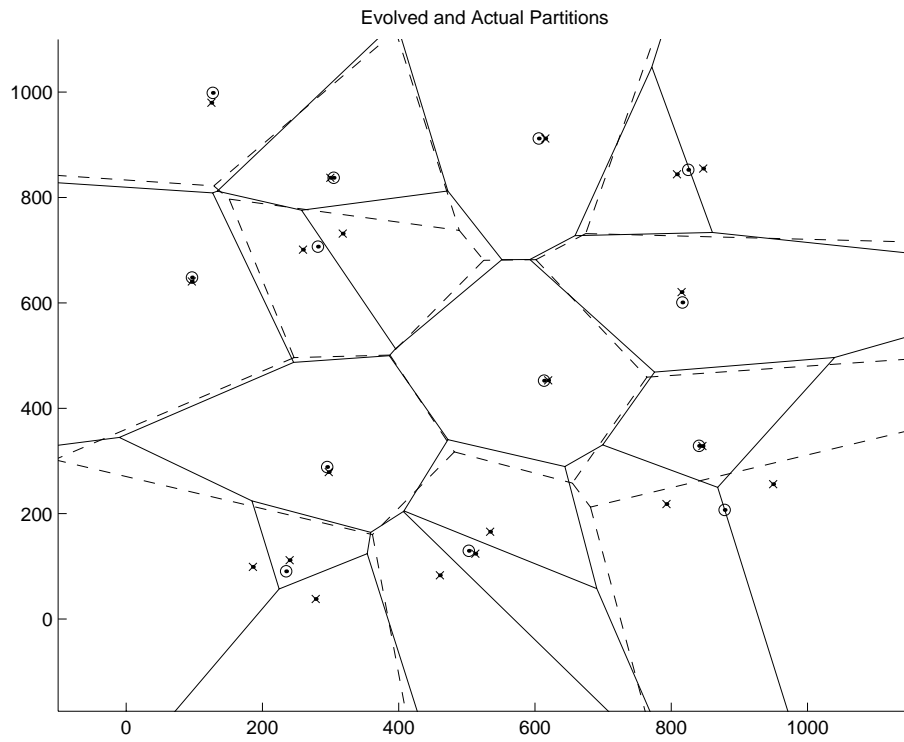


Figure 4.8: Actual (—) and evolved partitions (---) for touching case. Actual (×) and typical evolved (⊙) cluster centers.

strategy. Generally, these cluster centers were the ones with minimal overlap. The cluster centers that did exhibit overlap, though, were often combined into a single cluster by the evolution strategy. As expected, the evolved cluster centers in these instances are close to the mean of the actual cluster centers; and, hence, are close to the Voronoi edges of the ‘true’ partitions as seen in Figure 4.8.

CONCLUSION AND FUTURE WORK

A novel evolution strategy implementing variable length chromosomes has been developed for vector quantization design. Results on 2-D spatial data are promising, as the evolved clusterings often surpassed the fitness measures of the ‘true’ clusterings for both the non-touching and touching test cases.

A number of issues regarding the proposed ES remain. In particular, the effectiveness of the proposed ES on data with dimensions greater than two needs to be addressed. Since the ES implementation is easily generalizable to any dimension, it is expected that the ES should remain effective as data dimension increases. However, the use of two point crossover may limit algorithm performance. The reason is that epistasis, or linkage, between good cluster centers probably will not be between adjacent genes in the chromosome. Thus, research on uniform crossover operators that can sidestep this problem is necessary.

A further issue is that of the fitness function. Since evolutionary algorithms are highly modular, any fitness function can be substituted in place of the MSE heuristic fitness measure used. Modification of the fitness function will result in new types of partitionings such as hyper-ellipsoids (as opposed to Voronoi diagrams). Since the proposed ES is developed as a general framework for vector quantization, it would be interesting to observe its performance using different fitness criteria. Also, and perhaps most importantly, a comparison of the dynamic partitioning approach taken here with other approaches is needed.

4.5.2 Design of 2-D Polygons

INTRODUCTION

Shape design is a vital part of engineering design because functionality is strongly dependent on shape. Space limitations and other constraints further affect shape design making it an even more difficult problem. Often the shape design problem can be reduced to the design of 2-D polygons. Examples are the design of cross sections for aerodynamic structures and heat sinks. In this section,

the design of 2-D polygons is addressed. Rather than developing a simulation tool to evaluate evolved shapes for particular functions (*i.e.*, heat dissipation characteristics), shapes are designed to a target shape. The evaluation function is then simply a heuristic that measures the closeness between two 2-D polygons.

The remainder of this section is structured to (i) briefly review related work, (ii) describe the implementation details, (iii) present and discuss results, and (iv) conclude with comments on future research. Much of this work can be found in [55].

PREVIOUS WORK

A fair amount of work has been done on shape design via evolutionary methods. Early work discretized shape space into grids of binary elements that were either full or void of material [13, 14, 49]. Truss structures and flexures were synthesized using this approach. Others worked on voxel like representations to design aerodynamic structures and heat sinks [10]. Similar to this voxel work was the work done on Lego structures and L-systems for table design [34]. More related to the current approach was work done on evolving 2-D polygons for the design of mask-layouts for bulk wet etching processes [55, 62, 66, 67].

IMPLEMENTATION DETAILS

An evolution strategy is used to solve the problem of 2-D polygon design and it closely parallels that used in the pattern classification task.

Encoding Scheme

Each vertex of the 2-D polygon is encoded as a single gene, which is an ordered pair that consists of an angle and a radius (*i.e.*, a polar coordinate representation). A polygon is encoded through the concatenation of genes with neighboring genes in the chromosome determining the connectivity of the vertices. Index values are also stored with each gene and are computed as the fraction of the current side length to the total side length, or perimeter. The first side length is calculated *from* the positive x -axis, while the last side length is calculated *to* the positive x -axis.

For example, the square with vertices defined by the cartesian coordinates of $(\pm 25, \pm 25)$ is

encoded as

$$\begin{array}{rcccc}
 \text{Angles :} & 45 & 135 & 225 & 315 \\
 \text{Radii :} & 25\sqrt{2} & 25\sqrt{2} & 25\sqrt{2} & 25\sqrt{2} \\
 \text{Indices :} & .125 & .375 & .625 & .875
 \end{array}$$

Note that the procedure for calculating indices results in index ranges of $[0, 1]$ as desired for the extended crossover operator. Also, total perimeters are always normalized to 100.

Initialization

Initialization of the starting population substrate is accomplished in the following manner. First, the number of vertices l is chosen uniformly at random from a preset range. Then, l vertices are generated by selecting radii and angles at random. These vertices are ordered according to angle, resulting in polygons that are radially distributed (*i.e.*, only star-like polygons are allowed). In order to enlarge the space of feasible polygons, some out-of-order angle vertices are added. The number of out-of-order vertices, n_o , is chosen uniformly at random from the range $[0, l/3]$. For each of these vertices, an in-order vertex is chosen at random and the new out-of-order vertex, which is a perturbation of the in-order vertex, is placed after the in-order vertex. After all the vertices have been added, the perimeter is normalized to 100 and the indices are calculated. Although self-intersecting polygons may be generated under this procedure, their fitness values are poor for the task at hand. Consequently, no measures are taken to enforce non-self-intersecting polygons.

Variation Operators, Selection Strategy, and Termination

As with the previous design problem, canonical ES mutation operators are implemented and the developed VLR and corresponding crossover operator are used. Likewise, a (10+60) selection strategy is chosen. Furthermore, because it is highly unlikely that the globally optimal fitness of zero can be achieved, termination occurs either when 500 generations are reached or if the average population fitness does not vary more than 0.0001 from the previous generation.

Fitness

Design of shapes to a target shape requires quantification of the similarity between two shapes. A heuristic measure is adopted here in favor of more stringent and complex measures such as the L2

norm defined in [4]. Fitness evaluation has the analytic form

$$f = \sum_{j=1}^N \min_{i \in l+n_o} d(t_j, p_i) \quad (4.7)$$

where f is the fitness value, N is the number of vertices in the target shape, l is the number of in-order vertices, n_o is number of out-of-order vertices, t_j is target shape's j vertex, $d(\cdot)$ is the Euclidean distance norm, and p_i is the test shape's i vertex. The fitness function is further penalized in proportion to the number of repeated t_j and t_j that are matched in the incorrect order. For example, say that the t_j s closest to p_1 , p_2 , and p_3 are t_5 , t_5 , and t_2 , respectively. Then, because t_5 is repeated twice, fitness is penalized by multiplying by two. Similarly, because the t_j aren't in order an additional penalization occurs. Furthermore, penalties are incurred when any t_j is left unmatched. Again, these penalties are in proportion to the number of unmatched target vertices. So, smaller fitnesses denote more similar shapes. Furthermore, from experimental results, it is known that fitnesses below 100 indicate qualitatively equivalent shapes.

RESULTS

2-D shape design was conducted with a variety of target shapes, all of which exhibited similar performance. Results of a typical run are shown in Figures 4.9–4.11. At least qualitatively it seems that shape design is quite effective because convergence to the target shape occurs at around 60 generations. Beyond this point, only small shape changes occur because of the decreasing self-adaptive mutation rates.

Results are averaged over 25 runs for four different initial size ranges (*i.e.*, number of vertices). These are shown in Table 4.3. The target shape is non-convex, non-radially distributed, and has 10 vertices. **Range** indicates the the initial size ranges of the starting polygons (*i.e.*, size is the number of sides). **Gens** indicates the first generation at which the correct number of vertices was reached, while **Fitness** indicates the fitness of the final evolved shape. The results show that the correct number of vertices is found rapidly with mean generation times below 6. This further confirms the capabilities of VLR for variable complexity search. There are a few other trends that are observed from the results. The initial range 3-6, which does not include the target size, takes longer to converge than those ranges that include it. However, as the initial range grows too large, the search takes more time to converge as expected because the initial solutions are further away from the target shape. The fitness of the final evolved solutions follow a similar pattern. These results indicate that

excessively large initial ranges are detrimental, as expected.

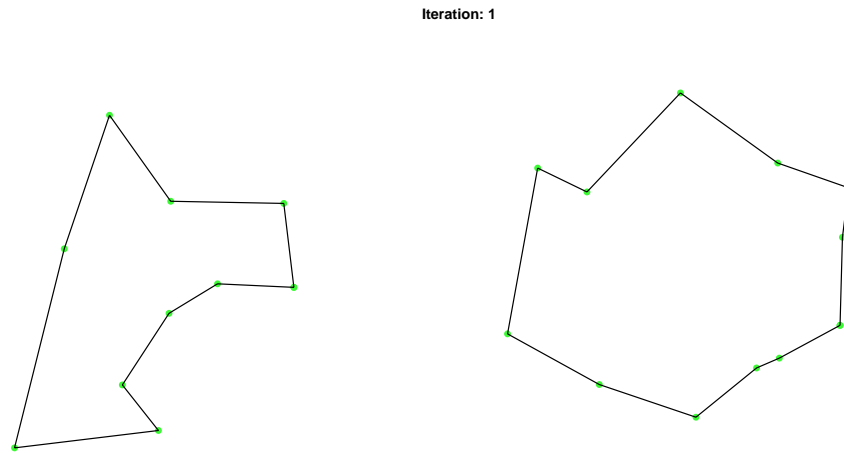


Figure 4.9: Target Shape and Best Shape for Generation 1

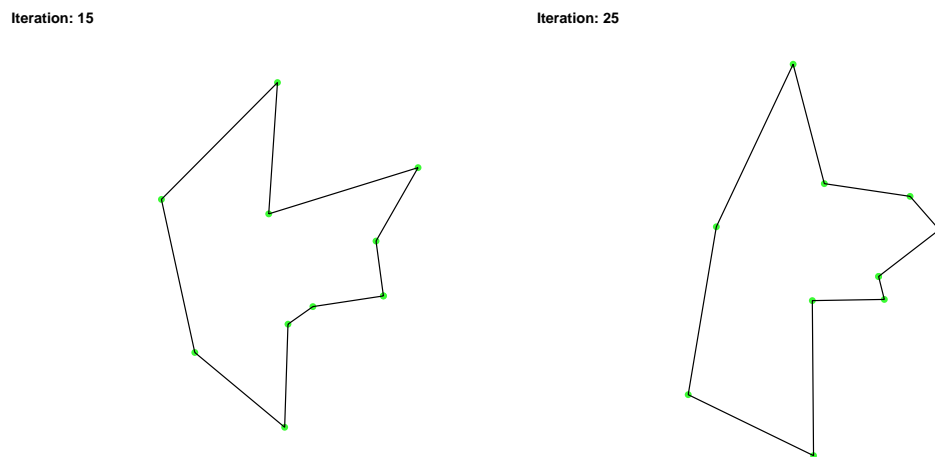
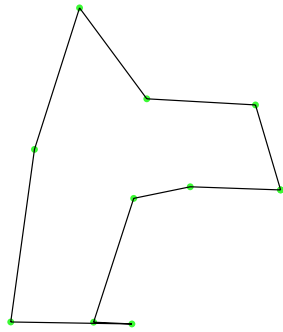


Figure 4.10: Best Shapes for Generation 15 and 25

CONCLUSION AND FUTURE WORK

The developed VLR was successfully applied to 2-D polygon design. Although 2-D shape design is not necessarily a real world problem, this work paves the road for more applicable shape design through integration of shape functionality simulators. This remains the most urgent task for future research. Nonetheless, the results of this work point to the ability of the developed VLR for

Iteration: 65



Iteration: 105

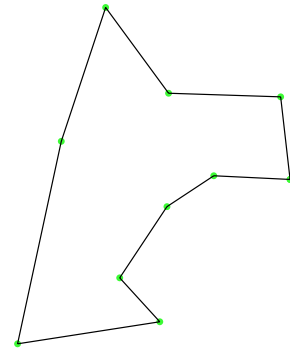


Figure 4.11: Best shapes at generations 65 and 105.

| Range | Gens. | σ | Fitness | σ |
|--------------|--------------|----------|----------------|----------|
| 3-6 | 5.09 | 1.41 | 209.71 | 181.96 |
| 3-10 | 2.79 | 0.97 | 192.99 | 174.63 |
| 3-28 | 4.08 | 1.66 | 110.04 | 39.96 |
| 3-100 | 5.91 | 2.66 | 175.41 | 149.67 |

Table 4.3: Generations to length convergence and final fitness values for different initial ranges.

successful evolution of 2-D shapes and real world design problems.

4.6 Summary and Discussion

A simple extension is made to canonical chromosomes to enable variable length search and therefore variable complexity design synthesis. The crossover indices are made explicit and real valued rather than the typically used integer values that are encoded implicitly by gene position. In combination with randomly initialized indices, this results in variable length representations (VLR). VLR is successfully applied to a suite of test problems generated to test its validity. Positive results led to application of the VLR to two real design problems, that of pattern classifiers and 2-D polygon design. Again, positive results are obtained, pointing to the effectiveness of VLR for design problems with variable dimensionality search spaces.

Closer inspection of the developed VLR reveal some interesting observations. The crossover range is chosen uniformly at random from the encompassed chromosome index ranges (typically $[0, 1]$), but the indices are not uniformly distributed over this same range. The result is that genes that have smaller separations will be more likely to stay together after crossover events than those with larger separations. In other words, certain gene pairs are more tightly “linked” than others. But, why should gene pairs have different linkages? Furthermore, are there good gene linkages, and if so, how can they be evolved? These questions are answered in the next chapter on linkage learning and modularity.

In light of these emergent linkage phenomena, it can be seen that both design examples in this chapter evolved linkage characteristics through modification of indices. However, in each case the indices were tied to actual parameter values. Thus, evolution of optimal solutions constrained linkages to evolve in a certain manner, which may or may not have been optimal. The upshot of this observation is that for proper linkage evolution there must be a separation between indices and gene values. Otherwise, constraints on linkage evolution may be too restrictive.

Chapter 5

Modularity and Linkage Learning

5.1 Introduction

Although the earliest evidence for life has been dated to roughly 3.5 billion years ago, multicellular organisms did not appear until 2.5 billion years later. Another half billion years after this event, a rapid proliferation occurred in the variety and complexity of organisms¹. This period of rapid evolution became known as the Cambrian explosion. Although the reasons are still not clear, many have attributed the explosion in evolutionary development to the acquisition of hierarchical modules into the genome [24]. Genetic modularity has several key benefits. For one, well-evolved solutions are always maintained and not disrupted during crossover. Perhaps more importantly, however, is that compartmentalization of certain attributes and features enables rapid experimentation of new body types (*e.g.*, variation of number of limbs). Such reuse of genetic modules reduces evolutionary time because the “wheel does not have to be reinvented”. As described in the previous chapter, these newly developed modular genomes result in more evolvable organisms. The definition of evolvability is reiterated and given by Altenberg as “the ability of a population to produce variants fitter than any yet existing” [1].

Evolvability is integral to evolutionary computation implementations. Because most EC implementations have fixed representations and operators, it is imperative that they are chosen such that individuals have a minimal level of evolvability. For example, take the problem of software code design via evolutionary methods. Early approaches represented programs as strings of letters and generated new programs through random variation. Following Altenberg’s definition, it is obvious that populations of such individuals have little or no evolvability because fitter variants of

¹The dates and observations come primarily from dating the fossil records.

functional programs are found with near zero probability. In contrast, Koza's genetic programming (GP) represented programs as trees and used subtree crossover to generate offspring programs. This representation displays an adequate level of evolvability, which has been verified through experimentation. Unfortunately, there is no metric of evolvability without prior knowledge of the search space; consequently, it is often necessary for experts to help in the application of EC. Still, lessened reliance on human interaction is desirable and evolution of evolvable solutions can be made to occur. Increased evolvability, as mentioned already, leads to more efficient and faster searches. This chapter builds on the work developed in [56] on the so-called linkage learning and modularity problem.

The chapter introduces a biologically plausible representation based on non-coding segments for evolution of evolvable representations in evolutionary computation. This representation is a simple extension of the variable length representation introduced in Chapter 4. The remainder of this chapter is organized to: (i) review previous related work, in particular the use of non-coding segments in genetic algorithms, (ii) develop a non-fixed length non-coding segment representation, (iii) extend the non-coding segment representation to a coding, range representation, (iv) present results of the non-coding segment representation on a set of Royal Road functions with clearly defined modularities, (v) present results for two real design problems, truss and neural network design, and (vi) conclude with a summary and discussion of future work.

5.2 Previous Work

Much work exists on the online evolution of evolvability²(*i.e.*, evolution of more evolvable solutions during an EC run) and is too extensive for thorough discussion here. Instead a representative set is reviewed. A classic example of adaptive evolvability³ are the self-adaptive mutation operators found in evolution strategies [5]. Self-adaptation of search parameters allows mutation to adjust its step size, or search radius, according to feedback from previously explored points of the performance landscape. On the otherhand, messy genetic algorithms (mGA) use self-adaptive representations to increase evolvability. mGA genes are tagged with gene markers, much like indices, that identify the gene's functionality. The combination of crossover and random initial orderings of gene markers results in representations in which any ordering of the genes can be obtained. For example, the first gene could be adjacent to the fourth gene. This is beneficial if the first gene and fourth gene

²Although most work is not classified as evolving evolvability.

³Here, adaptive is taken to mean online evolution in contrast to the definition of adaptation given in Chapter 3.

are highly dependent on each other because crossover is less likely to disrupt the two when they are adjacent than when they are separated by two genes as in a canonical representation. The probability of disruption is quantified as the *linkage* between any two genes. Tightly linked genes are those that have low probabilities of disruption, while loosely linked genes have high probabilities of being separated by a crossover operation. Typically, adjacent genes are the most tightly linked and this mimics nature. However, in contrast to nature, typical EC linkages between adjacent genes are fixed over all adjacent gene pairs (*i.e.*, disruption probability is equal for all adjacent gene pairs). Biological chromosomes have distinctly unequal linkage characteristics for different gene pairs. Although the reasons for this variation are not well known, some have attributed it to the presence of non-coding, or 'junk', DNA segments in biological genomes [90].

5.2.1 Non-Coding DNA Segments

The majority of DNA in organisms is non-coding, meaning that these segments are not transcribed into RNA for protein synthesis nor do they have any known genomic regulatory functions. Their presence is a modern day mystery, but it has been hypothesized that they have evolved to prevent the disruption of good building blocks during a crossover event. After all, if crossover is equally likely to occur in both coding and non-coding segments, then non-coding segments can act as buffers to the disruption of well evolved coding segments. The longer the non-coding regions are, the more likely crossover will occur in the non-coding regions. Likewise, genes separated by no intergenic non-coding segments are less likely to be disrupted than genes separated by long intergenic non-coding sequences. Thus, modular genetic structures can arise as a result of variations in non-coding segment length. Because modularity leads to increased evolvability and increased evolvability further leads to more efficient search, many researchers have studied the effect of non-coding segments on linkage/modularity learning.

Levenick [60], Forrest [31], and Wu [89, 91] all implement similar non-coding segment representations in which a fixed number of non-coding genes are inserted between coding genes within a chromosome. In each work, the representation is tested on modular fitness functions whose modules, or building blocks, are known. The researchers were then able to modularize the chromosomes correctly by inserting non-coding segments in the proper proportion between genes. Issues of how to determine proper placement and length of non-coding sections without prior knowledge of the true linkage characteristics were largely left unanswered.

Later works by Lobo [65] and Levenick [61] were able to resolve these issues to some degree.

Furthermore, non-coding segments or bloat has been a vigorous research topic in the genetic programming community and is too lengthy to cite. Lobo’s work has some interesting parallels to that in this chapter, which seeks to provide efficient linkage learning as an extension of the variable length representation developed in Chapter 4. Other work on evolvability, linkage learning, and modularity can be found in [2, 42, 46, 72, 83, 71, 74].

5.3 Development of Linkage Learning and Modularity

5.3.1 Non-Coding Segments

Non-coding segments can be introduced into EC in a variety of ways. Initially, the approach taken is to add a non-coding gene between each coding gene⁴. The non-coding genes have positive real values that indicate their length. An example is shown below for a chromosome with four coding genes denoted by g_i and the corresponding non-coding genes by n_i .

$$n_1 \quad g_1 \quad n_2 \quad g_2 \quad n_3 \quad g_3 \quad n_4 \quad g_4 \quad n_5$$

It is almost immediately apparent that such an approach is equivalent to the separation encoding given in Chapter 4 for variable length representations. Although the non-coding genes are coded inline with the rest of the chromosome, they are equivalent to the separation values. So the VLR crossover operator is adopted with modifications if the search space is of fixed complexity (*i.e.*, fixed length chromosomes). Figure 5.1 shows the non-coding representation graphically. It is easy to see that genes 2 and 3 are tightly linked because the intergenic non-coding segment between these two is much smaller than all other segments.

Revisiting VLR crossover, non-coding genes can be summed to determine the indices of each coding gene. The non-coding lengths are normalized such that the indices vary between 0 and 1. Crossover occurs by selecting a range uniformly from $[0, 1]$ and swapping coding genes with indices in this range between parent chromosomes. This presents a problem for fixed length chromosomes that have different non-coding lengths. The solution is to simply choose a range for one parent, locate the genes within this range for the selected parent, then swap the genes with those in the second parent with the same gene positions. The swap includes the non-coding segments; so well

⁴Hopefully, this misuse of “gene” is not confusing as the non-coding genes should really be named non-coding segments.



Figure 5.1: Diagram of the non-coding representation.

evolved lengths can be transmitted to other individuals. The probability that two genes at the i and j positions are kept together after crossover is given by the relation

$$p_{ij} = \left(\frac{\sum_{k=1}^i n_k}{N+1} \right) \cdot \left(\frac{\sum_{m=j+1}^{N+1} n_m}{N+1} \right) \quad (5.1)$$

where p is the probability and N is the number of coding genes. The term in the first parenthesis is the probability of choosing the crossover point before gene i , while the second is the probability for selecting a crossover point after gene j . This probability can be used to quantify linkage between any pair of genes.

An additional mutation operator is developed specifically for evolution of non-coding genes. Mutation adds a random value taken from a Gaussian distribution to each non-coding gene. This perturbation allows new lengths or linkage characteristics to be discovered. Of course, this must occur prior to offspring generation, otherwise selection does not act on the newly modified non-coding genes.

The hypothesis is that if parent chromosomes exhibit good linkage characteristics (obviously

these are tied to both coding and non-coding genes), then it will have a higher probability of survival than chromosomes without good linkage characteristics but with the same coding genes. The reason is that the first parent's offspring will be more likely to retain the good building blocks and hence have a better chance of survival.

5.3.2 Coding Segments

The non-coding approach is extended to a more flexible representation in which the coding gene lengths are stored. A closer look at the non-coding or index approaches reveals that by encoding the index ranges of each gene, a nearly equivalent representation is achieved. For example, the first gene would have two parameters denoting its bounding indices. The aforementioned crossover operator is retained with slight changes. The only difference is that genes are swapped if their index ranges intersect the chosen crossover range. Such a representation makes n_{N+1} unnecessary and it is thrown out.

Now, consider if these ranges are allowed to overlap. In other words, imagine that consecutive genes do not have coincident starting and ending indices. Then, a much larger space of possible linkage characteristics is available for search because of the non-serial nature of the proposed representation. For example, consider a three gene chromosome with the following characteristics. The first two genes have overlapping ranges and the third gene's range contains the first two as illustrated in Figure 5.2. Hence, whenever either of the first two genes are selected from crossover, so is the third. Overlapping ranges result in parallel codings – as seen by the necessity of two dimensions needed to illustrate Figure 5.2. This parallel nature of range encodings is not possible in the serial non-coding segment approach.

Again, linkage between genes i and j can be calculated from

$$p_{ij} = \left(\frac{e_i - s_i}{\max_e - \min_s} \right) \cdot \left(\frac{e_j - s_j}{\max_e - \min_s} \right) \quad (5.2)$$

where e is the end of the index range and s is the start. The first term in parenthesis is the probability of choosing a crossover point coinciding with gene i and the second is for gene j .

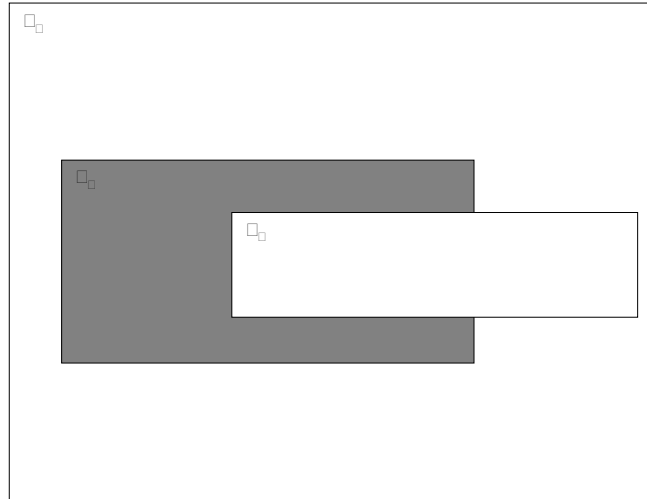


Figure 5.2: Diagram of the parallel linkage characteristics that overlapping range representations can exhibit.

5.4 Test Problem

A Royal Road function is used to test the effectiveness of linkage learning with the non-coding representation (NCR). This will also serve as a proxy for the range representation (RR) as their differences are slight, assuming RR retains serial ranges. Royal Road functions were first introduced in [70].

5.4.1 Royal Road Problem

Royal Road functions are a set of functions with hierarchial, modular fitness functions. What this means is that optimal solutions are composed of increasingly complex modules. A 64 bit, well posed, Royal Road function is used here. There are four hierarchial levels with module lengths starting at 8 and doubling at each level (*i.e.*, 8, 16, 32, to 64). Modules do not overlap within levels such that the number of modules per level decreases from 8 to 4 to 2 to 1. Each module consists entirely of 1s. For clarity, all 15 modules are shown in Figure 5.3. A shorthand notation is used where each digit actually consists of 8 bits. 1s indicate that all bits have a value of 1 and *s indicate that any 8 bit string is allowed.

The fitness function is defined according to how many of the modules are matched. The so-

```

level = 0  1 * * * * * * * *
            * 1 * * * * * * *
            * * 1 * * * * * *
            * * * 1 * * * * *
            * * * * 1 * * * *
            * * * * * 1 * * *
            * * * * * * 1 * *
            * * * * * * * 1 *
            * * * * * * * * 1

level = 1  1 1 * * * * * *
            * * 1 1 * * * *
            * * * * 1 1 * *
            * * * * * * 1 1

level = 2  1 1 1 1 * * * *
            * * * * 1 1 1 1

level = 3  1 1 1 1 1 1 1 1

```

Figure 5.3: Royal Road function modules.

called flat fitness function is used to test NCR and is given by

$$f = \sum_{l=0}^3 \sum_{i=1}^{2^{3-l}} m_i^l$$

where l is module level and m_i^l is module i of level l . m is equal to 1 when the module is present in the chromosome and 0 otherwise. In words, this means that the fitness value is equal to the number of matched modules. The maximal fitness is 15. Because of the hierarchial fitness modules, Royal Road functions are ideal for testing linkage learning representations. It is expected that faster convergence to the global optimum will be achieved with better linkage learning characteristics.

The developed NCR is applied to the 64 bit Royal Road function. Chromosomes from the seed population are randomly intialized with bits having equal probabilities of being 0 or 1. Non-coding segments are randomly generated such that the total index range is between 0 and 2. A rank based selection strategy is used with the best individual producing two offspring and the worst none. The offspring are always generated via the NCR crossover operator, with possible prior mutation of the non-coding genes. The resultant offspring are then subjected to mutation, which flips each coding gene with a probability of 2/64. Evolution is terminated either when the maximal fitness is found or when 1500 generations have passed.

Results and Discussion

Results are compared to two other EC implementations, a standard 2 point crossover EC and a fixed length NCR in which the lengths of each non-coding segment are *a priori* specified. For the latter case, each non-coding gene is set to 0.1 except for every eighth gene, which is set to 8. So, the probability that a crossover endpoint is chosen between level 0 modules is 10 times as great as being chosen within level 0 modules. This properly modularizes the chromosome and is expected to serve as a benchmark for the non-fixed NCR. Results are averaged over 30 runs and are presented for population sizes of 25, 50, 100, and 250. It was observed that small populations did not have effective linkage learning, so larger populations were introduced.

Table 5.1 shows the convergence results for the three different ECs on the Royal Road test function. There are three headings for each EC. μ indicates the average generation for convergence, σ is the standard deviation, and **Median** is the median time to convergence. For all population sizes and ECs, convergence results are not normally distributed. Each distribution is skewed towards faster convergence times as seen by comparison of median to mean values. The results for

| | Linkage Learning | | | Standard | | | Fixed | | |
|-----|------------------|----------|-------|----------|----------|-------|-------|----------|-------|
| pop | μ | σ | Med | μ | σ | Med | μ | σ | Med |
| 25 | 642.7 | 372.4 | 555 | 632.1 | 301.1 | 552.5 | 635.7 | 308.2 | 555 |
| 50 | 446.9 | 265.9 | 361 | 478.4 | 185.4 | 474 | 422.5 | 143.3 | 397.5 |
| 100 | 328.8 | 144 | 302.5 | 420.6 | 198.1 | 379 | 372.3 | 203.9 | 371 |
| 250 | 292.7 | 195.2 | 230 | 283.5 | 124.8 | 254 | 238.5 | 98.2 | 225.5 |

Table 5.1: Generation convergence results for Royal Road test function.

population sizes of 25 are essentially equivalent. Because of the limited pool of solutions, it is likely that proper linkages could not be evolved. In fact, it is somewhat surprising that the linkage learning approach is able to match the performance of the standard and fixed cases because it has overhead for determining proper linkage characteristics. Population sizes of 50 and 100 clearly show the improved performance of the fixed NCR over the standard representation. The linkage learning NCR does not show significant improvement over the standard case for a population size of 50. Although, the median values indicate that the NCR runs either converge much more quickly than the standard runs or much slower. Again, this is probably a result of small population sizes in which improper linkage characteristics can be learned. The results for population sizes of 100 are astounding as linkage learning NCR significantly outperforms both the fixed NCR and standard cases. Because the only difference between algorithms is linkage between genes, this implies that NCR can evolve the proper linkage characteristics. However, results for a large population size of 250 are mixed. The fixed representation not only outperforms the standard case as expected, but also the linkage learning NCR. Recall that three EC are tested, the developed linkage learning NCR, the standard canonical 2 point crossover, and the fixed NCR with the known modularity already built in. Moreover, NCR results are nearly equivalent to the standard case except that the median for NCR is considerably lower. In fact, the median for NCR is nearly equivalent to that of the fixed case. These observations imply that NCR is either highly successful, comparable to the fixed representation results, or extremely slow. It is suspected that the slow convergence rates are a result of excessive population sizes in which many initially good individuals can restrict the evolution of proper linkages. For example, if many individuals initially match some of the zero level, then there is no drive to evolve correct linkage characteristics because most crossovers will result in increased performance, regardless of gene index values.

Figure 5.4 shows the indices of the chromosome at the discovery of a module of the given level for a typical NCR evolution for a population of 100. Modules are demarcated by the change in symbol used to indicate each index value. Intuitively, optimal index arrangement would coincide with the module boundaries (*i.e.*, there would be large separations between indices at module boundaries). In particular, these large non-coding segments would be expected to coincide with the previous level's modularity because to improve fitness, these modules would have to be retained. Evidently, the results of Figure 5.4 do not comply with the expected index distribution. A possible and likely explanation is that the expected modularity is one of a set of optimal index distributions. It can be seen that many genes are very tightly linked (almost on top of one another) and that this provides sufficient modularity. The results of the average time to discovery of each level, as shown in Figure 5.5, confirm that the evolved linkages are sufficient. Notice that the linkage learning NCR significantly outperforms both the standard and fixed cases for time to discovery of each level. These results further confirm that NCR is able to determine effective linkages for fast evolution.

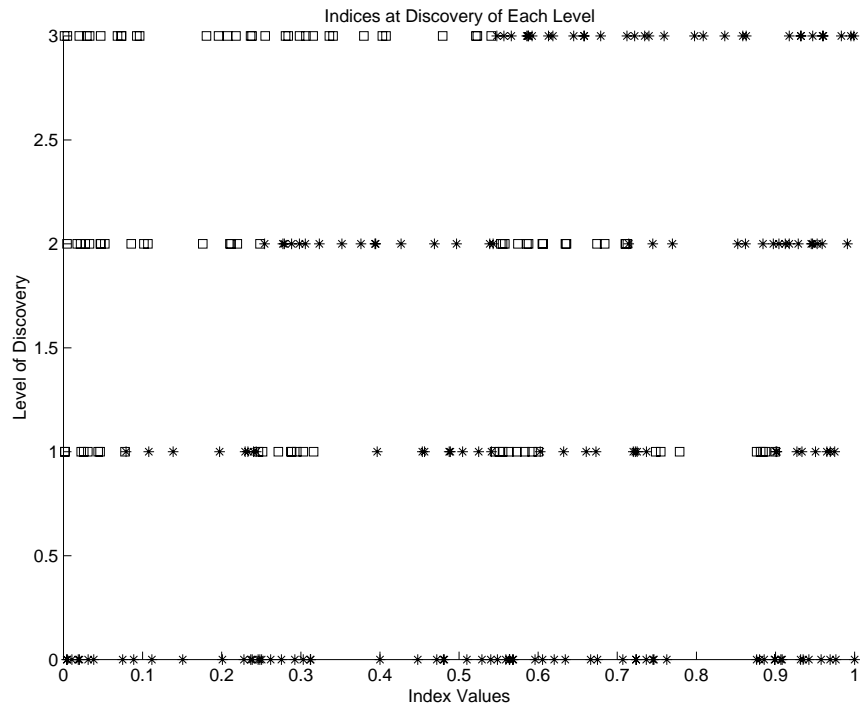


Figure 5.4: Gene indices for the first chromosome that matches a module of the given Royal Road module level.

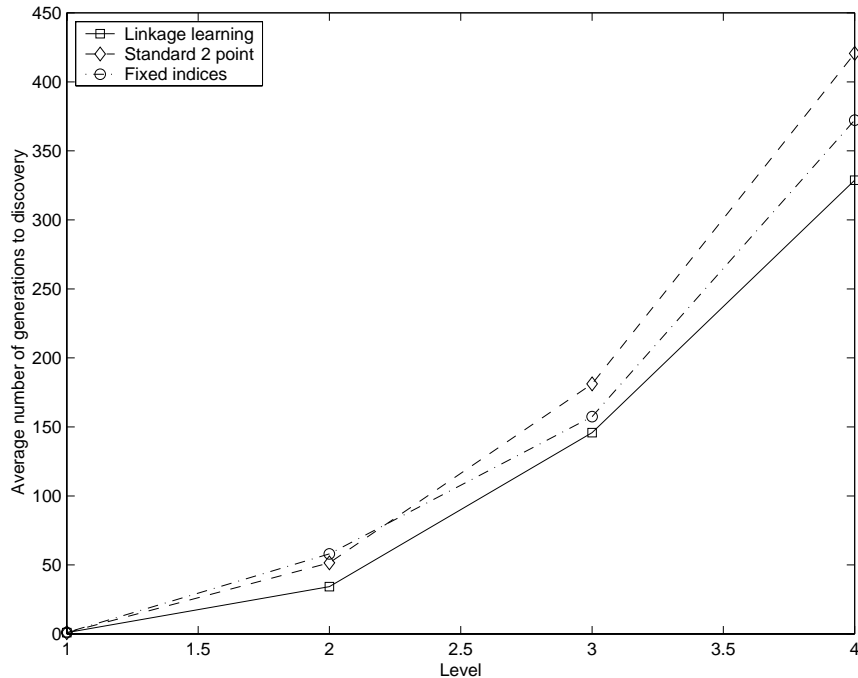


Figure 5.5: Average generation at which each level is discovered for population size of 100.

5.5 Design Problems

In contrast to the test problem, design problems have variable complexity search spaces. The chosen design problems are neural network and truss system design. Both neural network and truss system design are graph design problems. Each of these problems are discussed in detail below.

5.5.1 Design of Artificial Neural Networks

INTRODUCTION

Artificial neural networks (ANNs) are simplifications of biological neural networks used in a variety of applications such as pattern recognition. ANNs are basically function approximators that take a set of inputs and transform them into outputs. This is done by passing the inputs through a set of nodes, called *hidden* nodes, multiplied by some weighting factor. The hidden nodes have a nonlinear transfer function that make ANNs universal approximators. The outputs of the hidden nodes can be fed to more hidden nodes, the output, or previous hidden nodes and inputs. If there are no connections back to previous hidden nodes or to the inputs, the networks are known as feedforward. Feedback networks, which have connections back to previous nodes or inputs, are not examined here.

These types of ANNs are multilayered graphs with inputs at the bottom layer that feed into multiple hidden layers that are then connected to the output layer. Figure 5.6 shows an example of a feedforward neural network with two hidden layers. The output of each hidden node is calculated using

$$x_j = g\left(\sum_{i=1}^n w_{ij}x_i\right) \quad (5.3)$$

where x_j is the output of node j , n is the number of nodes, w_{ij} is the connection weight between nodes i and j , and g is a nonlinear function that is typically a sigmoid. In feedforward networks w_{ij} is equal to 0 if $i \leq j$. Thus, design of multilayer ANNs reduces to design of the optimal number of nodes and connections, connection weights, and node transfer functions. The developed range representation (RR) is applied to ANN design and is able to evolve all aspects of the ANN design except for the node transfer function (because sigmoids are sufficient for universal approximation). Much of this work was presented earlier [57]. The remainder of this section is organized to give a brief review of prior work, which is followed by a detailed description of the implemented EC. Results are presented and compared to canonical, non-linkage learning EC. The section concludes with a discussion and summary.

PREVIOUS WORK

The literature is replete with ANN learning (equivalently, design) algorithms. The default method is backpropagation and is so named because errors are propagated back from the output nodes to the input nodes. It is a gradient descent type method and is therefore highly susceptible to local optima. Another disadvantage is that it is unable to modify ANN architecture (*i.e.*, number of nodes and connections between nodes). These shortcomings have led to the use of alternative methods such as evolutionary computation and simulated annealing. Again, however, many of these cannot handle topology design. Of the learning algorithms capable of topology design, the results have been mixed. The interested reader is referred to a small subset on heuristic methods for neural network design as reviewed and found in [28, 25, 41, 50, 59, 82, 87, 92]. For more information on neural networks see [16, 33].

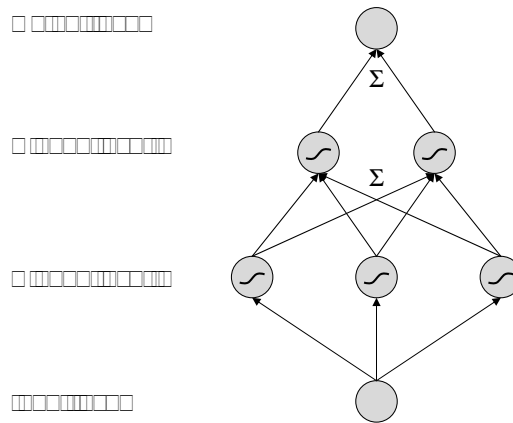


Figure 5.6: Example of a feedforward artificial neural network.

IMPLEMENTATION DETAILS

Encoding Scheme

It will be shown here that the developed range representation naturally lends itself to graph encodings. The graphs can then be transformed into feedforward neural networks through the addition of weight encodings.

Technically speaking, “a *graph* G is a finite nonempty set of objects called *vertices*. . . together with a (possibly empty) set of unordered pairs of distinct vertices of G called *edges*” [15]. In ANN terminology the vertices are the nodes and the edges are the connections. These graphs can be encoded in the following manner. Each gene encodes a single node and has six parameters. These parameters are a 2-D position vector and four parameters to encode the gene’s index in 2-D index space. Hence, a chromosome with a collection of genes encodes a graph in which connections are determined through gene interaction. If two genes have overlapping/intersecting index ranges, then they are connected, which is manifested in position space. This representation scheme is unable to encode all possible connection topologies for a given set of nodes. For example, with 1-D index ranges, any graph with four or more nodes connected in a cycle (as in Figure 5.7a) cannot be encoded using this representation. This is easily remedied using 2-D index space and the required index ranges are shown in Figure 5.7b. However, there remains a maximum lower bound on the number of nodes for which all connection topologies can be encoded. This maximum lower bound increases to 5, from 3 for 1-D index ranges, and increases as $2n + 1$ where n is the dimension of index space. The relation is proven informally for the n -dimensional case, but is illustrated using 2-D examples for clarity.

For an n -dimensional index space, there is always a configuration of $2n$ nodes for which the only representation is to have the index ranges aligned as faces of a hypercube (see Figure 5.7). Now, imagine that a node that is connected to the previous $2n$ nodes is added. Then, because the index range of the new node, j , must intersect all the previous nodes’ ranges, the new index ranges at the very least must fill the previously defined hypercube as shown in Figure 5.8 for the 2-D index spaces. Such a range representation precludes the possibility of adding another node that is not connected to j , but that is connected to the previous $2n$ nodes as shown in Figure 5.9. This is because no index range exists that intersects only the $2n$ previous ranges without intersecting the ranges of j as well. So, the maximum lower bound on the number of nodes for which all configurations can be encoded grows as $2n$ and an additional node, or $2n + 1$. Regardless of this shortcoming, 2-D index spaces

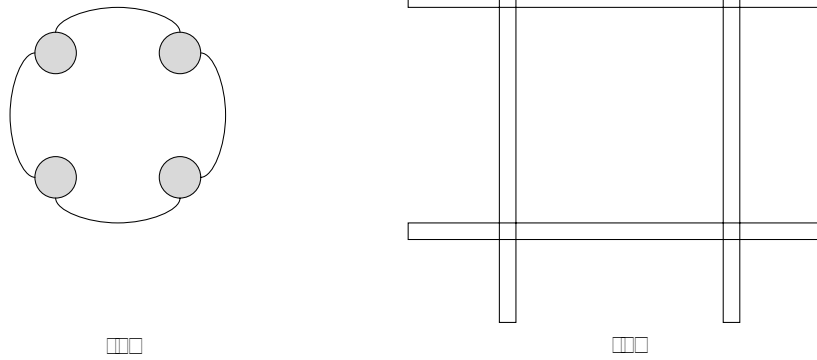


Figure 5.7: (a) Connection cycle. (b) 2-D index ranges needed to encode connection topology of (a).

are used because they provide a sufficiently large space of possible connection topologies and most viable neural networks do not have the aforementioned cyclical properties.

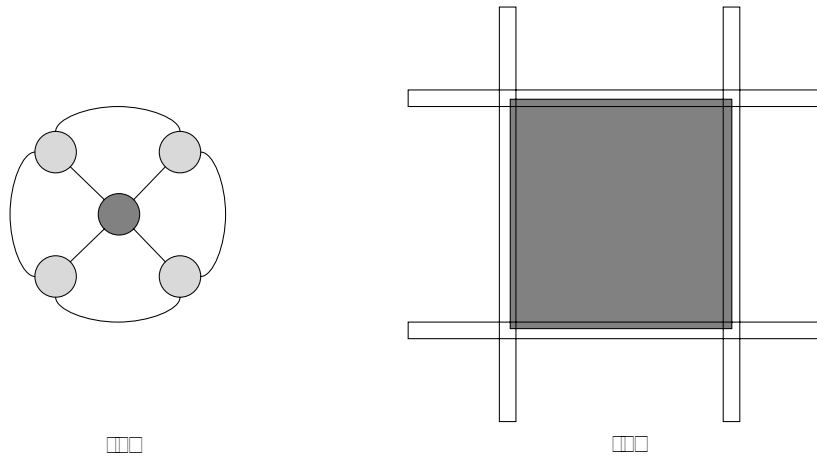


Figure 5.8: (a) Connection cycle with additional node. (b) 2-D index ranges needed to encode connection topology of (a). Required range of added node is shaded.

The connection topologies of the ANNs are determined from the intersecting index ranges. The actual topology is given by the positions of each node, which is again two dimensional for easy graphical interpretation. These positions are also used to determine the weights of the connections between nodes. Connection weights are calculated as the signed difference between the first dimension of the respective nodes' positions denoted by p_1^i . The second dimension is rounded down to an integer value and determines to which layer the node belongs. Because feedforward networks are of concern, only connections from nodes with lower to higher layer numbers are permitted. After the connections have been made, the input nodes are assigned according to ascending p_2 value, then p_1 value. Similarly, the output nodes are assigned according to descending p_2 value, then p_1 value. Every other node is a hidden node. If a hidden node does not have input connections, it is considered a bias or threshold node and given an input of 1. This completes the description of the encoding

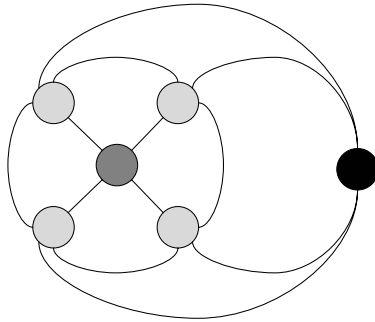


Figure 5.9: Connection cycle with 2 additional nodes that cannot be encoded by any combination of 2-D index ranges.

for feedforward ANNs. It should be noted that this representation can result in hanging nodes, or nodes that are not contained in any path from input node to output node. Furthermore, because any multiple output system can be decomposed into single output systems, only single output systems are tested for feasibility of the developed approach.

An example of an ANN coded in the manner described is shown in Figure 5.10. Figure 5.10(b) shows the index ranges for a five node configuration. The node number of each index range is marked in the upper left corner of the index range. Figure 5.10(a) shows the position of each node. Connections are added according to the range overlaps of Figure 5.10(b). For a single input and single output problem, node 1 is the input node because it has the smallest p_2 value and node 5 is the output node because it has the largest p_2 value. Node 2 has no inputs and is thus a bias node with a default input of 1. Connection weight can be inferred by looking at the p_1 distances between nodes.

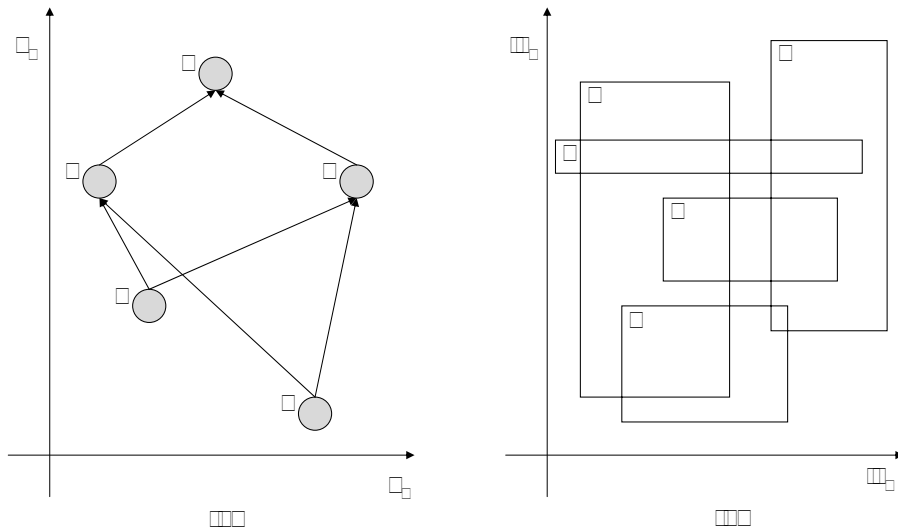


Figure 5.10: Parent 1. (a) Network topology. Connections are determined from overlapping index ranges shown in (b) and weights are proportional to signed p_1 difference. (b) Index ranges for each node.

Initialization

For each member of the initial population (population size is set to 200), the number of nodes is selected uniformly at random from 7 to 27 nodes. Because the test functions are single input/single output, this should ensure that there is at least one hidden node. Each node, or gene, has a 2-D position vector and a 2-D range matrix of four parameters. In addition, each parameter has a corresponding self-adaptive mutation rate. The initialization ranges for every parameter and mutation rate are shown in Table 5.2. id_i^- denotes the lower bound of the node's index range. id_i^+ is the extent, or length, of the node in index space. p_i is the node's position. The σ_j are the self-adaptive mutation rates of parameter j . So, there are a total of 12 parameters per node. This initialization procedure provides a diverse set of possible ANNs from which to start evolution.

| Parameter | Min | Max |
|---------------|-----|-----|
| id_i^- | 0 | 5 |
| id_i^+ | 0 | 1 |
| σ_{id} | 0.1 | 0.1 |
| p_i | 0 | 10 |
| σ_p | 0.5 | 0.5 |

Table 5.2: Nodal parameters and initial ranges for neural network encodings.

Variation Operators

Three mutation-like, point operators are implemented, namely, perturbation, deletion, and insertion. Perturbation is the standard evolution strategy mutation in which a Gaussian random variable with zero mean and standard deviation of σ_j perturbs parameter j . Deletion and insertion remove or add a random node, respectively.

The crossover operator is extended to 2-D index space by selecting 2-D crossover ranges. These ranges are chosen from an uniform distribution over the minimum and maximum indices of the chromosome's genes. Thus, crossover maintains similar connection topologies rather than similar weightings. The reason for this is because no amount of variation in weights will be sufficient for good approximation if the number of nodes is less than the complexity of the function.

The crossover operation is illustrated in a sequence of figures. Figure 5.10, seen previously, shows Parent 1 and Figure 5.11 shows Parent 2. The nodes of Parent 1 are marked by gray circles

while the nodes of Parent 2 are black. In addition, the nodes of Parent 2 are denoted by letters rather than numbers. These conventions are maintained in the remaining illustrative figures. Figure 5.12 overlays the positions and indices of both parents onto the same figures. The chosen crossover range is indicated by the bold rectangular region. Intersections with the crossover range indicate the crossover will swap nodes a and c with nodes 1, 3, and 5 between Parent 2 and Parent 1. The resultant offspring from this crossover operation are shown in Figures 5.13 and 5.14.

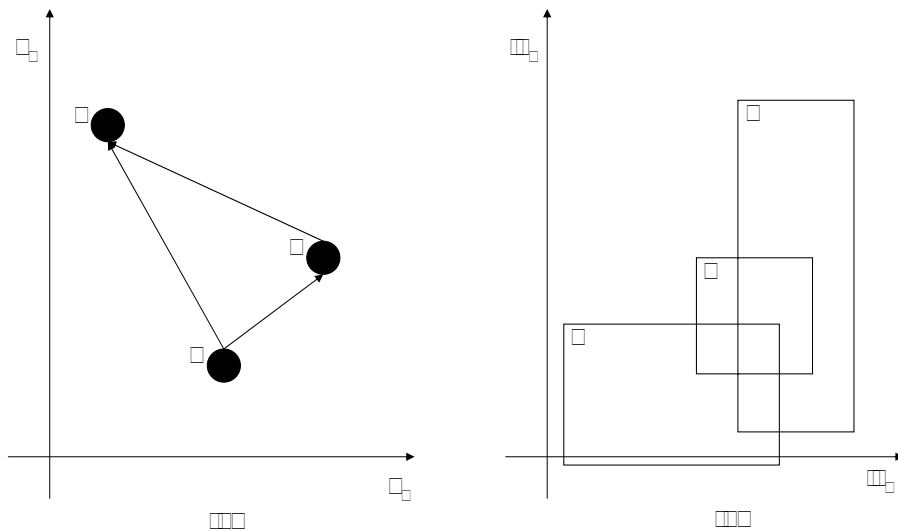


Figure 5.11: Parent 2. (a) Positions and network topology. (b) Index ranges for each node.

Selection Strategy

A steady state selection scheme is used in which a single individual is replaced every generation. All population members have equal probability of being chosen as a parent with the worst one being replaced by the offspring. For more information on steady state selection see [85].

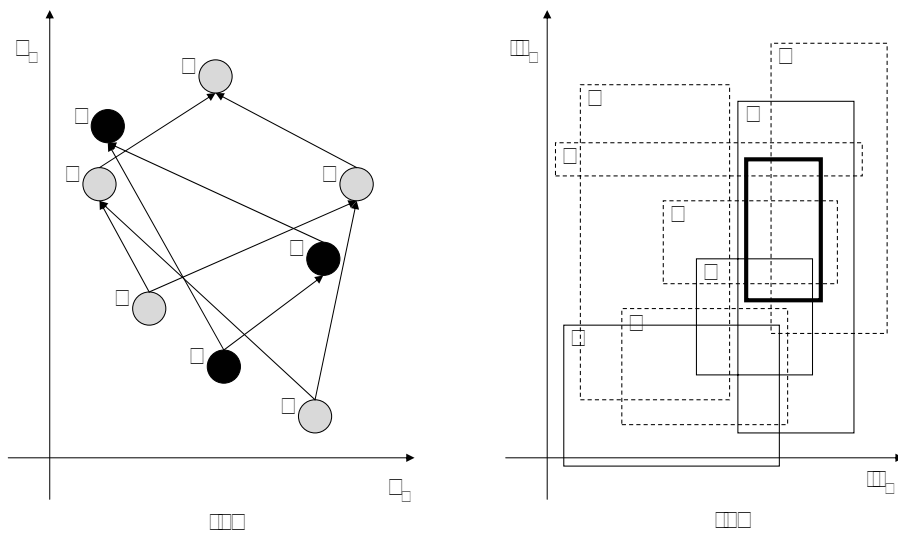


Figure 5.12: Overlay of Parents 1 and 2. (a) Positions and network topology. Parent 1 denoted by gray nodes and integers. Parent 2 denoted by black nodes and letters. (b) Index ranges for each node. Crossover range denoted by the bold rectangle.

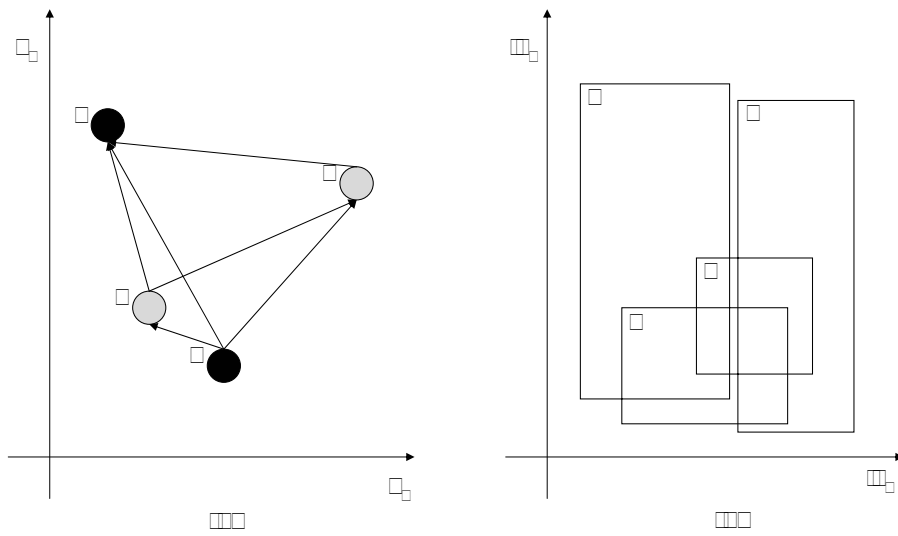


Figure 5.13: Offspring 1. (a) Positions and network topology. (b) Index ranges for each node.

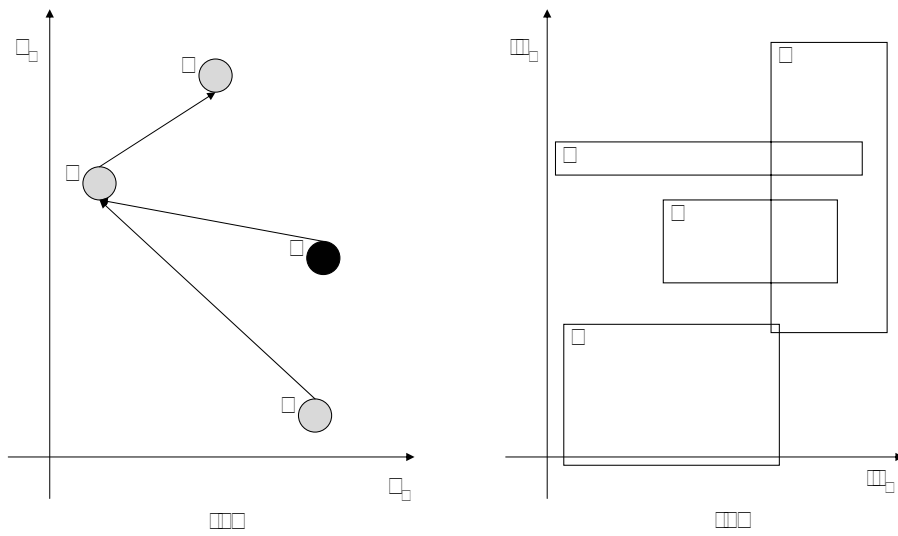


Figure 5.14: Offspring 2. (a) Positions and network topology. (b) Index ranges for each node.

Fitness Evaluation

Since ANNs are equivalent to function approximators, the mean square error (MSE) is used to evaluate the fitness of an individual. The MSE is expressed as

$$f = \frac{\sum_{i=1}^n (Y_i - y_i)^2}{n} \quad (5.4)$$

where n is the number of test datum, Y_i are the actual outputs, and y_i are the outputs predicted by the evolved ANN. As with the clustering design problem of Chapter 4, parsimonious solutions are always preferable. So, parsimony is enforced in two manners. The first requires that for offspring to survive, both their fitness and complexity (*i.e.*, number of connections) must be less than that of the parents. The second enforcement of parsimony is a penalization factor that either multiplies the fitness or adds to it, depending on the test function. The penalization factor was chosen empirically to be

$$\text{penalty} = \sqrt[4]{n} \quad (5.5)$$

where n is the number of connections.

Termination

The developed EC is allowed to run for 15,000 generations, which may seem large at first; however, recall that a steady state selection scheme is used in which a single individual is replaced every generation. This corresponds to roughly 150 generations for a standard proportional selection scheme for population sizes of 100, which are used here to test the evolution of neural networks.

RESULTS

Several single input/single output functions were generated to determine the ability of the evolutionary computation to synthesize good neural networks. The test functions are shown in Table 5.3. x is the input parameter and y is the output parameter. For each function, the training set consisted of 50 sample datum with inputs chosen uniformly at random from the range $[-5, +5]$. Population size was set to 100. Crossover is used to generate all the offspring, while mutation of any type is not used. This allows conclusions to be made from the results of whether linkage learning as accomplished by the crossover operators is effective in comparison to canonical EC methods.

Convergence results for each test function are shown in Table 5.4 and are averaged over 15 runs.

| No. | Function |
|-----|--------------------|
| 1 | $y = x$ |
| 2 | $y = \tanh(x + 2)$ |
| 3 | $y = \sin(x)$ |
| 4 | $y = \cos(x)$ |
| 5 | $y = x $ |

Table 5.3: Neural network test functions.

Range denotes results obtained using the range swapping crossover operator as developed earlier in the chapter. **Uniform** denotes results obtained using an uniform crossover operator that swaps each gene between parents with 50% probability. Comparison of these results show that the canonical (uniform) crossover operator easily outperforms the developed range crossover on every function except for the first. Even more troubling was that the range swap consistently evolved only linear mappings; clearly, this is only suitable for Function 1. Closer inspection of the results revealed that range swapping kept genes too immobile with respect to their closest neighbors. Basically, certain genes were initially too tightly linked, which prevented any future modification. These results led to the development of the **Uniform Range** crossover implementation. Again, results of the uniform range swap are shown in Table 5.4. Uniform range crossover, as its name suggests, combines uniform and range crossover. The first parent undergoes uniform crossover, with each of its genes having a 50% chance of being swapped to the second parent. If a gene is selected for crossover in the first parent, it is swapped with the gene in the second parent with the most similar range (as determined by the distance between the centroids of the index ranges). In addition, a minimal threshold (set empirically) must be met for the reciprocal swap from the second parent to occur. This uniform swap from parent 1 and reciprocal range swap from parent 2 ensures that networks with similar connection topologies are maintained after crossover. Thus, uniform range crossover evolves the likelihood that genes from different parents will be exchanged. Essentially, this can be considered interchromosomal gene linkage, while the previously evolved linkages were intrachromosomal. Results of the new operator are significantly better than both range and uniform crossover, except on Functions 1 and 4. It can be concluded that uniform range is able to effectively manipulate inter-chromosomal gene linkage characteristics. Results for each test function are presented and discussed subsequently.

| | Range | | Uniform | | Uniform Range | |
|----------|---------|----------|---------|----------|---------------|----------|
| Function | μ | σ | μ | σ | μ | σ |
| 1 | 1.0783 | 0.0811 | 1.0780 | 0.0824 | 1.0682 | 0.0608 |
| 2 | 2.5335 | 0.0008 | 2.3401 | 0.4481 | 1.9329 | 0.6915 |
| 3 | 6.0965 | 0.0223 | 4.7067 | 0.8589 | 4.4077 | 0.7467 |
| 4 | 5.9937 | 0.0001 | 5.8806 | 0.2904 | 5.8363 | 0.4184 |
| 5 | 11.7871 | 0.3454 | 11.8868 | 1.5419 | 8.466 | 1.3415 |

Table 5.4: Best evolved neural network fitness after 15,000 generations.

Function 1

Every crossover implementation was able to consistently evolve nearly optimal neural networks with a single connection from input to output with a weight of 1. Typical results of an evolution are shown in Figures 5.15–5.17. This was a relatively easy problem and no single crossover operator outperformed any of the others. It is evident that the penalization factor was able to effectively promote parsimonious solutions.

Function 2

As stated previously, range swapping only evolved linear mappings near the mean value of the training datum. Uniform crossover did not fare much better on Function 2 as it was only able to evolve non-linear mappings in 3 of the 15 runs. In contrast, uniform range swapping evolved non-linear mappings in 8 runs and was able to evolve significantly better approximations than the 3 successful runs of uniform crossover. Typical results of a successful, uniform range crossover evolution are shown in Figures 5.18–5.20. Note that the input node is the lowest node while the output node is the highest. Input and output nodes can also be differentiated from other nodes by their filled circle. Evidently, parsimony enforcement is not optimal as there are multiple hanging nodes that do not contribute to the fitness in the final evolved solution.

Function 3

Range crossover did extremely poorly on Function 3, yet again evolving linear approximations. Moreover, these were not even the optimal linear approximations as both uniform and uniform range

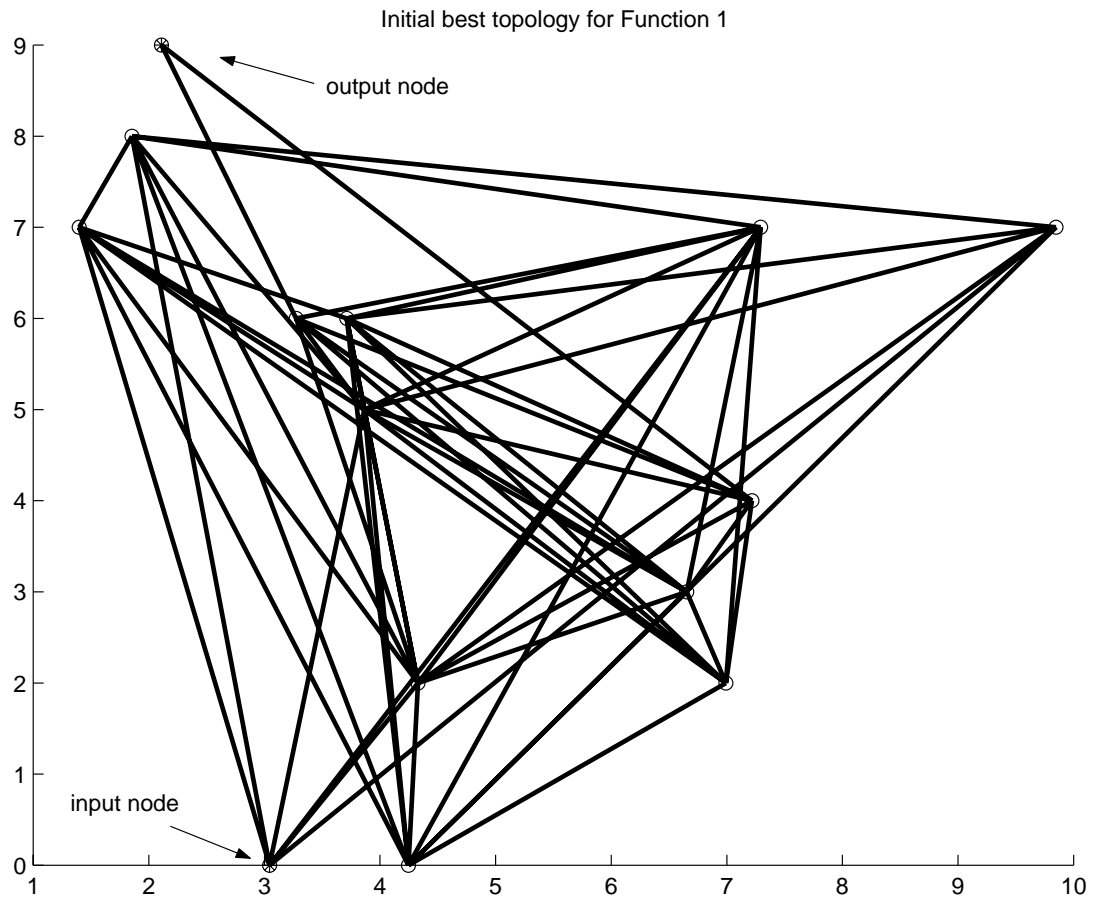


Figure 5.15: Initial best topology for Function 1 using uniform range crossover.

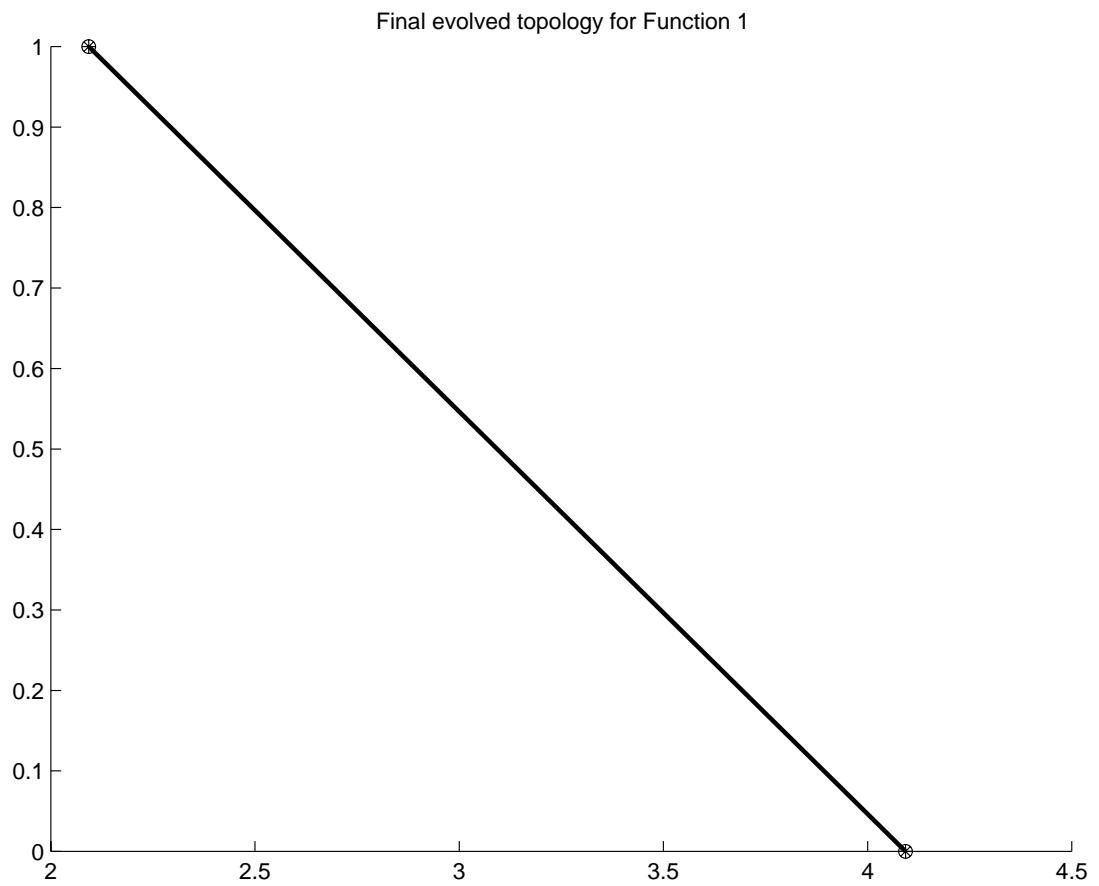


Figure 5.16: Final evolved topology for Function 1 using uniform range crossover.

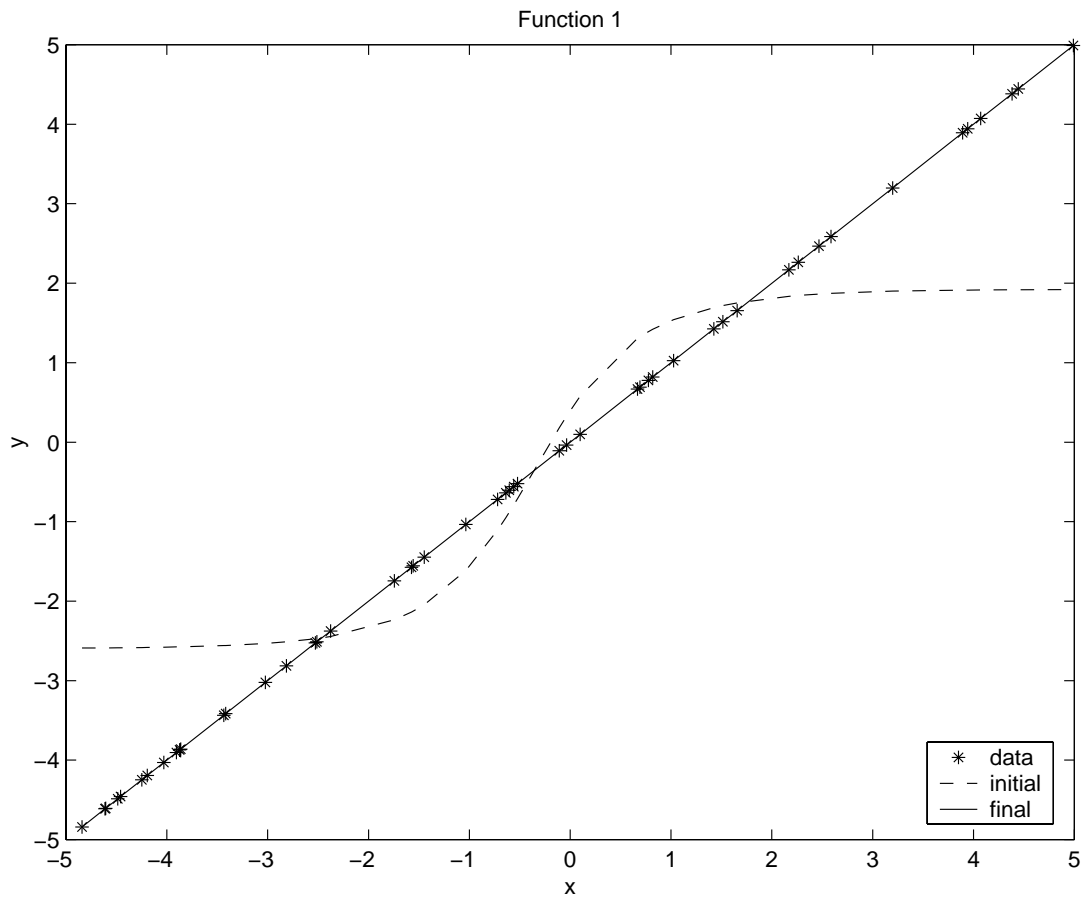


Figure 5.17: Function 1 data, initial best approximation, and final evolved approximation using uniform range crossover.

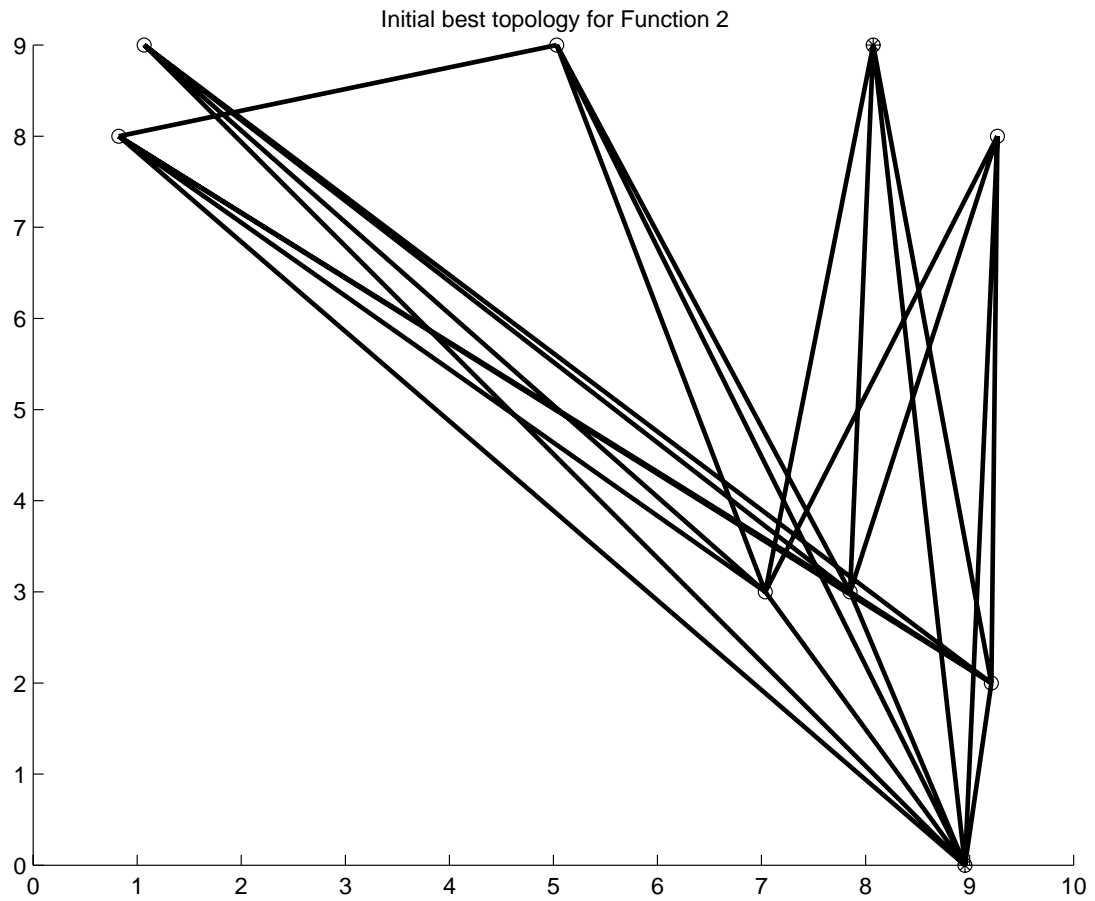


Figure 5.18: Initial best topology for Function 2 using uniform range crossover.

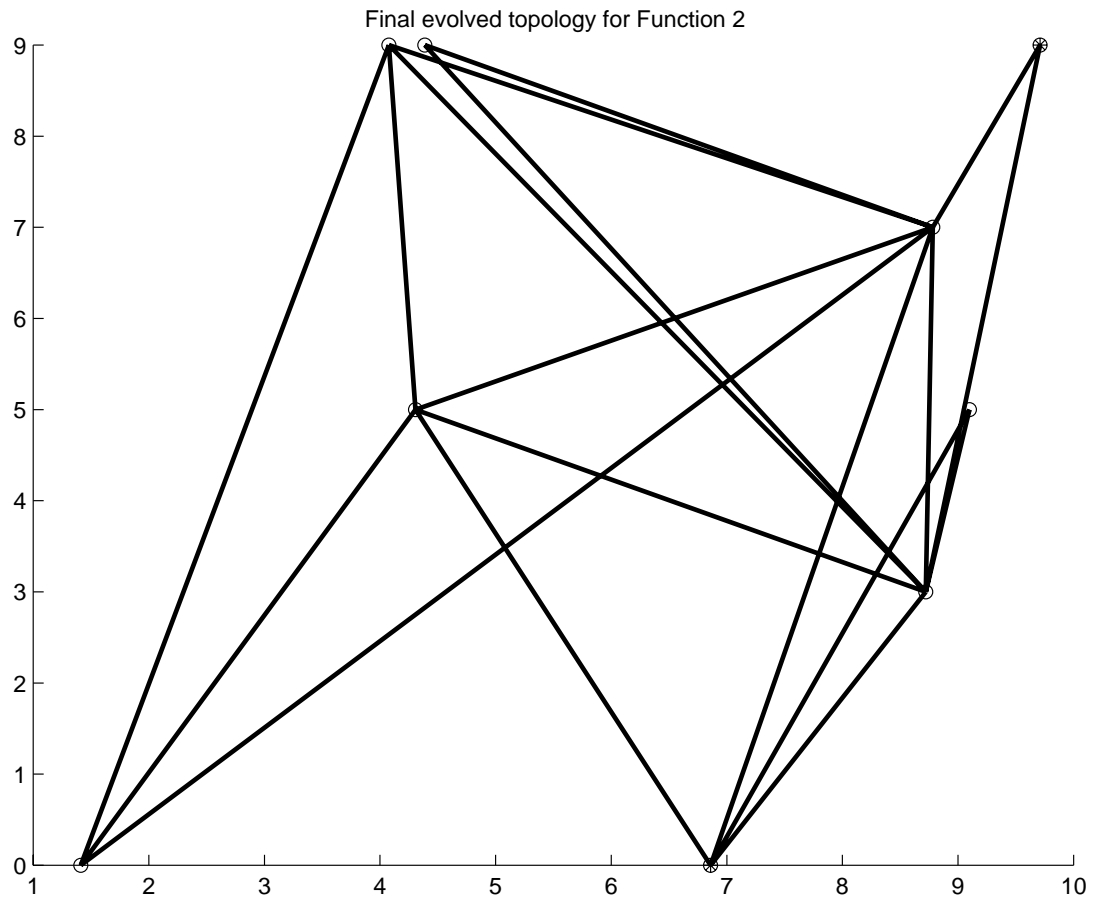


Figure 5.19: Final evolved topology for Function 2 using uniform range crossover.

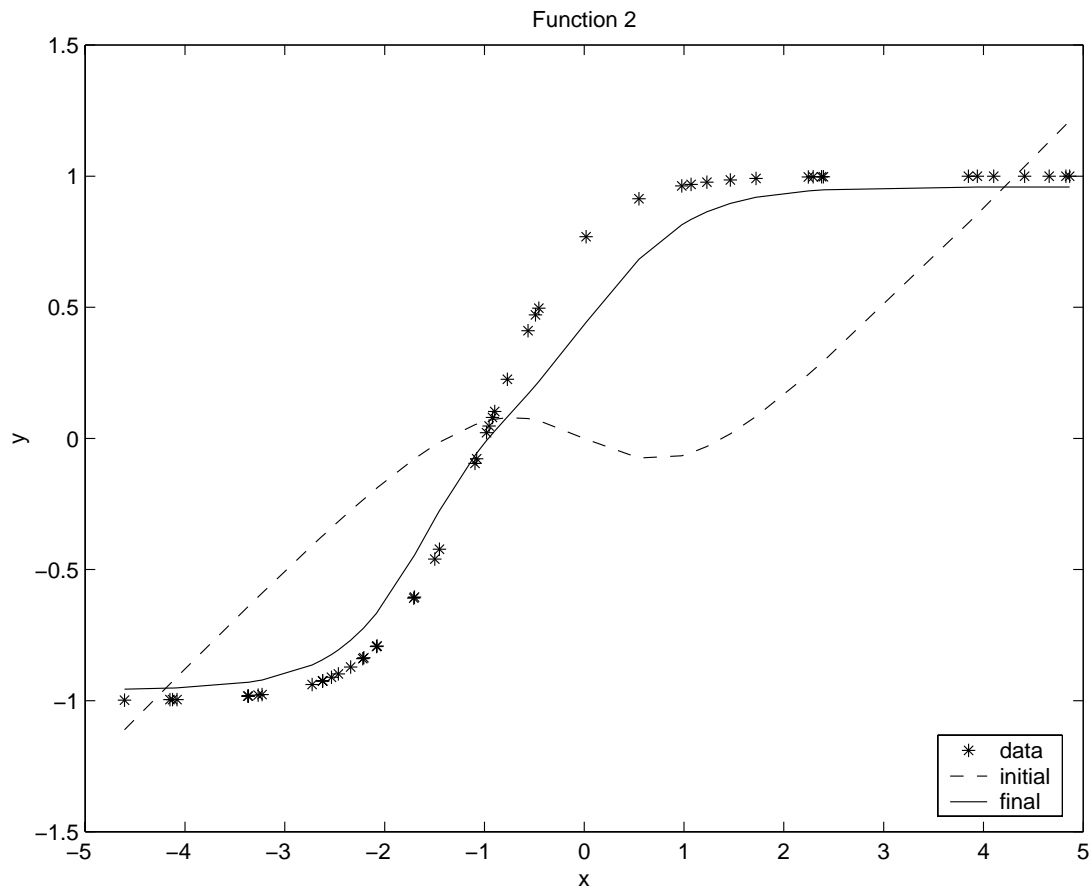


Figure 5.20: Function 2 data, initial best approximation, and final evolved approximation using uniform range crossover using uniform range crossover.

crossover were able to evolve more suitable linear approximations. As with Function 2, uniform range crossover nearly doubled the number of successful (*i.e.* non-linear) evolved approximations as compared to uniform. In this case the numbers were 9 to 4. However, uniform crossover was able to evolve one particularly good solution, explaining the similarity in mean fitness values. Typical results of a successful evolution by uniform range are shown in Figures 5.21–5.23.

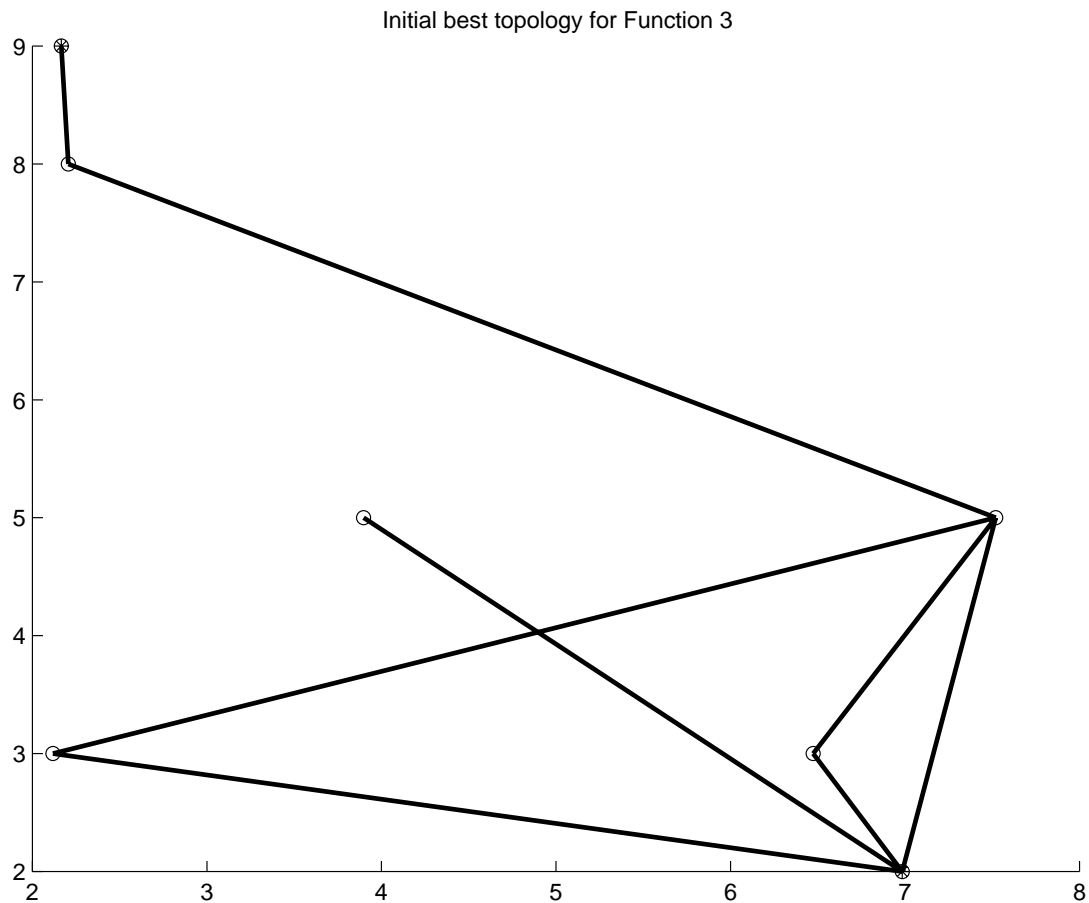


Figure 5.21: Initial best topology for Function 3 using uniform range crossover.

Function 4

Function 4 proved to be an exceedingly difficult function to evolve good approximations for, as nearly all evolutions resulted in linear mappings. In fact, both uniform and uniform range crossover only evolved non-linear approximations twice each with uniform range having better approximations in both cases. Results of one of the more successful runs are shown in Figures 5.24–5.26. It would seem that the linear mappings are a result of too prohibitive penalization factors, as the com-

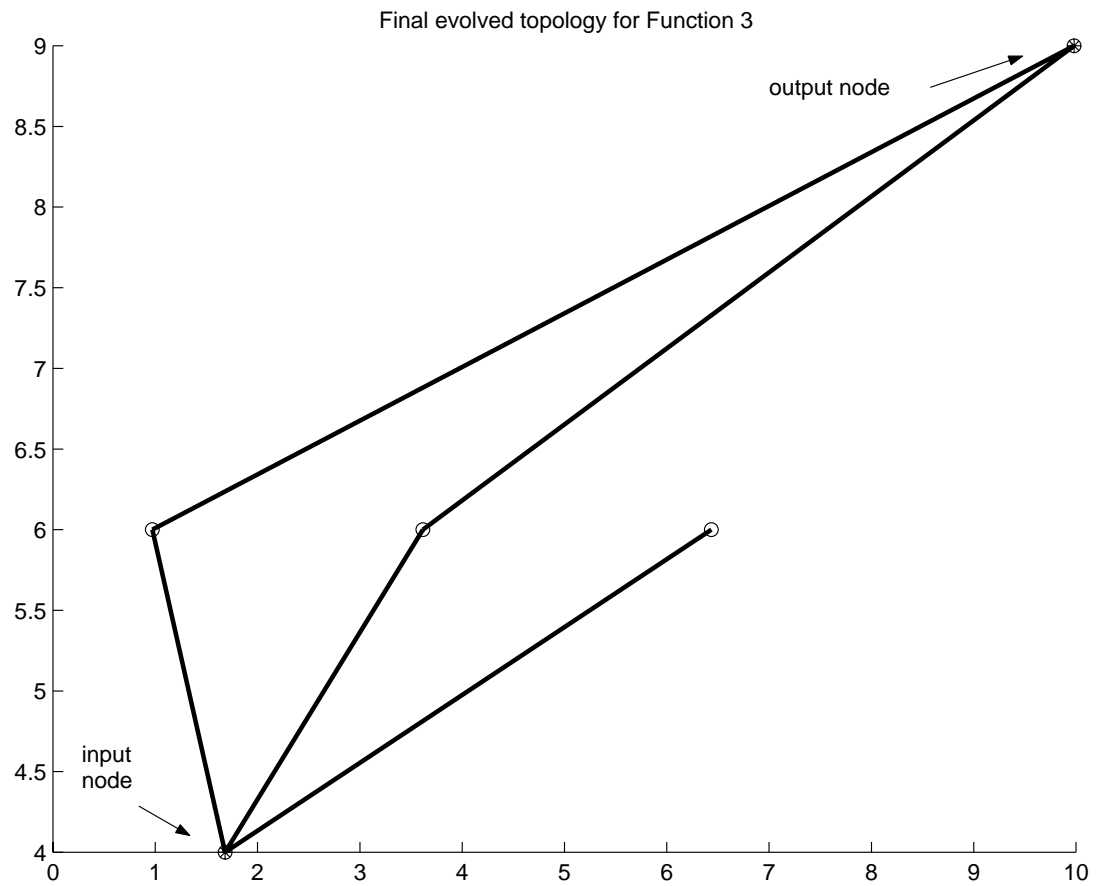


Figure 5.22: Final evolved topology for Function 3 using uniform range crossover.

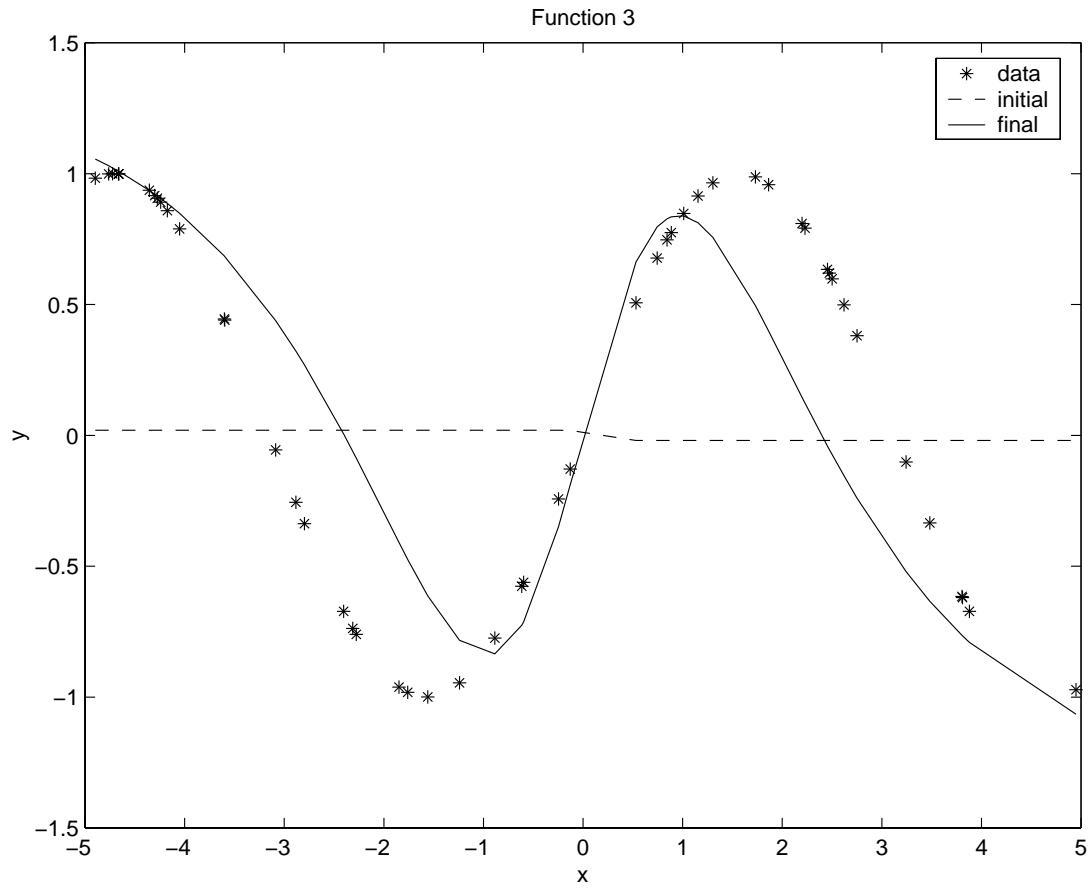


Figure 5.23: Function 3 data, initial best approximation, and final evolved approximation using uniform range crossover.

plex neural networks required for the function approximation could not be evolved without severe penalization.

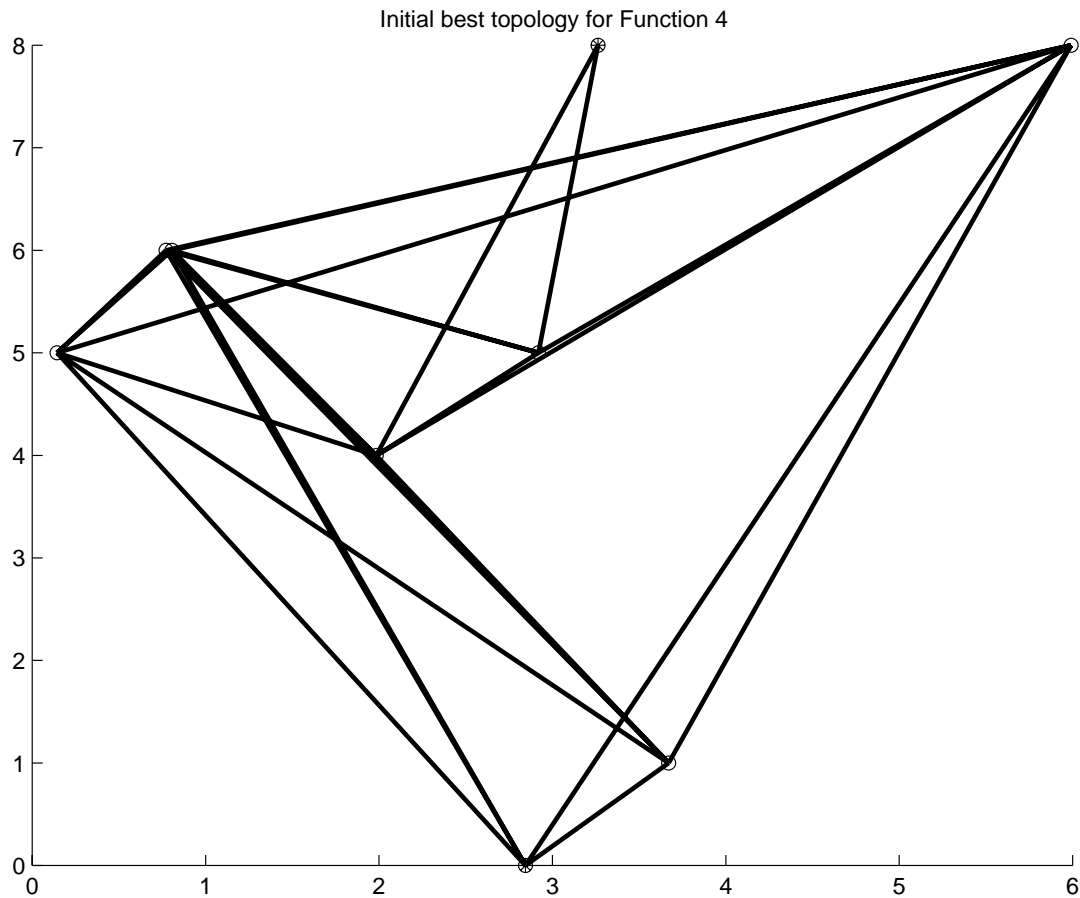


Figure 5.24: Initial best topology for Function 4 using uniform range crossover.

Function 5

Like Function 4, Function 5 initially proved a difficult problem with both range and uniform crossover only being able to evolve linear mappings at the mean training datum value. Somewhat surprisingly, uniform range crossover was able to evolve non-linear approximations in every single run. Typical results for an uniform range evolution are shown in Figures 5.27– 5.29.

CONCLUSION AND FUTURE WORK

A novel representation scheme is developed for encoding graph structures, which can then be easily transformed into a feedforward ANN. The topology of the network is defined by the interaction of

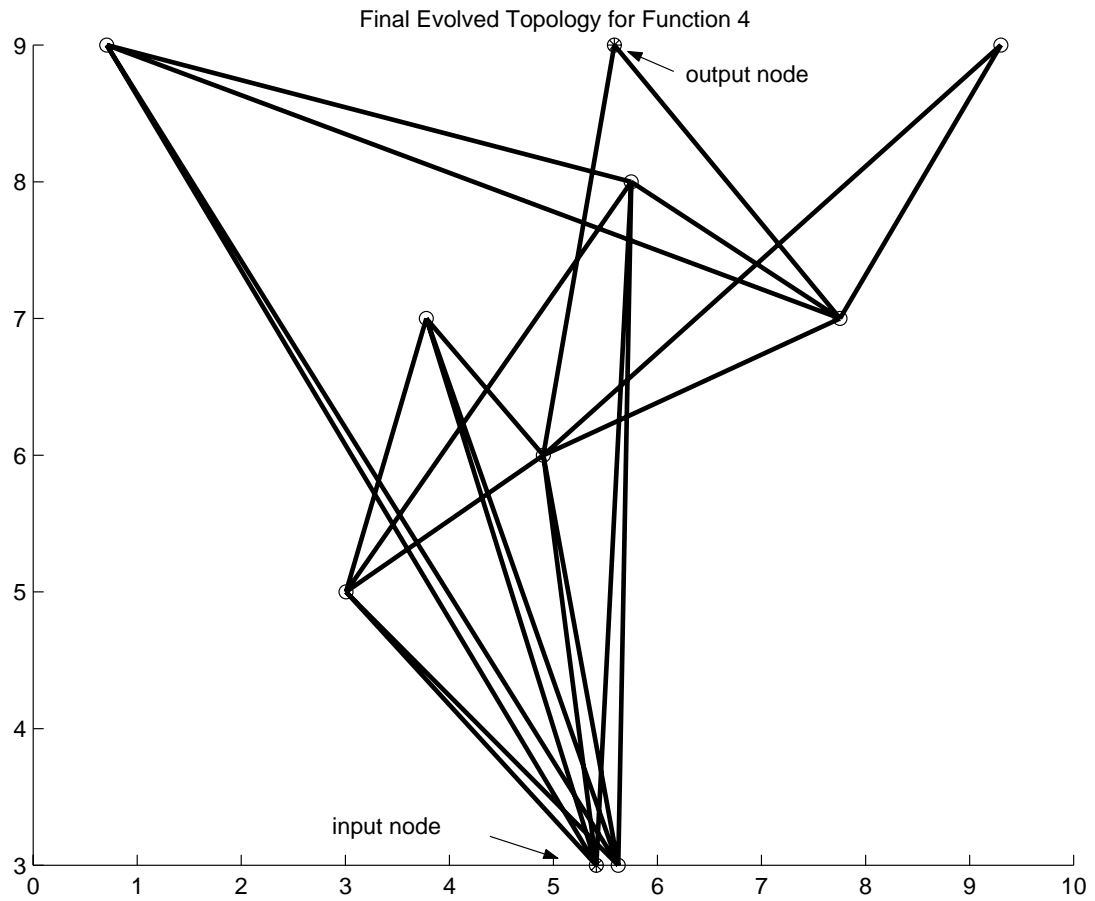


Figure 5.25: Final evolved topology for Function 4 using uniform range crossover.

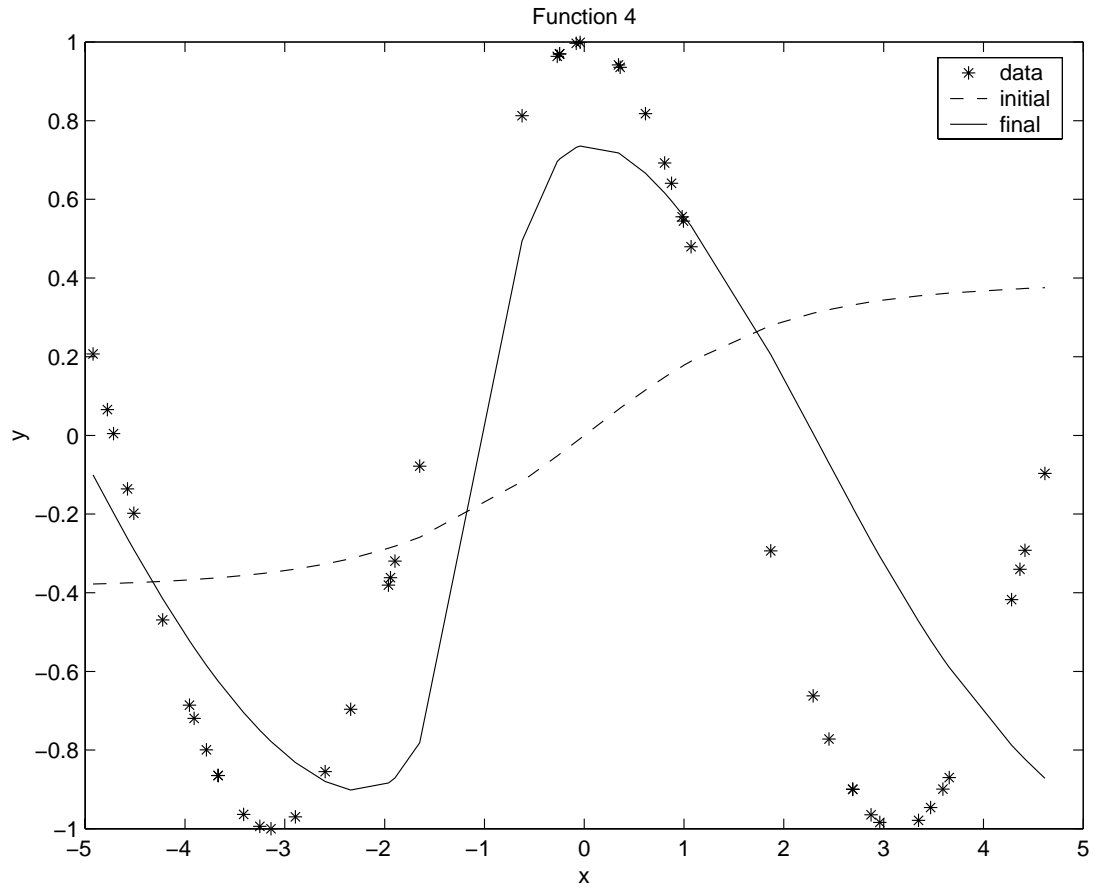


Figure 5.26: Function 4 data, initial best approximation, and final evolved approximation using uniform range crossover.

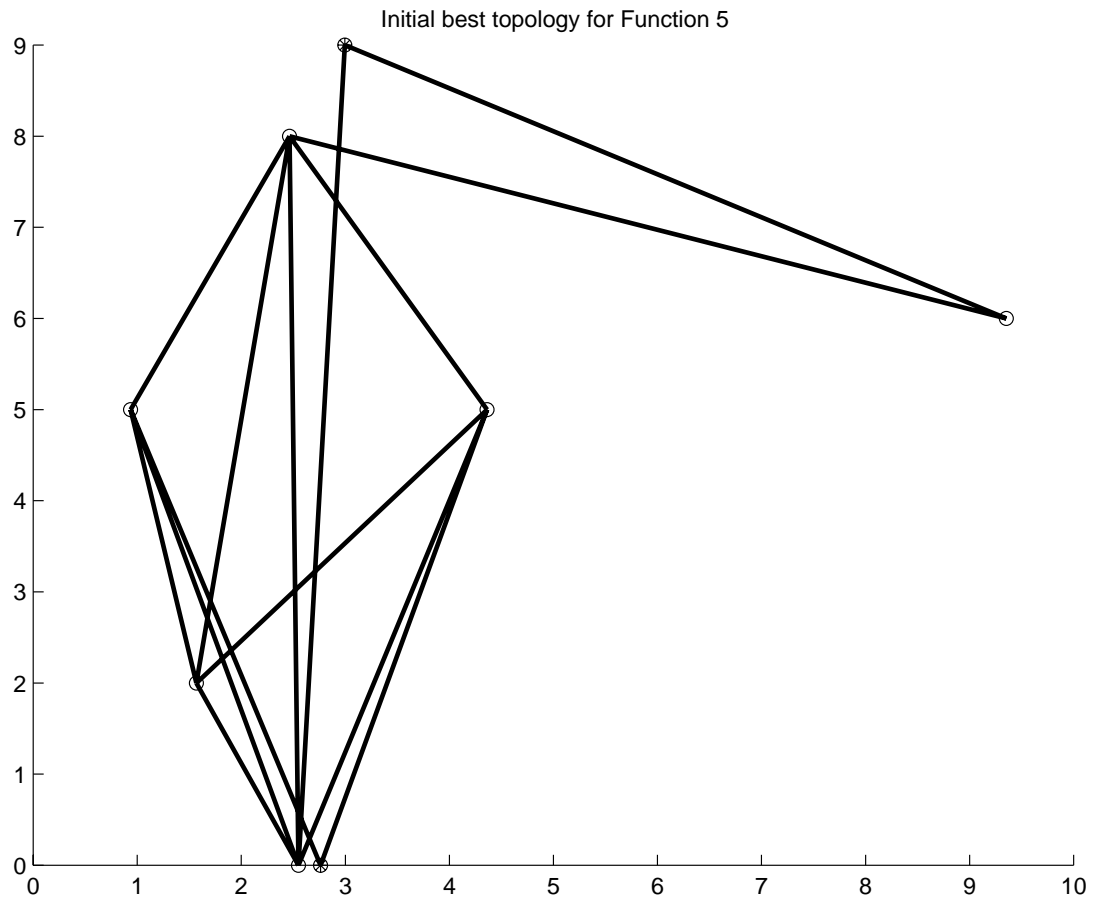


Figure 5.27: Initial best topology for Function 5.

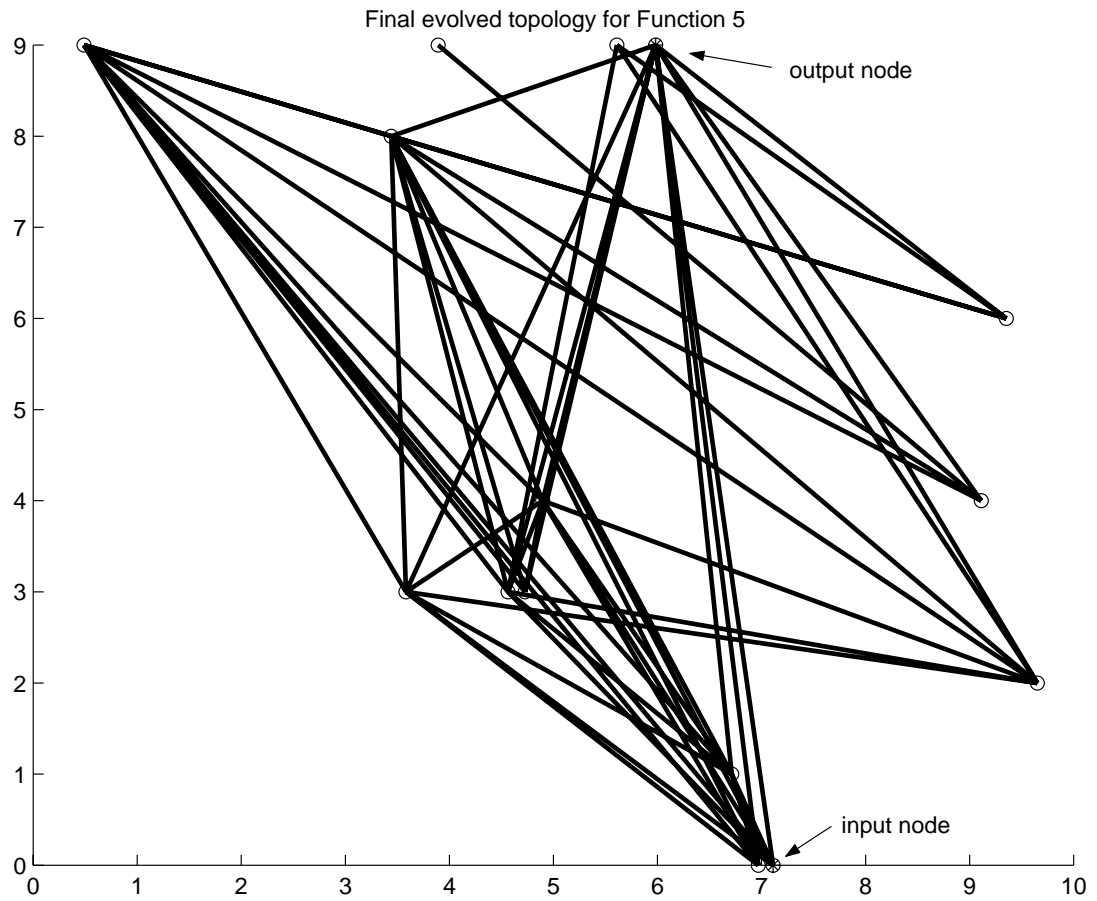


Figure 5.28: Final evolved topology for Function 5.

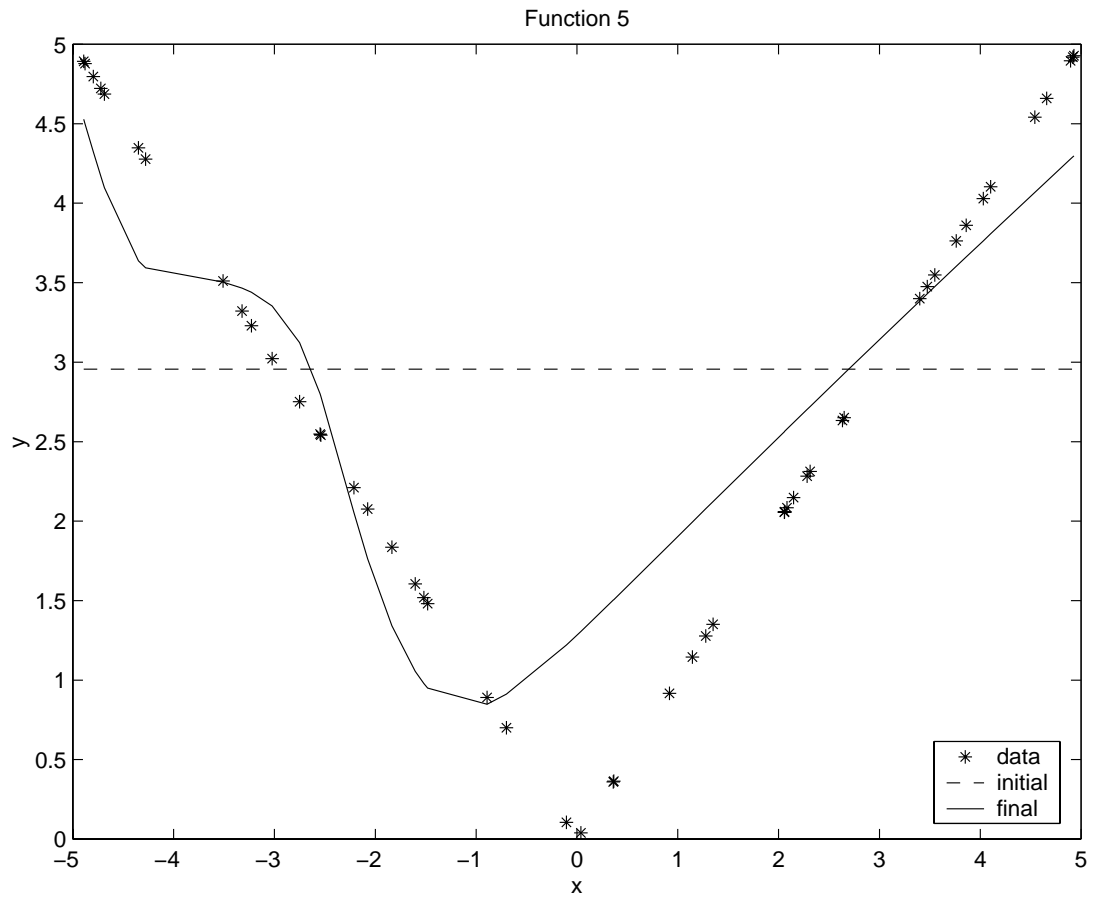


Figure 5.29: Function 5 data, initial best approximation, and final evolved approximation.

the nodes in index space, while the weights are determined through interaction in position space. This separation allows specification of a large space of feedforward neural networks. The number of hidden layers and connections are dynamically allocated. This is in contrast to matrix approaches where the number of nodes is fixed ahead of time and element (i, j) stores the weight between nodes i and j . Such an approach is memory inefficient when the optimal number of nodes is less than the matrix size and inflexible when the number is greater than the matrix size.

Results of the developed EC and corresponding crossover operator are somewhat poor, leading to the development of a crossover operator that maintains linkages between genes of different parents rather than of the same parent. This so-called uniform range crossover operator is able to effectively manipulate these inter-chromosomal gene linkages to significantly outperform uniform crossover. Results are promising and parsimonious solutions are often achieved through the use of penalization factors. However, overuse of penalization seems to have led to premature convergence of many evolutions to linear approximations, which have no penalization because they typically only have a single connection. So, further study on parsimony maintaining mechanisms are required.

Another future topic of research is, obviously, application to real world problems. At the same time, the effectiveness on multiple input and output mappings are of interest as well. From these studies on more complex systems, the scaling properties of the algorithm can be determined empirically; thereby allowing predictions of the expected behavior of the algorithm on new problems.

5.5.2 Design of 2-D Truss Structures

INTRODUCTION

Trusses are assemblages of links used for many mechanical structures such as cantilevered beams in booms and cranes, bridges, roofs, and electric cable pylons, which are better known as “iron maidens” in North America. Although truss design is fairly mature, truss design for novel applications is still primarily a trial and error process. An automated truss design methodology is introduced in this section. The approach closely follows that of neural network design in Section 5.5.1. The reason is that truss systems can also be represented as weighted graphs. In truss terminology, vertices are joints and edges are links.

PREVIOUS WORK

Truss design via EC has a long history, which is not surprising because discrete structures are typically easy to adapt to string like genetic representations. Two primary approaches were taken, cellular and matrix encodings. Cellular refers to the discretization of a domain into individual cells that either are or are not filled with material. These pixel-like representations proved useful, but the mapping between pixels to truss systems was difficult. In addition, the domain size and grid resolution must be known prior to the start of evolution. Matrix encodings fix the number of joints and the links are stored in a binary connection matrix. While this representation avoids the pitfalls of cellular encodings, if the number of joints is overspecified, then there is inefficient memory usage and if underspecified, then the optimal solution cannot be found. Much of this work can be found in [13, 14, 49].

IMPLEMENTATION DETAILS

Encoding Scheme

As was the case for ANNs, each gene encodes a single joint and the location of each joint is a 2-D position (2-D truss structures are evolved for easy graphical interpretation). Links between joints are then formed according to the intersection of gene index ranges, which are again 2-D. The index and position space dimensionalities are equal because this provides a good tradeoff between the number of search alternatives and the “curse of dimensionality”. In the former, recall that there is a bound on how many joints can be encoded before the range representation can no longer represent all possible connection topologies. So, higher dimensionality index spaces have more connection topologies from which to search. But, increasing index space dimensionality leads to exponential increases in hypervolume. This is known as the curse of dimensionality. Hence, upon moving to higher dimensional index spaces, sparser connection topologies are generated initially because of lower probabilities of overlapping ranges. The choice of 2-D positions and 2-D indices leads to a good compromise between number of solution alternatives and non-sparse connection topologies.

The structure of the truss is further modified depending on the proposed design problem. This modification is discussed later in relation to fitness evaluation.

Initialization

A population size of 50 is chosen. Initialization of population members is similar to the ANN case. Each member is generated with between 3 and 23 joints. Every joint's position is chosen uniformly at random from $[(0,0),(5,5)]$. The index ranges are then selected by choosing a location in the range $[(0,0),(5,5)]$ and choosing a width in each direction from a Gaussian with zero mean and 0.1 variance. No additional self-adaptive mutation parameters are used; so, each gene encodes 6 parameters.

Variation Operators

Four variation operators are used for truss evolution. These are mutation, insertion, deletion, and crossover. Mutation perturbs either the index ranges or positions, but not both simultaneously. Insertion and deletion respectively create or remove a joint at random. The crossover operator is extended to 2-D index space as done with ANNs.

The probabilities of each operator occurring are 0.5 for crossover, 0.35 for mutation (0.4 of the time of which is range perturbation and 0.6 is position), and 0.075 for both insertion and deletion.

Selection Strategy

A steady state selection is used in which a single individual is replaced every generation. All population members have equal probability of being chosen as a parent with the worst one being replaced by the offspring.

Fitness Evaluation

There are two primary goals for truss design, analogous to those of ANN design in which the goals are good approximation and low complexity. For trusses, the competing objectives are high stiffness (*i.e.*, good load bearing characteristics with small deflection) and low weight (*i.e.*, low complexity or fewer links).

The fitness function is the maximum deflection (under a specified loading), which is penalized according to the total length of all link members (each of equal cross section and materials). Written analytically, this is given as

$$f = \max_{\forall j} \delta_j \cdot \sqrt{\sum_{i=1}^n l_i} \quad (5.6)$$

where δ_j is the deflection of joint j , n is the number of links, and l_i is the length of link i . The penalization factor was chosen empirically. Because non-valid truss structures are also possible using the proposed representation, they are given infinite deflections as penalization.

Joint deflections are calculated using FRAME, a truss analysis software program developed by H.P. Gavin at Duke university [35]. The joint deflections are dependent on both truss configuration and applied loads. The configuration of the truss is often constrained by the design problem. For example, 2-D cantilevered truss design typically requires that there are at least three joints, two of which are vertically aligned pin joints (*i.e.*, the supports) and the third is the loaded joint. Such constraints need to be met by the genetic representation, and thus further modification of the encoded trusses is necessary. These modifications are accomplished in a somewhat *ad hoc* manner where joints are moved to the constrained positions and joint ranges are normalized to specified lengths. For example, in the cantilevered truss problem, the two leftmost joints (assume the cantilever is supported on its left) are aligned vertically to meet the support conditions. Then, because a total cantilever length is often specified, the range of gene positions is scaled to the proper length. Finally, the loadings in question can be applied to the appropriate truss.

Termination

Evolution is terminated after 500 generations or, equivalently, 500 new evaluations and offspring. This is a particularly small number and is necessary because truss analysis takes a significant amount of time for every evaluation.

RESULTS

A simple 2-D truss design problem is addressed using the recently developed uniform range crossover operator. The problem is the design of a cantilevered truss system. Rather than redoing the comparisons done for neural networks, the design problem presented here is simply for confirmation of the effectiveness of the developed approach to solving other graph design problems.

Cantilevered Truss

A cantilevered truss is a truss system that is fixed on one end and must support a load at the opposite end with minimal deflection. The length of the truss system is fixed at 10 and deflections are calculated with an applied load of -100 on the rightmost joint (no units are used, although any standardized set would be acceptable). In addition, the height between the supports is set to 2. Recall

that the fitness is penalized for weight (*i.e.*, because of equal material properties this is equivalent to the sum of link lengths). This prevents the number of link members from becoming unbounded, since infinite stiffness can be obtained with infinite members.

The population size is set to 50 and evolution is allowed to run for 500 generations. This turned out to be a relatively easy problem as the initial best solution was always fairly good. Typical results are shown in Figures 5.30 and 5.31. The load vector is shown on the rightmost joint in both figures. These results corroborate the results on neural networks and show that the developed representation is quite effective for graph design.

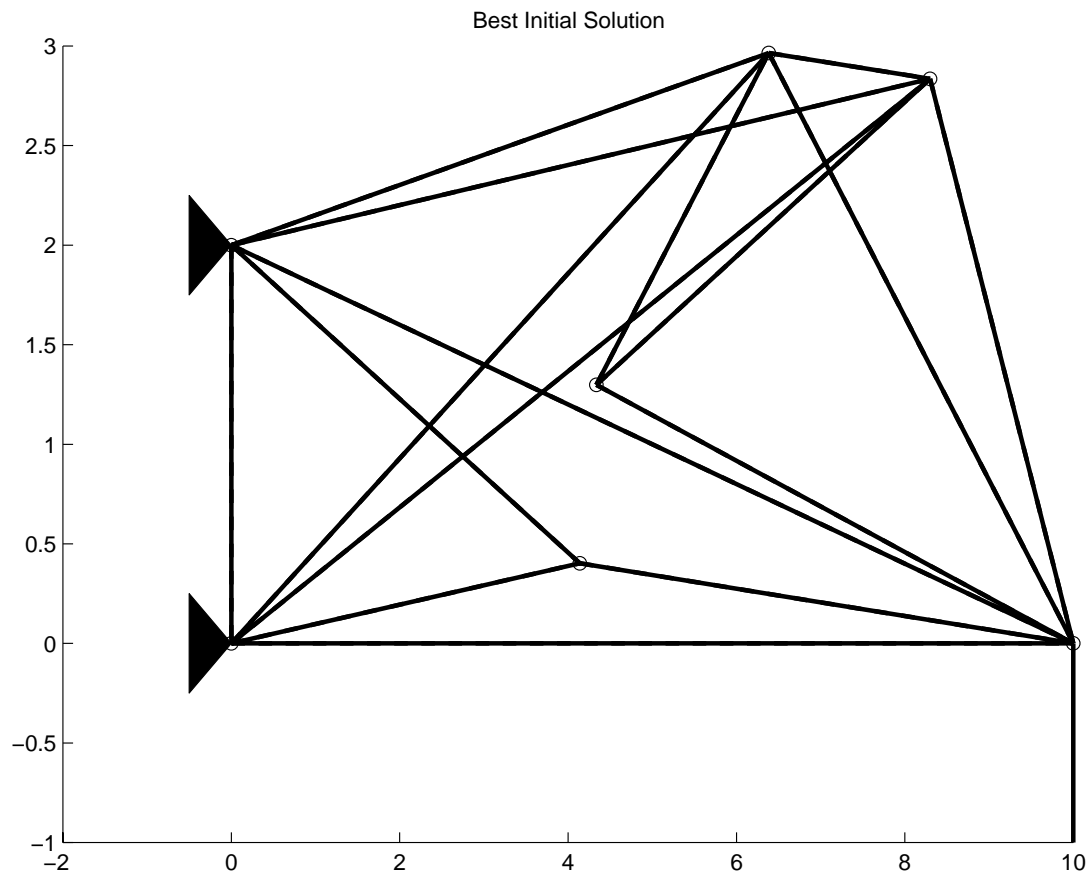


Figure 5.30: Best truss solution in initial population.

CONCLUSION AND FUTURE WORK

Graphs are highly useful structures that have wide applicability; thus, their design becomes an important issue. In this section and in the previous, a novel method has been developed for evolving similar types of graphs. Both resulted in good designs and confirm that the approach is a useful tool

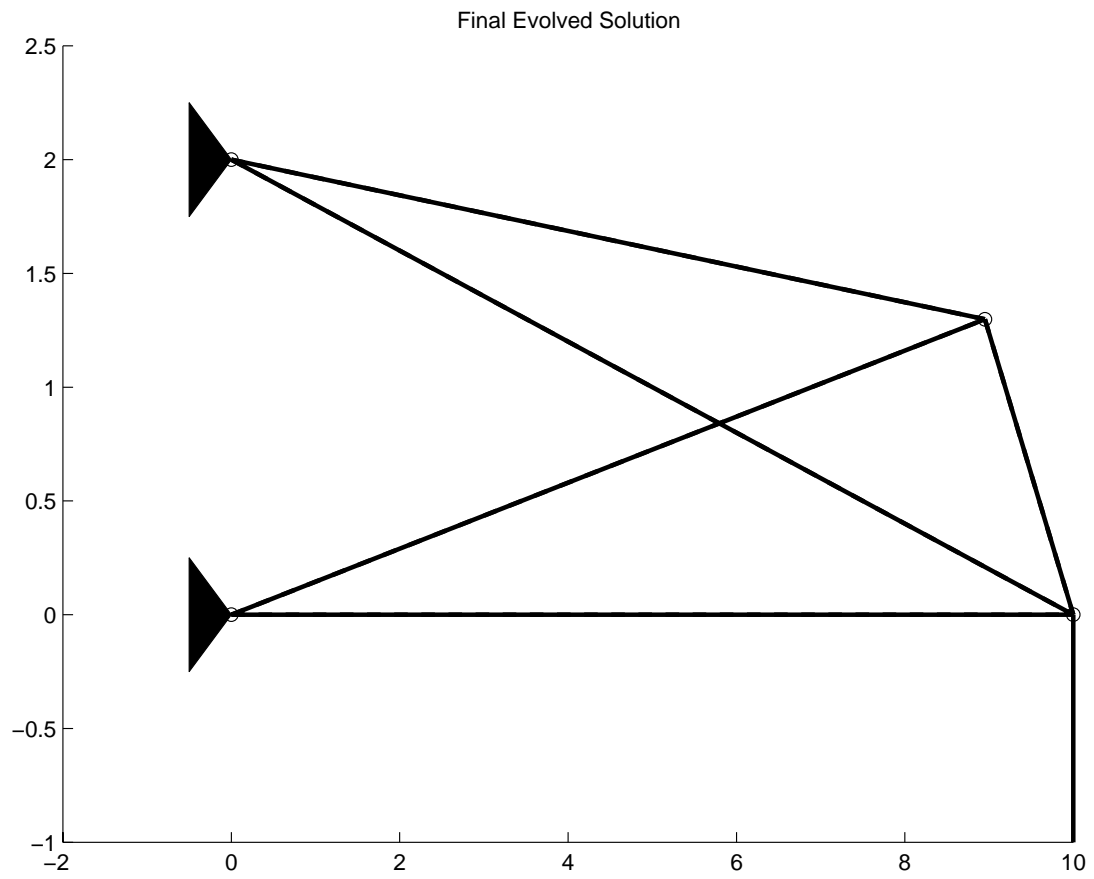


Figure 5.31: Final evolved truss solution using uniform range crossover.

for graph design.

Although results are promising, more work needs to be completed on truss design. In particular, extensions to 3-D are of interest as are extensions to deployable structures (*i.e.*, trusses that can unfold from compact spaces). A more rigorous method for implementing design constraints is also needed. The current *ad hoc* method of readjusting joint positions is not expected to be sufficient as the complexity of the design problem increases.

5.6 Summary and Discussion

Gene separations were introduced in Chapter 4 to enable variable complexity search. It was quickly observed that random initialization of gene separations resulted in different gene linkages within chromosomes. A biological analogy was then made, with the separations mimicking the non-coding DNA segments in real genetic codes. As these non-coding segments evolve their lengths, different linkage characteristics are evolved, leading to more evolvable populations. In addition, linkage learning permits a designer to determine crucial interacting portions of the final solution – as tightly linked genes probably evolved because their disruption would have been disadvantageous.

Subsequently, the non-coding representation was extended to a more flexible range representation scheme. Rather than storing non-coding lengths, coding ranges are encoded with each gene. Because ranges of different genes are allowed to overlap, a much larger space of gene linkages becomes available for search. Furthermore, nothing is lost in the transformation because non-coding segments and their lengths are implicitly encoded through gaps that exist between gene ranges. The idea of overlapping genes also finds a precedent in nature as some genes have overlapping reading frames.

The non-coding representation is tested on the Royal Road function, which has well known linkage characteristics. Comparisons with canonical EC and other results show the effectiveness of the approach for linkage learning. Two design problems are then solved using the linkage learning range representation (RR)⁵. These design problems are the design of artificial neural networks and truss systems, both of which are graph design problems. Results are less promising and indicate that RR perhaps may prevent good linkages from being learned in these problems. A new approach, still involving index ranges, is developed in which crossover maintains linkages between genes of

⁵Perhaps linkage evolving is more appropriate because learning implies something done within the lifetime of an individual. Unfortunately, linkage learning is the adopted term in the literature and is thus used here.

different parents (*i.e.*, certain genes have higher probabilities of being swapped with one another). These results show that the range representations can be useful in both interchromosomal and intrachromosomal gene linkage learning. Unfortunately, the class of problems for which each is applicable is not known and is a topic of future research.

Despite its success, RR has some shortcomings. Although good linkage characteristics and modules are evolved, no mechanism is in place for module reuse. This is a significant drawback when trying to evolve systems that may exhibit module reuse (*e.g.*, automobiles with four of the same tire). Recent developments have been able to evolve module reuse as found in [46, 47, 64, 81]. Each of these are generative encodings based on Lindenmayer or L-systems [63]. L-systems are grammars, and, like most well-defined grammars, can be used to generate infinitely many structures by applying grammatical rules on the alphabets in different combinations. L-systems add push/pop functionality to grammars that can be used to effectively evolve modules and their reuse. The idea is that a module, say a table leg, can be pushed onto a generative stack at any time. This means that table legs can be added anywhere. By popping modules off a stack, the current position can be retained. Thus, rather than growing one table leg on top of another, by popping off a table leg, growth of the table top can continue and a table leg can be pushed on at the appropriate time and place. This description of L-systems was decidedly brief and the reader is referred to [46, 47, 63, 64, 81] for more details. The upshot of this discussion is that module reuse can be accomplished, and that L-systems are promising candidates for enabling such reuse.

Another problem with RR is that as the dimension of index space increases, the probability of choosing crossover ranges with no intersecting genes increases. If this is the case, then crossover does not occur. The problem is compounded when the encompassed ranges of two chromosomes do not overlap. Currently, there is no way to address these issues. However, an innovative approach is developed to take advantage of non-overlapping chromosomes that results in speciation. The following chapter presents this approach and provides an unified EC implementation for achieving variable complexity search, linkage learning, and speciation.

Chapter 6

Speciation

6.1 Introduction

Most design problems have multimodal search spaces with sets of nearly equally good or acceptable solutions. These types of problems have performance landscapes with multiple peaks each of which corresponds to an acceptable design. Traditional evolutionary computation (EC) techniques are unable to find sets of solutions primarily because of stochastic effects (also known as genetic drift) that lead to loss of population diversity and convergence to a single solution. In addition, lack of population diversity is often the reason premature convergence is encountered. Thus, by promoting diversity, it is likely that both premature convergence and convergence to single solutions can be avoided; *albeit*, often at the expense of longer evolution.

Once again, nature provides an elegant solution for maintaining population diversity, namely, speciation or the formation of species. Although there are numerous definitions of species, species is defined here to be a group of individuals whose members may only interact with other members of its group. In other words, individuals of different species may not interact. Because of this interspecies non-interaction, pockets of distinctly different individuals can be evolved, resulting in forced maintenance of population diversity. Hence, elucidation of speciation mechanisms is vitally important for designing appropriate diversity promoting mechanisms in EC.

Speciation is dichotomized into allopatric and sympatric speciation. Allopatric speciation refers to speciation as a result of geographical isolation. In this case, one founding population is separated into two subpopulations that are geographically and reproductively isolated. Genetic drift and differing environmental factors will lead to divergent evolution of the two subpopulations into two different species. In contrast, sympatric speciation is the formation of species from a single found-

ing species in the *same* geographical location. That sympatric speciation even occurs is widely disputed, yet there is empirical and theoretical evidence that it can and does occur [12, 23, 77]. A well known hypothesis proposes that sympatric speciation occurs through the combination of disruptive selection and assortative mating. Disruptive selection biases survival to individuals with extreme traits, while assortative mating biases mate preference to similar individuals. For example, imagine that in a certain environment there are two kinds of seeds, those that are in tight crevices and those that are encased in large coverings. Then, birds with medium sized beaks are at a disadvantage because their beaks are too large for the small crevices and too small to break the large coverings. So, disruptive selection occurs for birds with extreme beak sizes. Furthermore, if the birds exhibit assortative mating, then two species of birds will arise as the small and large beak birds evolve independently.

Initially, the mechanisms of allopatric and sympatric speciation may seem quite different. Upon closer inspection however, it is seen that the rudimentary mechanisms are the same. Reproductive isolation of a single founding population into two or more subpopulations must occur for independent evolution and speciation. This observation serves as the inspiration of a novel method for speciation based on evolving reproductive isolation amongst individuals. The developed speciation approach is introduced in this chapter after a brief review of other EC speciation approaches. Results of the developed speciation are presented on a variety of multimodal test functions. The design problem on neural networks of Chapter 5 is then revisited and the chapter concludes with a summary and discussion of future work.

6.2 Previous Work

Numerous approaches exist for maintaining population diversity. Each approach requires some type of speciation in which reproductive isolation of groups is enforced. These speciation mechanisms can be differentiated into two distinct approaches – topological isolation and separation. Topological isolation is isolation of individuals through location. Each member of a population has a location and can only interact with individuals within some neighborhood. So, speciation in topological isolation is either pre-defined or emerges as individuals self-organize their locations. Conversely, separation has an active controller that determines which individuals can and cannot mate with one another. Typically, these controllers must have global knowledge of all individuals in order to determine which individuals are most similar and should thus be part of their own species.

6.2.1 Speciation by Topological Isolation

A simple solution to topological isolation is to create two or more non-interacting populations and allow them to evolve separately. This is the approach taken by most parallel computing versions of EC in which different populations are evolved on each computing node, examples of which can be found in [17, 73]. Often, migration of individuals is allowed, but the rate of migration and choice of migrating individuals are almost always specified prior to evolution. It is obvious that different search landscapes will have different requirements for good migratory behavior. Nevertheless, these approaches have been quite successful. These EC models are often called island or coarse-grained models. A serial version of island models was developed in [84] where an additional bit was used to encode each individual's "island".

Another popular approach is that taken by cellular genetic algorithms (CGAs), which are also known as diffusion or fine-grained genetic algorithms. In CGAs, each individual inhabits a single cell of a grid. Matings are restricted to individuals in neighboring cells. Thus, individuals separated by many cells cannot interact and a gradation of species, rather than abrupt species differentiations as in island models, occurs. One benefit that is accrued through this approach is that migratory behaviors do not need to be specified *a priori* and emerge as a result of evolution. However, the idea of fixed topologies is wholly artificial and open to debate. The interested reader is referred to [51, 68] for more information on CGAs. One publication on CGAs that closely follows the research in this chapter is found in [58].

6.2.2 Speciation by Separation

The prime example of speciation by separation is the niching approach taken by Goldberg and others [7, 21, 38]. Basically, a controller determines which individuals belong to which species through similarity thresholding. Only individuals within a specified "niching" radius of each other can compete and mate with one another. There are two inherent difficulties to this approach. First, radius size needs to be specified *a priori* and may not adequately portray the actual widths of the performance peaks. Second, the number of species must be specified in advance. This is equivalent to specifying the number of performance peaks without any prior knowledge, which is a risky endeavor. These niching approaches have often gone by the name of fitness sharing.

6.3 Development of Speciation

Any sexual population can be represented by a graph structure. Graph nodes are equated to population members and graph edges indicate possible mating partners. Weights on the edges can then be used to portray the probability of mating between the connected members. Thus, species are represented by groups of nodes that have no edges between any of their members. So, a straightforward speciation mechanism would be to add and remove edges that would modify the interaction characteristics of the population.

Populations in canonical EC have fully connected topologies in which all individuals are possible mating partners. If proportional selection is used, edge weights can be determined from each individual's relative fitness, which is a difficult and tedious task. Given a fixed population size of n , $(n - 1)^2/2$ parameters are required to encode the entire population's graph structure¹. So, as population size increases, the number of parameters grows exponentially. This exponential growth only occurs because of the direct graph encoding in which global knowledge of all possible interactions is required. Hence, an indirect graph encoding in which matings can be determined in a localized manner is desirable, particularly when population sizes grow. In fact, nature utilizes localized behaviors, rather than global knowledge, for determination of mating characteristics.

An indirect graph encoding is developed here that, when coupled with the developed operators, enable rapid and simple speciation. Every population member is embedded in a topological space (2-D in most instances) and allowed to inhabit a small area of said space. The inhabited range is encoded as a rectangle, or hypercube in higher dimensions, and is equivalent to the index ranges introduced in Chapter 5. In fact, if the range representation developed in Chapter 5 is being used, then the inhabited range of each individual is simply the minimum and maximum bounds of the index ranges of all genes. Possible mates are then determined through range overlap and a graph of the population interaction structure can be generated. Edge weights, or mating probabilities, are proportional to the amount of overlap. The idea is that individuals only mate in accordance to how much time they inhabit the same area. Specifically, the probability of two individuals mating is equated to the intersection of their inhabited areas, over their union, which is given as

$$p_{ij} = C \frac{\cap A_i A_j}{\cup A_i A_j} \quad (6.1)$$

¹This number doubles for directed graphs where the probability of mate selection depends on which parent is chosen first. For example, the probability of choosing mate j for individual i is not generally equal to the probability of choosing i for j .

where p_{ij} is the probability of individuals i and j mating, C is a constant to be defined later, and A_i is the area of individual i . C is used to reduce the effects of dimensionality. Because overlaps will decrease in higher dimensional spaces, C can remediate this effect by growing at the same rate. Empirical results show that $C = 1$ is adequate for 2-D and 1-D index spaces.

The indirect graph encoding described above can be used to encode populations with any number of species, but mechanisms for modification of graph topology and edge weights are still required for species formation. The developed operators are dependent on localized individual behavior and a steady state reproduction scheme (recall that this means one individual is replaced every generation). Two parents are selected at random and are recombined according to the probability given in Equation 6.1. If recombination does not occur, then each parent is mutated and replaced by its offspring if the offspring is more fit. If recombination does occur, then reinforcement learning is used to modify the ranges of each parent. Basically, the idea is to reward good mating combinations and punish poor mating combinations. It is expected that offspring will retain mating characteristics similar to their parents. The hypothesis is that reinforcement learning will increase the probability of good mating combinations while at the same time decreasing the probability of poor combinations.

Good matings are defined as those in which both offspring are fitter than both parents. Likewise, poor matings are those in which both offspring are less fit than both parents. All other cases are neutral matings and both parent and offspring ranges are left unmodified. In the case of good matings, positive reinforcement is needed to increase the probability of crossover in the next meeting. This is accomplished by moving the ranges of both offspring closer together according to the relation

$$\text{id}_i^{1'} = \frac{w \text{id}_i^1 + \text{id}_i^2}{1 + w} \quad (6.2)$$

where id_i^j is the range bound for parent j in dimension i , w is a noisy weighting factor, and id' indicates the new range value. So, the new id value is a weighted mean of the parents' ranges. Such a modification results in larger overlaps between offspring and the offspring will be more likely to mate if and when they are selected for steady state replacement. However, the ranges cannot be moved together too quickly, so a noisy weighting factor multiplies the previous range value to prevent premature range convergence. w is probabilistic and varies uniformly from 1 to 2.

In the case of poor matings, negative reinforcement is used to decrease the probability of subsequent crossovers. From Equation 6.1 it is evident that crossover probability can be decreased either by decreasing the intersection of ranges or by increasing the union of the ranges. The former is

chosen and is implemented as a repulsion between the two ranges. This repulsion of index ranges is accomplished in the following manner

$$\text{id}_i^{1'} = \begin{cases} \text{id}_i^1 + .1|G(0,1)| & \text{if } \text{id}_i^2 < \text{id}_i^1 \\ \text{id}_i^1 - .1|G(0,1)| & \text{otherwise} \end{cases} \quad (6.3)$$

where $G(0,1)$ is a Gaussian random variable with zero mean and unit variance. An analogous equation is used to move id_i^2 . These modifications of index ranges nicely accomplish the task of repelling individuals with poor mating characteristics. Therefore, the offspring are less likely to mate when presented the chance.

6.3.1 Discussion

Although this section has defined certain rewards and punishments as given by Equations 6.2 and 6.3, any reasonable attraction and repulsion operators could have been implemented. The developed operators were chosen because they are able to manipulate the relation in Equation 6.1 in a manner that reinforces or punishes the given mating combination. Empirical success of these operators has shown the effectiveness of these speciation mechanisms on multimodal search and design problems to be presented later.

The adopted implicit graph representation of the population topology suffers from the same drawbacks as the graph representations in Chapter 5. Restating these limitations, the complete set of all connection topologies cannot be encoded by the range representation when the number of individuals exceeds $2n + 1$ where n is the dimension of index space. An informal proof can be found in Section 5.5.1. However, those connection topologies that cannot be represented are highly atypical of population topologies such that the drawbacks of the range representation are outweighed by its advantages.

6.4 Test Problems

A suite of four test problems is used to benchmark the speciation algorithm in comparison to canonical EC. The first four functions are 20 bit problems while the fourth is a 50 bit 1-D Ising model [48]. For each of the first three functions, the binary strings are decoded into the real number range $[0, 100]$.

6.4.1 Multimodal Functions

The first test function is a Gaussian centered at 50 with a standard deviation of 12 as shown in Figure 6.1. This is an unimodal function that is easily solved by canonical EC. The second problem extends the first to a multimodal function with Gaussians spaced in 20 unit increments as shown in Figure 6.2. This serves as a first test of whether the developed speciation mechanisms can in fact form species in correspondence with the multimodal nature of the performance landscape. The third problem complicates the picture by varying the heights of the peaks as illustrated in Figure 6.3. The analytic forms of the three test functions are given in Table 6.1 where $G(\mu, \sigma)$ denotes the Gaussian with mean μ and variance σ . Functions 2 and 3 are analogous to the test functions developed by Beasley for niching [7].

| No. | Function |
|-----|---|
| 1 | $G(50, 12)$ |
| 2 | $G(10, 12) + G(30, 12) + G(50, 12) + G(70, 12) + G(90, 12)$ |
| 3 | $.9G(10, 12) + .7G(30, 12) + G(50, 12) + .5G(70, 12) + .3G(90, 12)$ |

Table 6.1: Speciation test functions.

RESULTS

Each EC implementation was run 30 times on the four test functions. In every case, the probability of mutation was set to zero in order to determine the effect of the reinforcement learning induced speciation. Comparisons are made to a standard, canonical EC implementation without any speciation mechanisms. For the first three test functions, a population size of 75 is chosen.

Function 1

Convergence results for the unimodal test function are shown in Table 6.2. μ indicates the mean number of iterations before the global optimum is found, σ is the standard deviation, and **Median** is the median value. Note that the number of iterations for the speciation case actually refers to number of iterations in which crossover occurred. Because mutation is zero, when crossover does not occur, no variation or search is accomplished. Not surprisingly, the standard EC performs better than the developed speciation approach. The difference in performance is marginal, particularly for

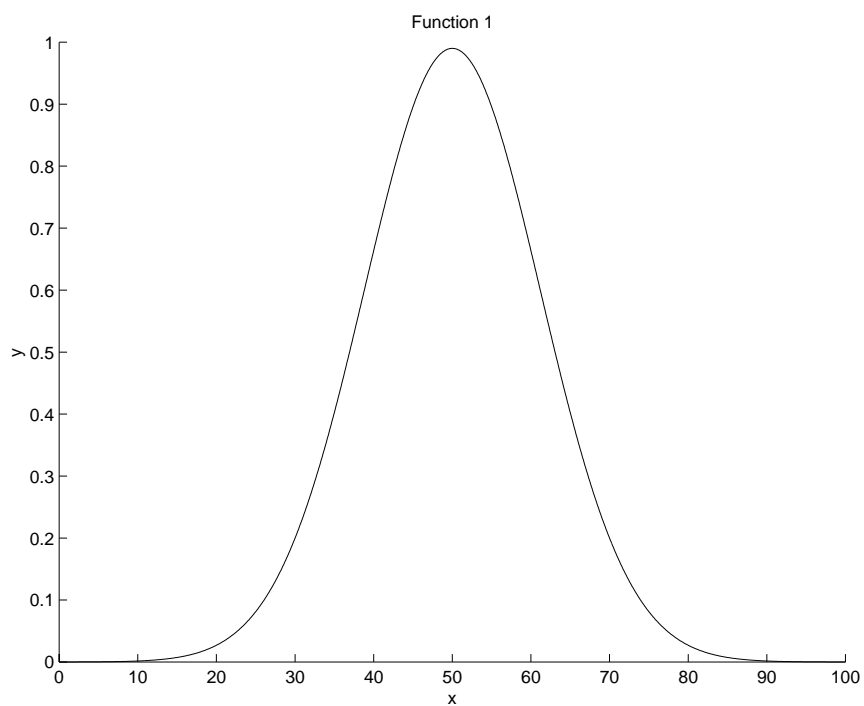


Figure 6.1: Function 1: Unimodal test function.

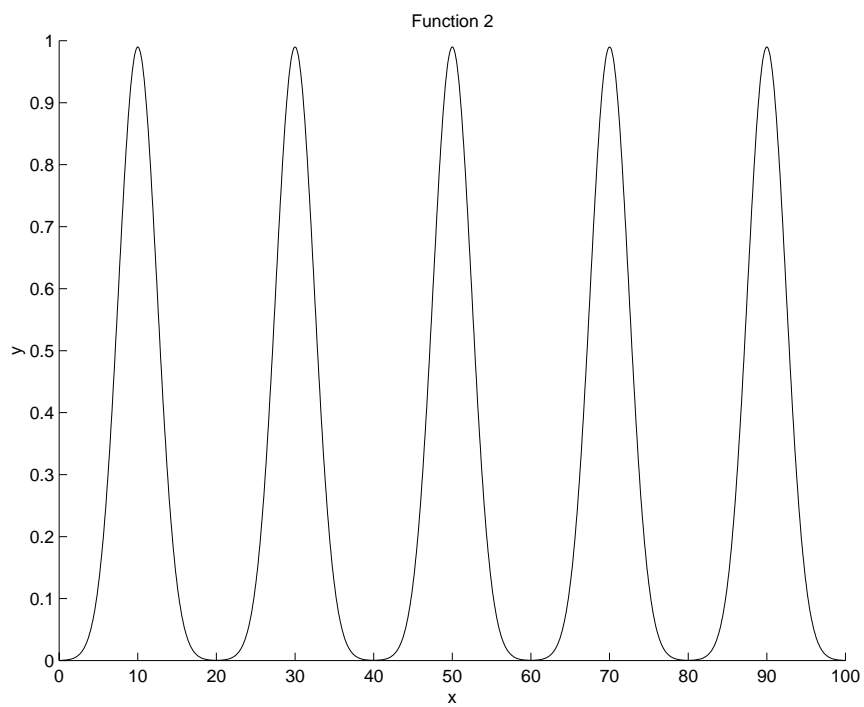


Figure 6.2: Function 2: Multimodal test function with equal peaks.

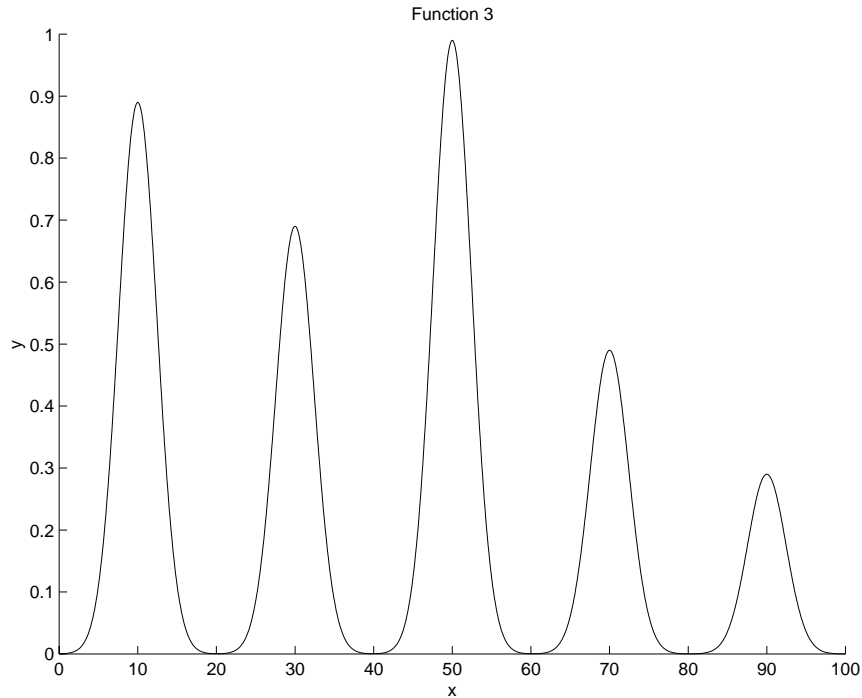


Figure 6.3: Function 3: Multimodal test function with different peaks.

| | μ | σ | Median |
|-------------------|-------|----------|---------------|
| Speciation | 250.2 | 228.1 | 187 |
| Standard | 206.6 | 152.2 | 178 |

Table 6.2: Convergence results for Function 1.

the median values. It is believed that speciation performs slightly less well because of unfortuitous selection of initial index ranges. Figures 6.4 and 6.5 show a typical final population after 675 iterations (roughly 3σ away from the mean time to finding the optimal solution; so, the global optimum is very likely to be found in these runs), for standard and speciation EC respectively. Qualitatively, it is easy to see that speciation is better able to maintain solution diversity. Typical evolved index ranges are shown in Figure 6.6. This figure is somewhat complicated and shows two things, each individual's index range and the corresponding x value divided by 25 (the population members are sorted according to ascending index value). The index range is represented by the horizontal line and the corresponding * indicates the x value. It is easy to see speciation in which the outlying x values have index ranges that do not overlap, and hence don't mate, with the individuals with x values centered around the optimum.

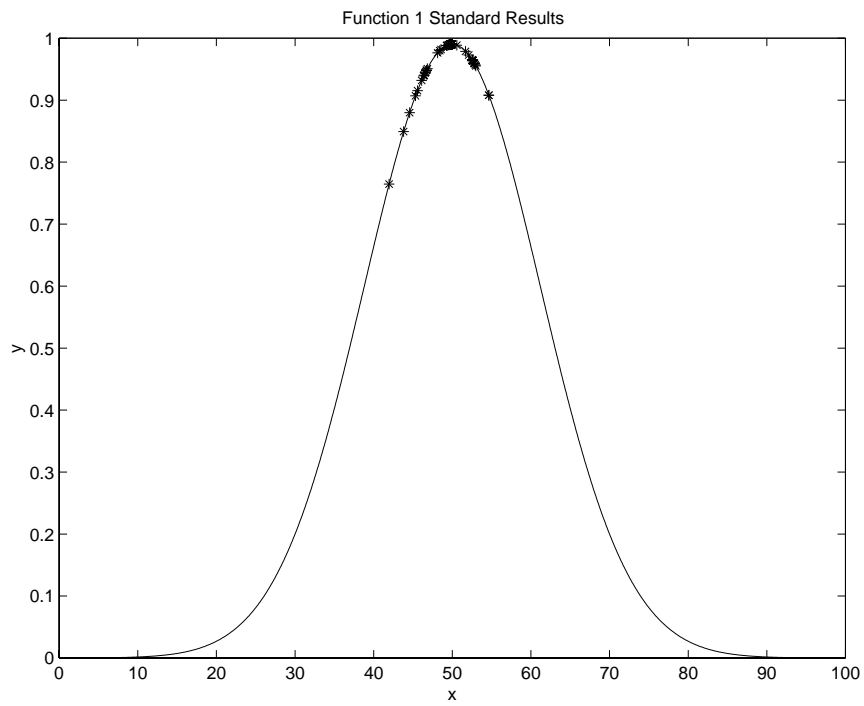


Figure 6.4: Function 1: Typical evolved solutions for standard EC.

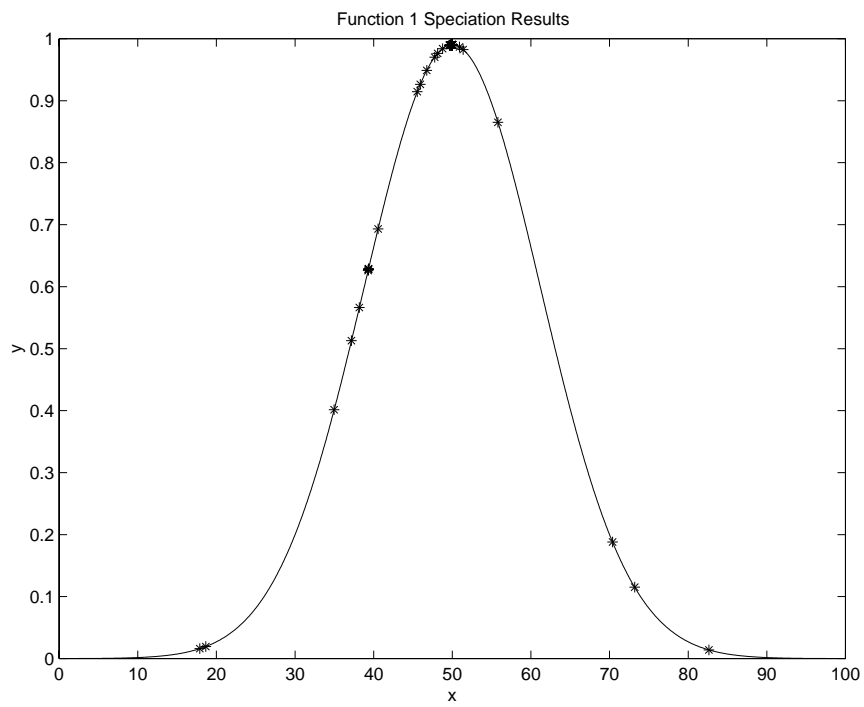


Figure 6.5: Function 1: Typical evolved solutions for speciation EC.

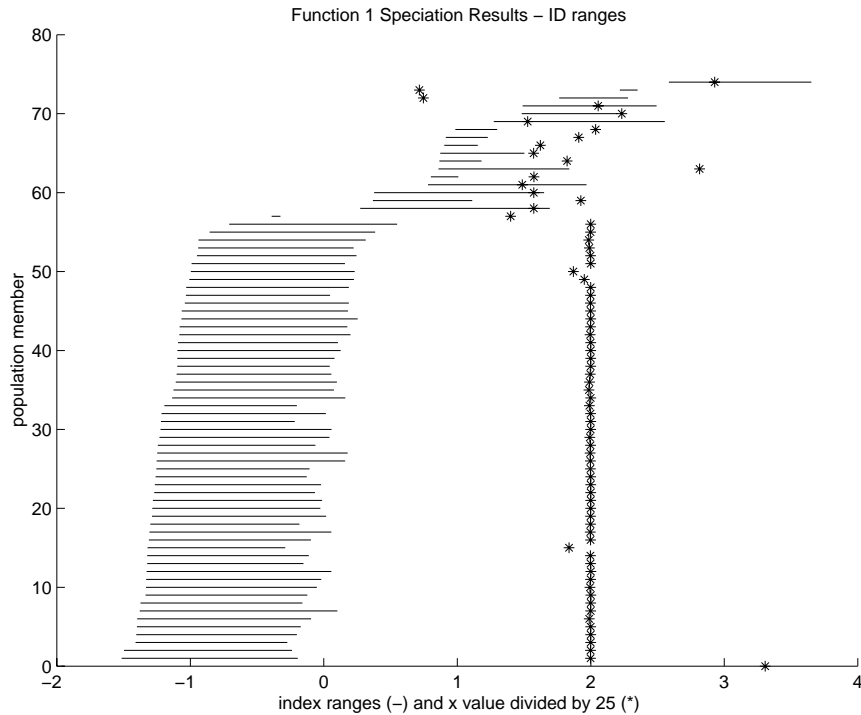


Figure 6.6: Function 1: Typical evolved index ranges for speciation EC.

Function 2

Convergence results for the multimodal test function with equal peaks are shown Table 6.3. Again, the results are not markedly different for speciation and standard EC. However, speciation does seem to be more effective at search. In particular, its median time to finding a global optimum is considerably lower than the standard case. The skewness of the convergence results in which median values are less than average values implies that speciation often significantly outperforms standard EC. This presumably is a result of the effective mating characteristics that are evolved through speciation. Typical evolved solutions for standard and speciation EC are shown in Figures 6.7 and 6.8 respectively (again, these were run for 675 generations each). It can clearly be seen that speciation is able to not only maintain diversity, but also to find all the optima. The corresponding index ranges for the speciation approach are shown in Figure 6.9. The index ranges show that each peak has evolved its own species in which it cannot interact with solutions from other x values. There are two interesting phenomena that can be observed in this figure. First, it seems that speciation is not wholly immune to genetic drift as the majority of individuals have converged to a single peak. Second, speciation can be accomplished by either having non-overlapping index ranges or by having a relatively small index range – effectively a species with a single individual.

| | μ | σ | Median |
|-------------------|-------|----------|---------------|
| Speciation | 218.9 | 175.7 | 175.5 |
| Standard | 234.9 | 115.1 | 247.5 |

Table 6.3: Convergence results for Function 2.

This is just a manifestation of Equation 6.1, which states that the probability of mating is equal to the intersection over the union of the index ranges. Thus, if an individual has a relatively small index range, it is quite unlikely to mate and as such constitutes a non-mating species. Unfortunately, this can be problematic because if the individual happens to be a poor solution, it will always be retained by the population.

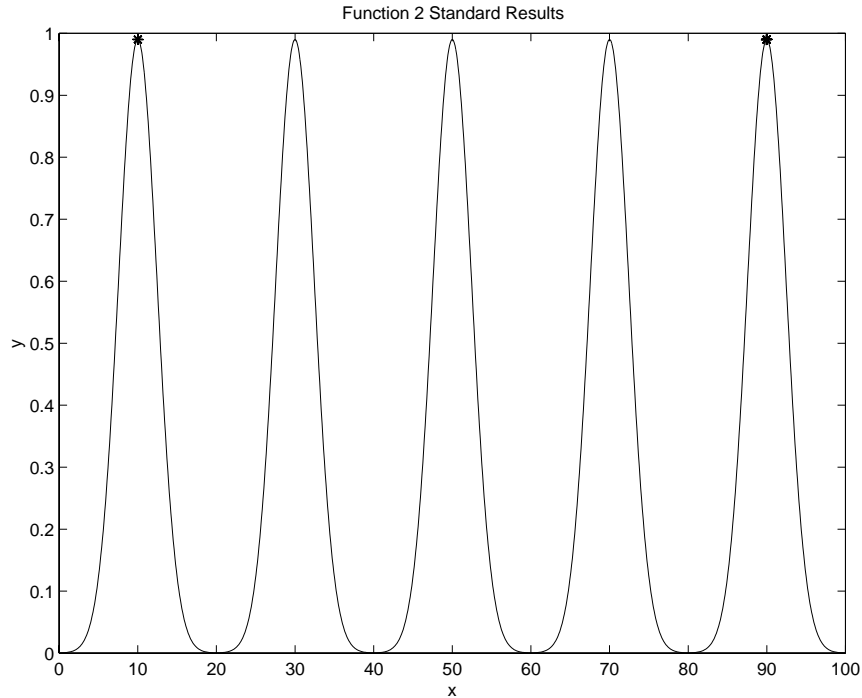


Figure 6.7: Function 2: Typical evolved solutions for standard EC.

Function 3

Convergence results for the multimodal test function with unequal peaks are shown Table 6.4. In contrast to the previous results, speciation significantly outperforms standard EC. This supports the hypothesis that speciation is able to evolve the appropriate mating characteristics. Typical evolved

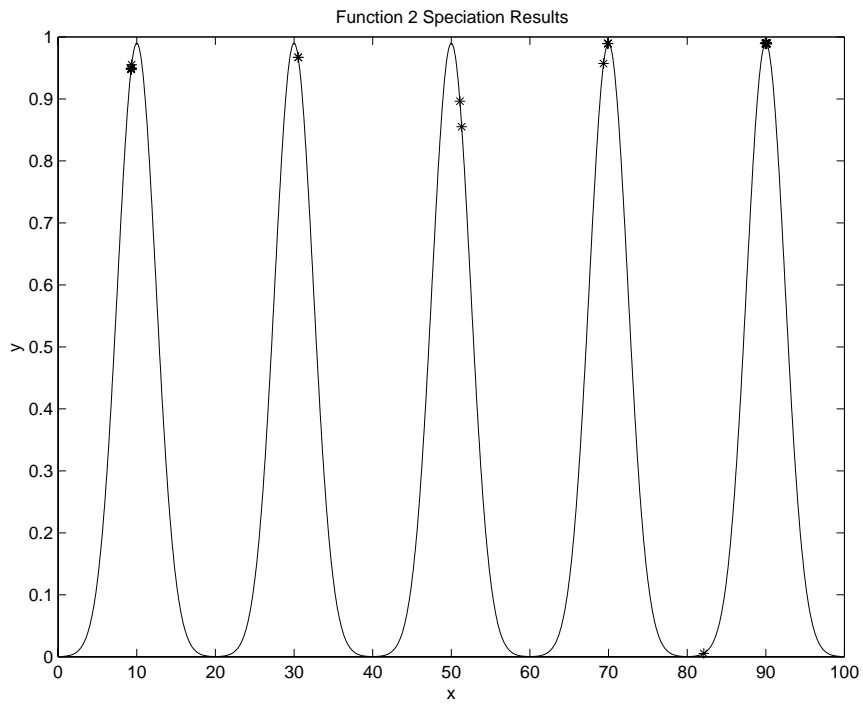


Figure 6.8: Function 2: Typical evolved solutions for speciation EC.

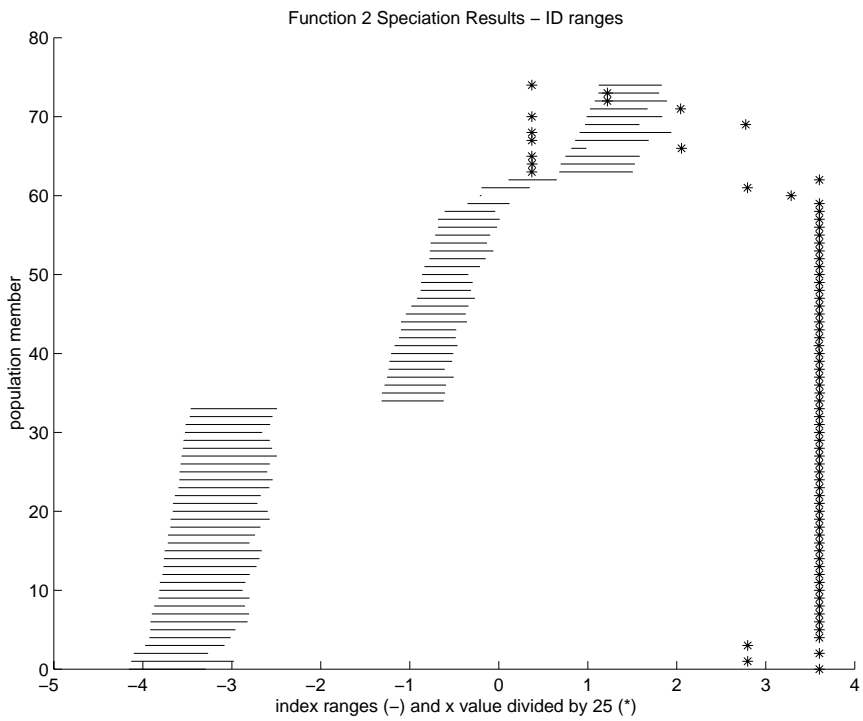


Figure 6.9: Function 2: Typical evolved index ranges for speciation EC.

| | μ | σ | Median |
|-------------------|-------|----------|---------------|
| Speciation | 474.3 | 350.3 | 478.5 |
| Standard | 593.6 | 334.6 | 538.5 |

Table 6.4: Convergence results for Function 3.

solutions for standard and speciation EC are shown in Figures 6.10 and 6.11 respectively (these were run for 1600 generations, or 3σ away from the mean convergence time). While speciation is still able to maintain more diversity than standard EC, it is not able to maintain solutions at every peak. It is apparent that the global optimum is able to dominate the population. Furthermore, poor solutions have been propagated because of small relative index ranges as described previously.

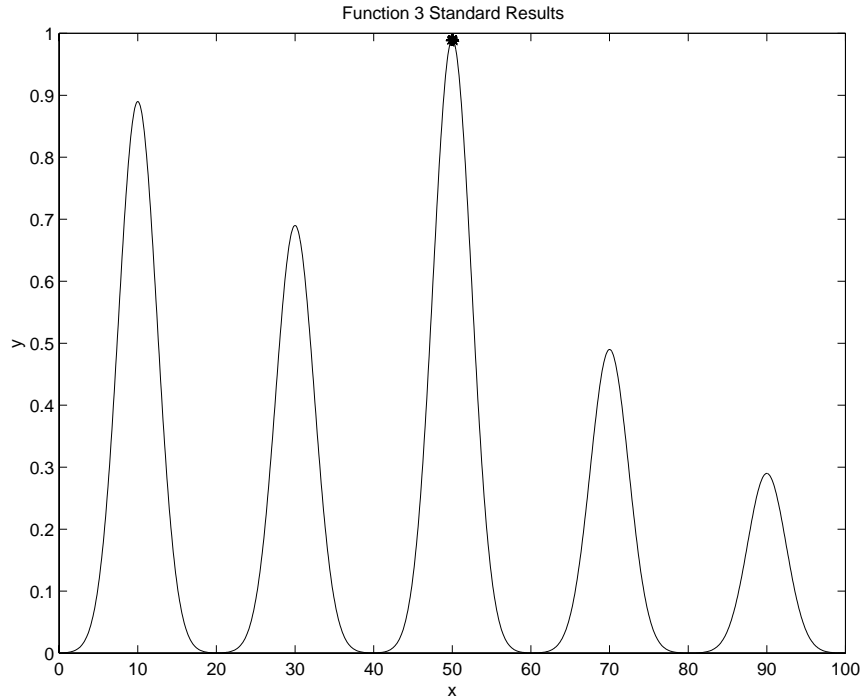


Figure 6.10: Function 3: Typical evolved solutions for standard EC.

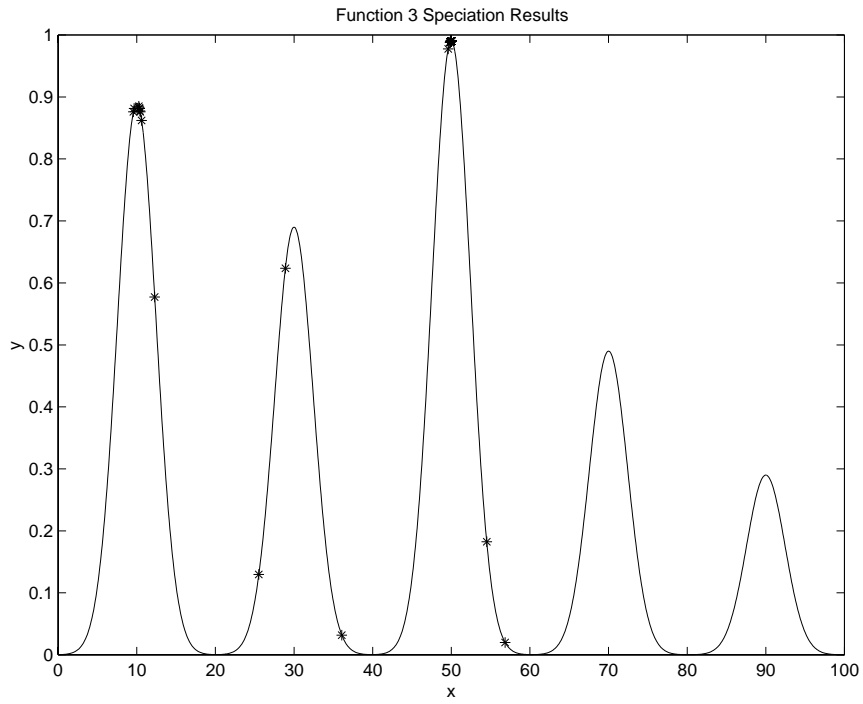


Figure 6.11: Function 3: Typical evolved solutions for speciation EC.

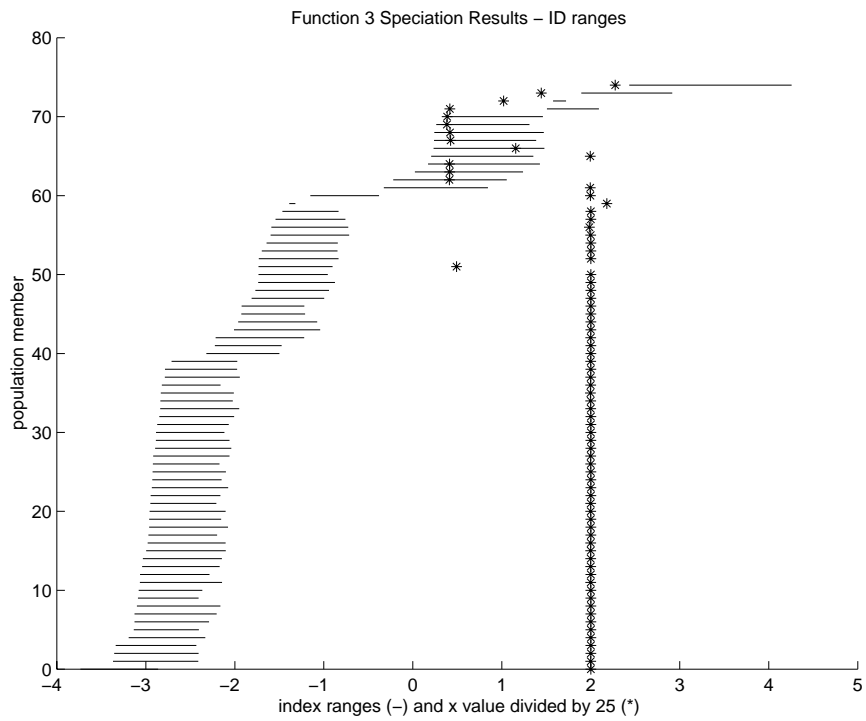


Figure 6.12: Function 3: Typical evolved index ranges for speciation EC.

6.4.2 1-D Ising Model

The fourth test problem is the 50 bit one-dimensional Ising model as found in [48]. The one-dimensional Ising model is a nearest neighbor interaction function and is given as

$$f : \{0, 1\}^n \rightarrow \mathbb{R} : x \mapsto \sum_{i=1}^n \delta(x_i, x_{i+1}) \quad (6.4)$$

where n is the number of bits (*i.e.*, 50 in the problem tested here), $x_{n+1} \equiv x_1$, and

$$\delta(x_i, x_j) = \begin{cases} 1 & \text{if } x_i = x_j; \\ -1 & \text{otherwise.} \end{cases} \quad (6.5)$$

In words, the fitness of the Ising model is the number of instances in which adjacent neighbors have the same value, or “spin”, minus the number of adjacent neighbors with different spins. Figure 6.13 shows a qualitative plot of average fitness versus the number of 1’s in the bit string. It can clearly be seen that there are two global optima, which are the string of either all 0’s or all 1’s. VanHoyweghen has shown that this problem requires “niching” (equivalently speciation) in the absence of mutation in order to find either of the globally optimal solutions [86]. If mate preference is not restricted to similar individuals, then strings with mostly zeroes can recombine with those with mostly ones, leading to poorly performing individuals and convergence to suboptimal strings. The spin-flip symmetry of the Ising model (*i.e.*, reversing the spin of each bit does not change the fitness value) necessitates a speciation approach such that conventional EC has difficulties in finding the global optima of the Ising model.

RESULTS

Convergence results for the 50 bit 1-D Ising model are shown Table 6.5. Populations of 250 were used in these tests. The **Replacement** heading will be explained later. The results for both speciation and standard EC are nearly equivalent with speciation showing slightly better performance and less variation in its time to finding an optimal solution. This is somewhat troubling as VanHoyweghen showed that the Ising model requires niching for EC that do not employ mutation [86]. The reasons for this discrepancy can be explained by taking a closer look at the speciation results. Typical evolved index ranges are presented in Figure 6.14. Also shown in this figure are the number of ones per individual divided by 10. These results clearly show that speciation determined the appropriate

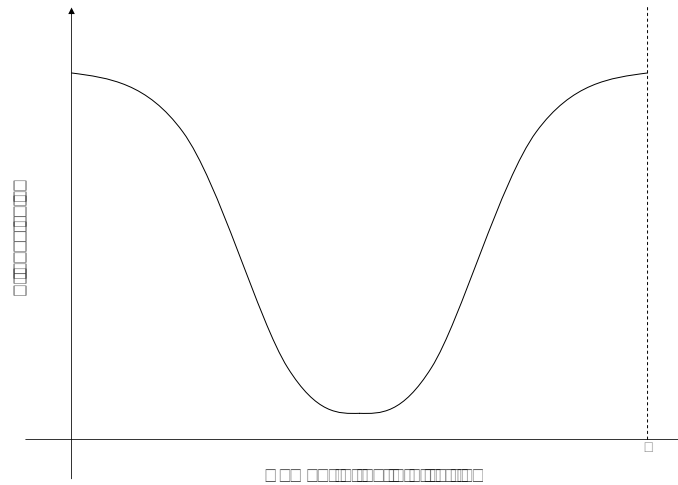


Figure 6.13: Qualitative diagram of the 1-D Ising model.

mating characteristics, as solutions with mostly zeroes cannot intermix with those with many ones. Unfortunately, because a steady state selection algorithm is implemented in which offspring may only replace direct parents, there is no mechanism for removing poorly performing individuals if they are ever able to form their own species as alluded to in the previous sections (recall also that there is no mutation). Because of this and the presence of other poorly performing individuals, many crossover operations are wasted on such individuals, leading to longer convergence times. To determine if this claim is true, a removal mechanism was implemented. Every 1500 generations, the worst individual was replaced by the best individual. Results are shown in Table 6.4.2 under the **Replacement** heading. The results support the claim that the delayed convergence times are a result of maintenance of poor individuals. So, it seems appropriate to retain this replacement mechanism in future speciation implementations. However, by replacing poorly performing individuals, there is always a chance that a local optimum is lost; and, if the goal of the design or search was to find local optima, then such a replacement initiative may be inappropriate.

| | μ | σ | Median |
|--------------------|-------|----------|---------------|
| Speciation | 23245 | 7776 | 22157 |
| Standard | 24738 | 10533 | 21019 |
| Replacement | 21042 | 7570 | 19974 |

Table 6.5: Convergence results for 1-D Ising model.

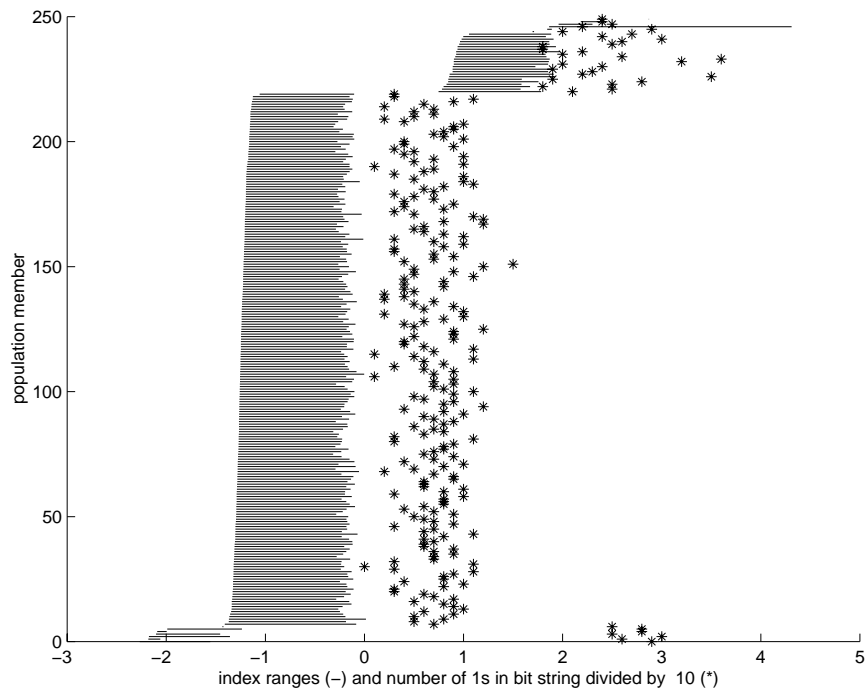


Figure 6.14: Ising Model: Typical evolved index ranges.

6.5 Design Problem

The neural network design problem presented in this section is the same as that introduced in Chapter 5. The linkage learning approaches of that chapter are easily extended to handle speciation in the previously described manner. New results with speciation are presented and compared to the previous results.

6.5.1 Design of Artificial Neural Networks

For implementation details, please refer back to Section 5.5.1. The only modification is an extension of the bounding index ranges for use with the speciation mechanisms developed in this chapter.

Data from five test functions are presented for evolution of the artificial neural network. These are the same as in Section 5.5.1 and are reproduced here.

| No. | Function |
|-----|--------------------|
| 1 | $y = x$ |
| 2 | $y = \tanh(x + 2)$ |
| 3 | $y = \sin(x)$ |
| 4 | $y = \cos(x)$ |
| 5 | $y = x $ |

Table 6.6: Neural network test functions.

Results of the designs evolved using speciation are presented in comparison to the linkage learning results as shown in Table 6.7. Speciation evolves considerably poorer solutions for Function 1 and 2, and significantly better solutions for Functions 3–5. These seemingly anomalous results can be explained through comparison of the evolved solutions. For both Functions 1 and 2, uniform and uniform range crossover were able to evolve fairly parsimonious solutions. In particular, Function 1 typically resulted in solutions with the globally optimal single connection between input and output nodes. However, for the case of speciation, while good approximations were achieved, parsimony was not as well enforced as nearly all evolved solutions had at least one hanging node or connection. Because of the excess edges which are penalized, the fitness results for speciation for Functions 1 and 2 were not quite as good as those for uniform and uniform range. This can be attributed to the fact that speciation leads to subpopulations that have fewer avenues of modification and removal of

extraneous nodes and edges because of their smaller mating pool.

Alternatively, for Functions 3–5 uniform and uniform range crossover evolved either parsimonious linear mappings or non-parsimonious nonlinear mappings. For these same functions, speciation was able to effectively avoid linear mappings to evolve good approximations on nearly every occasion. So, while speciation may not evolve more parsimonious solutions, it seems to be more effective in finding good approximations. The results support the claim that speciation is able to evolve good mating combinations, which leads to the better performance.

| | Speciation | | Uniform | | Uniform Range | |
|-----------------|-------------------|----------|----------------|----------|----------------------|----------|
| Function | μ | σ | μ | σ | μ | σ |
| 1 | 1.1308 | 0.1084 | 1.0780 | 0.0824 | 1.0682 | 0.0608 |
| 2 | 2.5967 | 0.4410 | 2.3401 | 0.4481 | 1.9329 | 0.6915 |
| 3 | 4.1113 | 0.9157 | 4.7067 | 0.8589 | 4.4077 | 0.7467 |
| 4 | 5.7915 | 0.3388 | 5.8806 | 0.2904 | 5.8363 | 0.4184 |
| 5 | 8.2032 | 1.3094 | 11.8868 | 1.5419 | 8.466 | 1.3415 |

Table 6.7: Best evolved neural network fitness after 15,000 generations.

6.6 Summary and Discussion

Index ranges were introduced in Chapter 5 to enable linkage learning. However, it was shown that because of the curse of dimensionality, as the dimensionality of index space increased, overlap between any two chromosomes becomes less and less likely. If two non-overlapping individuals mate, then crossover is imbalanced because only a single individual contributes genes to the other. In an attempt to alleviate this imbalance, the idea was contrived to prevent matings between such non-overlapping individuals. This idea was then developed in more detail to allow speciation in populations.

The basic premise was to use the bounding index ranges of an individual as a representation of the individual's area of habitation. Thus, individuals can only mate with individuals with overlapping areas of inhabitation. The probability of mating is proportional to the intersection of the inhabited areas over the union. The areas of habitation are then modified according to the fitness of the offspring. The idea was to reinforce good matings by making the areas more similar and punish

poor matings by pushing the parents further apart. Such an approach led to effective speciation mechanisms as verified by empirical results on test and design problems.

The developed speciation representation is a highly flexible representation that can be easily modified to mimic other well known speciation, or diversity preserving, population structures. For example, island models (in which individuals within the same island are freely interacting and individuals on different islands are not) can easily be represented by giving the same range of inhabitation to individuals on the same island. If migratory effects are desired, overlaps of the proper proportion can be made. Of course, certain island topologies cannot be represented as discussed previously in Section 5.5.1. This does not present a major difficulty because the inaccessible topologies are highly atypical.

There are three other noticeable characteristics of the developed speciation that present advantages as compared to other methods. The first is that both graded and discontinuous speciation can occur simultaneously. This is a middle ground between the island models and cellular models that has the advantages of both. Moreover, because individuals have transitory inhabitation ranges, emergent migratory behaviors are evolved. There is no need to specify migration probabilities in advance. Mutation and crossover probabilities can also be evolved by simply ensuring that each operation is mutually exclusive. For example, if two individuals do not recombine or mate, then they mutate instead. Thus, operator probabilities can be evolved rather specified *a priori* without any knowledge of what good probabilities may be. Such an approach is reminiscent of memetic algorithms in which individuals can learn on their own (*i.e.*, mutation) or learn through exchange of knowledge with other individuals (*i.e.*, recombination). This is a topic of future research – in which individual learning can be introduced into the speciation approach for more effective evolution.

Chapter 7

Conclusion

7.1 Summary

Although engineering design has always been associated with human creativity and skill, the generation of designs can be formalized in a structured and algorithmic manner. Such a formalization of design synthesis enables automatic design synthesis through computation. Inspiration for the design algorithm is taken from biological evolution, which has been automatically generating new designs for 3.5 billion years. The evolution algorithm consists of an iterative loop of transmission, variation, and selection of individuals with certain traits. Because evolution of good designs obviously occurs, the evolution algorithm is equated to the formalization of design synthesis. Modification of the evolution algorithm for use on computational substrates results in the evolutionary computation (EC) framework that can be and is used for automatic design synthesis.

Three universal aspects of good designs have been identified and discussed. These are variable complexity, modularity, and speciation. Placed within an evolutionary framework, the reasons for their universality are readily apparent; they enable more evolvable solutions. Because designs are subject to a dynamic environment, they must be able to change with the environment. Those that have a better ability to evolve (otherwise known as evolvability) can evolve more quickly in concert with the dynamic environment and thus have a better chance for continued survival. In terms of EC, the environment changes every generation as new populations exert different competitive pressures on each individual. Thus, if the performance landscape is fixed in time, then more evolvable solutions will lead to quicker convergence. In terms of design synthesis, this means shorter design cycles. Furthermore, modularity and speciation can be used to determine relationships between different design characteristics.

A representation and corresponding crossover operator have been developed to address each of the three identified universal characteristics. Variable complexity is achieved by extending the identifying index values of each gene to real numbers. An analogous crossover operator extends ranges to real valued intervals. In combination with random initialization of gene indices, crossover will induce length changes. This development is termed the variable length representation (VLR).

Closer inspection of VLR revealed that certain genes are more tightly linked to other genes because their indices were closer together. Tighter linkage means higher probability of remaining together after a crossover event. An index range representation (RR) is developed to enable linkage or modularity learning. Results are initially good on a set of test problems, but are poor on the design problems. This was because crossover ranges were too restrictive and could not separate tightly linked genes that had poor traits. An uniform range crossover operator (URC) is developed to overcome these issues and the results are promising. Rather than retaining linkages between genes on the same parent, URC maintains linkages between genes on different genes. The idea behind URC is to only swap genes with similar index ranges in the opposing parents. So, linkage in URC is the probability of swapping two genes in a crossover event rather than the previously given definition. These results show that the developed RR is capable of both types of linkage learning, when appropriate. However, at the same time, it cautions against blind application of EC.

The range representation is found to lose its effectiveness as the dimension of the index space increases. The culprit is the curse of dimensionality in which hypervolumes increase exponentially, leading to exponential decreases in the probability that two randomly generated chromosomes have overlapping index ranges. This is exactly speciation, in which individuals only interact with other individuals of their own subpopulation or species. An approach based on reinforcement learning is developed to evolve the appropriate speciation characteristics. Results on both test and design problems are positive with species forming at different peaks in the performance landscape. What perhaps is not apparent is that speciation approach is an extension of the RR modularity that is in turn an extension of the VLR. Hence, the developed speciation approach can be used to achieve all three design characteristics, or any combination thereof.

7.2 Discussion and Future Work

Unfortunately, or fortunately as the case may be, the work done in this thesis raises nearly as many questions as it answers. These are discussed subsequently. Also, a discussion and summary of future

work concludes each of the chapters on variable complexity, modularity, and speciation. Please refer to the appropriate sections for more details.

First and foremost, the class of problems for which the developed approaches are applicable need to be more clearly defined. This is true in general for all EC. The difficulty in both cases is that the search path depends on the interaction between solution encoding, which determines the performance landscape, and variation operator, which determines how performance landscape is traversed. Often these have to be delicately balanced through human intervention to achieve acceptable convergence rates, as seen by the need to introduce URC in replacement of RR in the design of neural networks and truss systems. Matters are further complicated because typical design problems do not have straightforward solution encodings that lend themselves to efficient search traversal by canonical variation operators. There are several possible alternatives to having hand tuned solution encodings and variation operators. As the trend is towards more autonomous search algorithms, choice of solution encoding and operators can be left to evolution. This still leaves unresolved issues because human intervention is needed to tell the EC what the “alphabet” of encodings and operators are. Evolution of operators has some precedent as seen by self-adaptive mutation rates that are able to evolve the local search radii of each individual.

Not surprisingly, evolution of solution encodings and variation operators have precedents in nature. As more complex organisms evolved, their encodings also became more complex. For example, the number of chromosomes, or high level modules, for human beings far exceeds that for single celled organisms. Along with this increased genome modularity and re-use was the development and domination of crossover as the primary variation operator. Mutation was relegated from the primary operator in asexual organisms to the secondary operator in sexual organisms. These observations suggest that incremental evolution from low to high complexity encodings with corresponding changes in variation operators and probabilities is a viable solution to online evolution of encodings and operators.

A more reachable near term answer for identifying problems amenable to solution by EC is to catalogue all the different problems that have been solved by EC. Then, new problems can be tackled by initially starting with the solution encodings and variation operators of similar problems. The difficulty, of course, is that no standardization of EC exists. In this regard, the approach developed in this thesis seems to be good candidate to be the standard EC implementation. There are three reasons that the developed approach is a good candidate. These are

1. Variable complexity, modularity, and speciation can all be handled together or separately in

any combination.

2. The range representation can encode most any other linkage representation. Moreover, because of the local interaction between genes, it does not suffer as dramatically as matrix representations when the number of genes and individuals increases.
3. The range representation for speciation can be used to encode most other typical connection topologies (*e.g.*, island and cellular models). Again, local interactions between individuals permits the number of connection parameters to grow linearly with population size rather than exponentially for global, matrix approaches.

Although this may be a more reachable near term solution, the future lies in being able to evolve solution encodings and their respective variation operators.

Appendix A

Source Code

Included in the following are the prototypes of the code that was written to develop the appropriate evolutionary computation methods. Details that vary from problem to problem have been stripped out for clarity. The main program that runs the evolution is shown first and is entitled main.cxx. Obviously, all the code has been written in C++.

```
#include <algo.h>
#include <vector.h>
#include <math.h>

main() {
    //-- random seed based on time
    Seed();

    //-- variable declarations
    int N = 100;           //-- population size
    Chromo pop[N];        //-- population
    double fit[N];        //-- fitness of each population member
    Chromo off[2];        //-- from steady state selection
    double off_fit[2];    //-- fitness of generated offspring
    int max_generations = 1000;
    int i, j, k;
    int index[2];
    double p_mut = 0.25;
```

```

/-- initialize the population
Initialize( pop[N] );

/-- determine fitness of each individual
for ( i = 0; i < N; i++ )
    fit[i] = Evaluate( pop[i] );

/-- actual evolution - uses steady-state replacement
for ( i = 1; i < max_generations; i++ ) {
    /-- select parents at random and copy into offspring
    for ( j = 0; j < 2; j++ ) {
        index[j] = int( drand48() * N );
        off[j]    = pop[index[j]];
    }

    /-- crossover with probability equal to index range overlap
    if ( drand48() < Overlap( off[0], off[1] )
        Crossover( off[0], off[1] );

    /-- mutate with probability p_mut and calculate fitness
    for ( j = 0; j < 2; j++ ) {
        if ( drand48() < p_mut )
            off[j].Mutate();

        off_fit[j] = Evaluate( off[j] );
    }

    /-- the steady-state replacement - minimization problem
    for ( j = 0; j < 2; j++ )
        if ( off_fit[j] <= fit[index[j]] ) {

```

```

    pop[index[j]] = off[j];
    fit[index[j]] = off_fit[j];
}

/-- reinforcement learning
if (off_fit[0]<=fit[index[0]] && off_fit[1]<=fit[index[1]]){

    for ( j = 0; j < 2; j++ )
        AttractID( pop[index[j]], off[j], off[(j+1)%2] );

} else if ( off_fit[0] > fit[index[0]] &&
           off_fit[1] > fit[index[1]] ) {

    for ( j = 0; j < 2; j++ )
        OpposeID( pop[index[j]], off[j], off[(j+1)%2] );
}
}
}

```

The random number generator function, `drand48()`, needs a seed, which is randomized according to the program execution time. The random seed generator is shown below. `drand48()` generates a random number uniformly from the range 0 to 1.

```

#include <time.h>

void Seed() {
    int seed;
    time_t now;

    time(&now);
    seed = (unsigned int) now;
    srand48(seed);
}

```

The code for a Gaussian random variable with unit variance is shown below. The function call implements the so-called polarization technique.

```
double Dist() {
    double x1, x2, w, y1, y2;

    do {
        x1 = 2 * drand48() - 1;
        x2 = 2 * drand48() - 1;
        w = x1 * x1 + x2 * x2 );
    } while ( w >= 1.0 );

    w = sqrt ( (-2 * log(w)) / w );

    y1 = x1 * w;
    y2 = x2 * w;

    if ( drand48() > 0.5 ) {
        return y1;
    } else {
        return y2;
    }
}
```

Chromosomes are composed of genes. Each of the class prototypes are shown below.

```
//-- Gene Class prototype
class Gene {
public:
    Gene& operator=( const Genome& ); //-- copy operator
    void Initialize();
    void Mutate();
    void MutateID();
public:
```

```

    type p;                                //-- gene parameter(s)
    double id[2];                            //-- indices (1-D space)
};

//-- Chromo Class prototype
class Chromo {
public:
    Chromo& operator=( const Chromo& );    //-- copy operator
    void Initialize();
    void Mutate();
    void MutateID();
    void Erase();                          //-- clears all genes
    void ShiftID( double );                //-- ID translation function
public:
    vector<Gene> g;                          //-- genes in chromosome
    double id[2];                            //-- index range bounds
}

```

The details of `Initialize()` and `Mutate()` are problem dependent and aren't shown in the above. This is true with the evaluation function as well. Crossover requires an overlap operator that determines the amount of overlap over the union of ranges. A simple implementation is developed below.

```

double Overlap( double id1[], double id2[] ) {
    double I = 0, U = 0;    //-- intersection and union variables
    double r[2], id[2];

    //-- determine difference between lower and upper bounds
    r[0] = id1[1] - id2[0];
    r[1] = id2[1] - id1[0];

    //-- if negative ranges, no overlap

```

```

if ( r[0] < 0 || r[1] < 0 )
    return 0;

/-- determine the max upper and min lower bounds
id[0] = id1[0] < id2[0] ? id1[0] : id2[0];
id[1] = id1[1] > id2[1] ? id1[1] : id2[1];

U = id[1] - id[0];
I = (r[0] + r[1]) - U;

return (I/U);
}

```

The above determines the crossover probability and also determines whether or not a gene lies within a given range if thresholded into a boolean function. This enables application of the crossover operator for variable length search as shown.

```

void Crossover( Chromo& C1, Chromo& C2 ) {
    int n1, n2;
    Chromo c1, c2;    /-- temp offspring
    double id[2], swap;
    double bounds[2];
    int i, j;

    c1 = C1;
    n1 = c1.g.size();

    c2 = C2;
    n2 = c2.g.size();

    /-- determine index range bounds
    bound[0] = C1.id[0] < C2.id[0] ? C1.id[0] : C2.id[0];
    bound[1] = C1.id[1] > C2.id[1] ? C1.id[1] : C2.id[1];
}

```

```
//-- select crossover range from bounds
for ( i = 0; i < 2; i++ )
    id[i] = bound[0] + drand48() * (bound[1] - bound[0])

//-- swap ids if necessary
if ( id[0] > id[1] ) {
    swap = id[0];
    id[0] = id[1];
    id[1] = swap;
}

//-- perform the crossover
for ( i = 0; i < n1; i++ )
    if ( Overlap( id, C1.g[i].id ) ) {
        c2.push_back( C1.g[i] );
    } else {
        c1.push_back( C1.g[i] );
    }

for ( i = 0; i < n2; i++ )
    if ( Overlap( id, C2.g[i].id ) ) {
        c1.push_back( C2.g[i] );
    } else {
        c2.push_back( C2.g[i] );
    }

//-- erase the originals
C1.Erase();
C2.Erase();

//-- copy temp into originals
```

```

C1 = c1;
C2 = c2;
}

```

All that remains are the reinforcement learning implementations that are shown below.

```

/---- Attracts IDs, P is the modified C1
void AttractID( Chromo& P, Chromo& C1, Chromo& C2 ) {
    double wt;

    /-- use a noisy weighted mean adjustment
    for ( int i = 0; i < 2; i++ ) {
        wt = 1.25 + 1.5 * drand48();
        P.id[i] = ( wt * C1.id[i] + C2.id[i] ) / ( 1 + wt );
    }
}

```

```

/---- Repels IDs, P is modified C1
void OpposeID( Chromo& P, Chromo& C1, Chromo& C2 ) {
    double shift;

    /-- use a noisy value to oppose IDs
    shift = 0.1 + 0.1 * fabs( Dist() );
    if ( C1.id[0] < C2.id[1] ) {
        P.ShiftID( -shift );
    } else {
        P.ShiftID( shift );
    }
}

```

Appendix B

Table of Abbreviations

| abbreviation | meaning |
|--------------|----------------------------|
| ANN | Artificial Neural Network |
| EC | Evolutionary Computation |
| EP | Evolutionary Programming |
| ES | Evolution Strategy |
| GA | Genetic Algorithm |
| GP | Genetic Programming |
| MDL | Minimum Description Length |
| MSE | Mean Square Error |
| NCR | Non-Coding Representation |
| NFL | No Free Lunch |
| RR | Range Representation |
| URC | Uniform Range Crossover |

Table B.1: Table of Abbreviations.

Bibliography

- [1] ALTENBERG, L. The evolution of evolvability in genetic programming. In *Advances in Genetic Programming* (Cambridge, MA, 1994), K. Kinneer, Jr., Ed., MIT Press, pp. 47–74.
- [2] ALTENBERG, L. Evolvability checkpoints against evolutionary pathologies. In *Evolvability Workshop at the 7th International Conference on Simulation and Synthesis of Living Systems (Artificial Life 7)* (2000).
- [3] ANTONSSON, E., AND CAGAN, J., Eds. *Formal Engineering Design Synthesis*. Cambridge University Press, Cambridge, UK, 2001.
- [4] ARKIN, E., CHEW, L., HUTTENLOCHER, D., KEDEM, K., AND MITCHELL, J. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13, 3 (1991), 209–215.
- [5] BACK, T. Self-adaptation. In *The Handbook of Evolutionary Computation* (1997), IOP Publishing and Oxford University Press, pp. 1–23.
- [6] BANZHAF, W., NORDIN, P., KELLER, R., AND FRANCONI, F. *Genetic Programming: An Introduction*. MIT Press, 1998.
- [7] BEASLEY, D., BULL, D., AND MARTIN, R. A sequential niche technique for multimodal function optimization. *Evolutionary Computation* 1, 2 (1993).
- [8] BENTLEY, P., Ed. *Evolutionary Design by Computers*. Morgan Kaufmann, New York, NY, 1999.
- [9] BENTLEY, P., Ed. *Creative Evolutionary Systems*. Morgan Kaufmann, New York, NY, 2001.
- [10] BENTLEY, P. J. *Generic Evolutionary Design of Solid Objects using a Genetic Algorithm*. PhD thesis, University of Huddersfield, U.K., 1996.
- [11] BURKE, D., DEJONG, K., GREFENSTETTE, J., RAMSEY, C., AND WU, A. Putting more genetics into genetic algorithms. *Evolutionary Computation* 6, 4 (1998), 387–410.
- [12] BUSH, G. Sympatric speciation in animals – new wine in old bottles. *Trends Ecol. Evol.* 9 (1994), 285–288.

- [13] CHAPMAN, C., AND JAKIELA, M. Genetic algorithm-based structural topology design with compliance and topology simplification considerations. *ASME Journal of Mechanical Design* 118, 1 (1996), 89–98.
- [14] CHAPMAN, C., JAKIELA, M., AND SAITOU, K. Genetic algorithms as an approach to configuration and topology design. *ASME Journal of Mechanical Design* 116, 4 (1994), 1005–1012.
- [15] CHARTRAND, G., AND LESNIAK, L. *Graphs and Digraphs*. Wadsworth, Inc., Monterey, CA, 1986.
- [16] CHERKASSKY, V., AND MULIER, F. *Learning from Data: Concepts, Theory, and Methods*. John Wiley, 1998.
- [17] COLLINS, R., AND JEFFERSON, D. Selection in massively parallel genetic algorithms. In *Proceedings of the 4th International Conference on Genetic Algorithms* (1991), Morgan-Kaufmann, pp. 249–256.
- [18] DARWIN, C. *On the Origin of Species by Means of Natural Selection: or the Preservation of Favoured Races in the Struggle for Life*. John Murray, 1859.
- [19] DAVIES, D., AND BOULDIN, D. A cluster separation measure. *IEEE Trans. on Pattern Analysis and Machine Intelligence* 1, 2 (1979), 224–227.
- [20] DAWKINS, R. *The Selfish Gene*. Oxford University Press, London, England, 1976.
- [21] DEB, K., AND GOLDBERG, D. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the 3rd International Conference on Genetic Algorithms* (1989), Morgan-Kaufmann, pp. 42–50.
- [22] DEJONG, K., SPEARS, W., AND GORDON, D. Using markov chains to analyze gafos. In *Foundations of Genetic Algorithms 3* (San Mateo, CA, 1995), D. Whitley and M. Vose, Eds., Morgan Kaufmann.
- [23] DIECKMANN, U., AND DOEBELI, M. On the origin of species by sympatric speciation. *Nature* 400 (1999), 354–357.
- [24] ERWIN, D., VALENTINE, J., AND JABLONSKI, D. The origin of animal body plans. *American Scientist* 85, 2 (1997).
- [25] FIGUEIRAPUJOL, J., AND POLI, R. Efficient evolution of asymmetric recurrent neural networks using a PDGP-inspired two-dimensional representation. In *Proceedings of the First European Workshop on Genetic Programming* (1998).
- [26] FOGEL, D. *Evolving Artificial Intelligence*. PhD thesis, University of California at San Diego, 1992.
- [27] FOGEL, D., Ed. *Evolutionary Computation: The Fossil Record*. IEEE Press, 1998.
- [28] FOGEL, D., FOGEL, L., AND PORTO, V. Evolving neural networks. *Biolog. Cybern.* 63 (1990), 487–493.

- [29] FOGEL, L. *On the Organization of Intellect*. PhD thesis, University of California at Los Angeles, 1964.
- [30] FOGEL, L., OWENS, A., AND WALSH, M. *Artificial Intelligence through Simulated Evolution*. John Wiley, 1966.
- [31] FORREST, S., AND MITCHELL, M. Relative building-block fitness and the building-block hypothesis. In *Proceedings of the Foundations of Genetic Algorithms Workshop* (1992).
- [32] FRANTI, P. Genetic algorithm with deterministic crossover for vector quantization. *Pattern Recognition Letters* 21, 1 (2000), 61–68.
- [33] FREEMAN, J., AND SKAPURA, D. *Neural Networks: Algorithms, Applications, and Programming Techniques*. Addison-Wesley Publishing Company, Reading, MA, 1991.
- [34] FUNES, P., AND POLLACK, J. Computer evolution of buildable objects. In *Fourth European Conference on Artificial Life* (Cambridge, MA, 1997), P. Husbands and I. Harvey, Eds., MIT Press, pp. 358–367.
- [35] GAVIN, H. FRAME: software for static and dynamic structural analysis of 2D and 3D frames and trusses with elastic and geometric stiffness. <http://www.duke.edu/hpgavin/frame> (2001).
- [36] GHOZEIL, A., AND FOGEL, D. Discovering patterns in spatial data using evolutionary programming. In *Genetic Programming 1996: Proceedings of the first annual conference* (Cambridge, MA, 1996), MIT Press, pp. 512–520.
- [37] GOLDBERG, D., DEB, K., AND KORB, B. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems* 3 (1989), 493–530.
- [38] GOLDBERG, D., AND RICHARDSON, J. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the 2nd International Conference on Genetic Algorithms* (1987), pp. 41–49.
- [39] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc., New York, NY, 1989.
- [40] HALL, L., OZYURT, B., AND BEZDEK, J. Clustering with a genetically optimized approach. *IEEE Trans. on Evolutionary Computation* 3, 2 (1999), 103–112.
- [41] HANCOCK, P. J. B. GANNET: Design of a neural net for face recognition by genetic algorithm. In *Proceedings of IEEE Workshop on Genetic Algorithms, Neural Networks and Simulated Annealing applied to problems in signal and image processing, Glasgow* (1990).
- [42] HARIK, G. Linkage learning via probabilistic modeling in the ECGA. Tech. Rep. IlliGAL Technical Report 99010, Illinois Genetic Algorithms Laboratory, Urbana, IL, 1999.
- [43] HARVEY, I. The SAGA cross: The mechanics of crossover for variable-length genetic algorithms. In *Parallel Problem Solving from Nature 2* (Cambridge, MA, 1992), Elsevier, pp. 269–278.

- [44] HARVEY, I. Species Adaptation Genetic Algorithms: A basis for a continuing SAGA. In *Proceedings of First European Conference on Artificial Life* (Cambridge, MA, 1992), MIT Press, pp. 346–354.
- [45] HOLLAND, J. H. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI, 1975.
- [46] HORNBY, G., LIPSON, H., AND POLLACK, J. Evolution of generative design systems for modular physical robots. In *2001 International Conference on Robotics and Automation* (2001).
- [47] HORNBY, G., AND POLLACK, J. The advantages of generative grammatical encodings for physical design. In *Proceedings of the Congress on Evolutionary Computation* (2001).
- [48] ISING, E. Beitrag zur theories des ferromagnetismus. *Z. Physik* 31, 235 (1924).
- [49] JAKIELA, M., CHAPMAN, C., DUDA, J., ADEWUYA, A., AND SAITOU, K. Continuum structural topology design with genetic algorithms. *Computer Methods in Applied Mechanics and Engineering* 186, 2–4 (2000), 339–356.
- [50] KAISE, N., AND FUJIMOTO, Y. Applying the evolutionary neural networks with genetic algorithms to control a rolling inverted pendulum. In *Proceedings of the Second Asia-Pacific Conference on Simulated Evolution and Learning* (1998).
- [51] KIRLEY, M., AND GREEN, D. An empirical investigation of optimisation in dynamic environments using the cellular genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 200)* (2000), Morgan Kaufmann, pp. 11–18.
- [52] KOZA, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.
- [53] KOZA, J., BENNETT III, F., ANDRE, D., AND KEANE, M. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann Publishers, 1999.
- [54] LEE, C.-Y., AND ANTONSSON, E. Dynamic partitional clustering using evolution strategies. In *Proceedings of the Third Asia Pacific Conference on Simulated Evolution and Learning* (2000).
- [55] LEE, C.-Y., AND ANTONSSON, E. Variable length genomes for evolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference* (Las Vegas, NV, 2000), Morgan Kaufmann, p. 806.
- [56] LEE, C.-Y., AND ANTONSSON, E. Adaptive evolvability via non-coding segment induced linkage. In *GECCO '01, Genetic and Evolutionary Computation Conference* (San Francisco, CA, 2001), Morgan Kaufmann, pp. 448–453.
- [57] LEE, C.-Y., AND ANTONSSON, E. Neural network design through embedded representations and evolutionary computation. In *2002 International Joint Conference on Neural Networks* (2002).

- [58] LEE, C.-Y., AND ANTONSSON, E. Reinforcement learning in steady-state cellular genetic algorithms. In *2002 Congress on Evolutionary Computation* (2002).
- [59] LEE, C.-Y., MA, L., AND ANTONSSON, E. Evolutionary and adaptive synthesis methods. In *Formal Engineering Design Synthesis* (Cambridge, UK, 2001), E. Antonsson and J. Cagan, Eds., Cambridge University Press.
- [60] LEVENICK, J. Inserting introns improves genetic algorithm success rate: Taking a cue from biology. In *Proceedings of Fourth International Conference on Genetic Algorithms* (1991), pp. 123–127.
- [61] LEVENICK, J. Metabits: Generic endogenous crossover control. In *Proceedings of Sixth International Conference on Genetic Algorithms* (1995), Morgan Kaufmann.
- [62] LI, H., AND ANTONSSON, E. K. Mask-layout synthesis through an evolutionary algorithm. In *MSM'99, Modeling and Simulation of Microsystems, Semiconductors, Sensors and Actuator* (Cambridge, MA, 1999), Applied Computational Research Society.
- [63] LINDENMAYER, A. Mathematical models for cellular interactions in development, parts i and ii. *Journal of Theoretical Biology* 18 (1968), 280–315.
- [64] LIPSON, H., AND POLLACK, J. Automatic design and manufacture of robotic life forms. *Nature* 406 (2000), 974–978.
- [65] LOBO, F., DEB, K., GOLDBERG, D., HARIK, G., AND WANG, L. Compressed introns in a linkage learning genetic algorithm. In *Proceedings of the Third Annual Conference on Genetic Programming* (1998), Morgan Kaufmann.
- [66] MA, L., AND ANTONSSON, E. K. Applying genetic algorithms to MEMS synthesis. In *ASME International Mechanical Engineering Congress and Exposition* (New York, NY, 2000), ASME.
- [67] MA, L., AND ANTONSSON, E. K. Mask-layout and process synthesis for MEMS. In *MSM'00, Modeling and Simulation of Microsystems, Semiconductors, Sensors and Actuator* (Cambridge, MA, 2000), Applied Computational Research Society.
- [68] MANDERICK, B., AND SPIESSENS, P. Fine grained parallel genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms* (1989), Morgan-Kaufmann, pp. 428–433.
- [69] MENDEL, G. Versuche uber pflanzen-hybriden. In *Verhandlungen des naturforschenden Vereines in Brunn* (1865), vol. 4.
- [70] MITCHELL, M., FORREST, S., AND HOLLAND, J. The royal road for genetic algorithms: Fitness landscapes and GA performance. In *Proceedings of the First European Conference on Artificial Life* (1992), pp. 245–254.

- [71] NEHANIV, C. Evolvability in biology, artifacts, and software systems. In *Evolvability Workshop at the Seventh International Conference on Simulation and Synthesis of Living Systems (Artificial Life 7)* (2000).
- [72] PEPPER, J. The evolution of modularity in genome architecture. In *Evolvability Workshop at the 7th International Conference on Simulation and Synthesis of Living Systems (Artificial Life 7)* (2000).
- [73] PETTEY, C., LEUZE, M., AND GREFENSTETTE, J. A parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms* (1987), pp. 155–161.
- [74] PRUGEL-BENNETT, A. Modelling crossover-induced linkage in genetic algorithms. *IEEE Transactions on Evolutionary Computation* (2001).
- [75] RECHENBERG, I. Cybernetic solution path of an experimental problem. Tech. Rep. Technical Report Library Translation No. 1122, Royal Aircraft Establishment, Farnborough, Hants, U.K., 1965.
- [76] RECHENBERG, I. *Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, Germany, 1973.
- [77] SCHLUTER, D. Experimental evidence that competition promotes divergence in adaptive radiation. *Science* 266 (1994), 798–801.
- [78] SCHURMANN, J. *Pattern Classification: A Unified View of Statistical and Neural Approaches*. John Wiley, New York, NY, 1996.
- [79] SCHWEFEL, H.-P. *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Stromungstechnik*. PhD thesis, Technische Universität, 1965.
- [80] SCHWEFEL, H.-P. *Evolution and Optimum Seeking*. John Wiley, New York, 1995.
- [81] SIMS, K. Artificial evolution for computer graphics. *Computer Graphics* 25, 4 (1991), 319–328.
- [82] SKOURIKHINE, A. An evolutionary algorithm for designing feedforward neural networks. In *Proceedings of the 7th International Evolutionary Programming Conference* (1998).
- [83] SMITH, J., AND FOGARTY, T. Recombination strategy adaptation via evolution of gene linkage. In *Evolvability Workshop at the 7th International Conference on Simulation and Synthesis of Living Systems (Artificial Life 7)* (2000).
- [84] SPEARS, W. Simple subpopulation schemes. In *Proceedings of the Evolutionary Programming Conference* (1994), World Scientific.
- [85] SYSWERDA, G. A study of reproduction in generational and steady-state genetic algorithms. In *Foundations of Genetic Algorithms* (1991), Morgan Kaufmann.
- [86] VANHOYWEGHEN, C., GOLDBERG, D., AND NAUDTS, B. Building block superiority, multimodality and synchronization problems. Tech. Rep. 2001020, Illinois Genetic Algorithms Laboratory, 2001.

- [87] WICKER, D., RIZKI, M., AND TAMBURINO, L. A hybrid evolutionary learning system for synthesizing neural network pattern recognition systems. In *Proceedings of the 7th International Evolutionary Programming Conference* (1998).
- [88] WOLPERT, D. H., AND MACREADY, W. G. No free lunch theorems for search. Tech. Rep. TR-95-02-010, The Santa Fe Institute, Santa Fe, NM, 1995.
- [89] WU, A., AND LINDSAY, R. Empirical studies of the genetic algorithm with non-coding segments. *Evolutionary Computation* 3, 2 (1995), 121–147.
- [90] WU, A., AND LINDSAY, R. A survey of intron research in genetics. In *Proceedings of the Fourth International Conference on Parallel Problem Solving from Nature* (1996).
- [91] WU, A., LINDSAY, R., AND SMITH, M. Studies on the effect of non-coding segments on the genetic algorithm. In *Proceedings of the Sixth IEEE International Conference on Tools with AI* (1994).
- [92] ZHAO, Q. A study on co-evolutionary learning of neural networks. In *Proceedings of the First Asia-Pacific Conference on Simulated Evolution and Learning* (1996).