

FUZZY EVOLUTIONARY CELLULAR AUTOMATA

by

James Neal Richter

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Computer Science

Approved:

Dr. Nicholas Flann
Major Professor

Dr. David Peak
Committee Member

Dr. Donald Cooley
Committee Member

Dr. Thomas Kent
Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2003

Copyright © James Neal Richter 2003

All Rights Reserved

ABSTRACT

Fuzzy Evolutionary Cellular Automata

by

James Neal Richter, Master of Science

Utah State University, 2003

Major Professor: Dr. Nicholas Flann
Department: Computer Science

An application of genetic algorithms in search of optimal cellular automata rules to solve the density classification task is presented. A review of recent work is detailed along with a study of the statistical significance of previous results. A review of powerful genetic algorithm enhancements is also presented, with the aim of demonstrating marked improvement in the robustness and optimization capability of the GA. These techniques are then applied to the evolutionary cellular automata model to show improvement in convergence speed and more effective search of the optimization landscape.

(98 pages)

ACKNOWLEDGMENTS

For my parents, Maureen and Jim & Francile, and my new bride, Stephanie.

And thanks to Mr. Fred Cunningham for showing me the joy of solving problems well.

Thanks to Dr. Nick Flann, Dr. David Peak, Dr. Don Cooley, Dr. John Paxton, and Dr. Heng Da Cheng and my colleagues at RightNow Technologies for their support and encouragement during this research.

James Neal Richter
June 2003

PREFACE

Research Note:

The research for Chapters 2 & 3 was conducted during 1998 & 1999. The experiments in Chapter 4 and appendices were conducted during 2000-2001. Each run of a particular high initial condition experiment took 20+ days on a Pentium 4 1.4Ghz machine.

Software:

StarOffice 5.2 & 6.0 were used to write the paper. GALib 2.4.5 was used for all genetic algorithm experiments. The GNU Scientific Library was used for random number algorithms and other uses. SciLab, Rlab, & Gnuplot were used for computations and graphing. Mirek Wójtowicz's Java Celebration is used to visualize the 1-D cellular automata rules.

CONTENTS

	Page
ABSTRACT	iii
ACKNOWLEDGMENTS	iv
PREFACE	v
LIST OF FIGURES	viii
CHAPTER	
1. INTRODUCTION	1
2. REVIEW OF ADAPTIVE GENETIC ALGORITHMS	2
1 Introduction	3
2 Analysis & Comparisons	4
2.1 Population Size	5
2.2 Crossover and Mutation	6
2.3 Incest Prohibition	9
2.4 Fuzzy Fitness Functions	10
2.5 Convergence Detection	11
2.6 Other Methods	12
2.7 Meta-Level Genetic Algorithms	13
2.8 Other Fuzzy Genetic Algorithm Surveys	14
3 Case Studies	14
3.1 Case Study A: Lee et al.	14
3.2 Case Study B: Herrera et al.	17
3.3 Case Study C: Shi et al.	20
4 Conclusions	21
References	22
3. REVIEW OF 1-D CELLULAR AUTOMATA AND DENSITY CLASSIFICATION	26
1 Cellular Automata	26
2 The Density Classification Problem	28
3 Conclusions	32

		vii
	References	33
4.	FUZZY EVOLUTIONARY CELLULAR AUTOMATA	34
	1 Introduction	34
	2 Evolutionary Cellular Automata	34
	3 Density Classification	36
	4 EvCA Algorithm	37
	5 Randomized Fitness Function Analysis	38
	6 Adaptive EvCA	40
	7 SEvCA and MEvCA	41
	8 Fuzzy Adaptive EvCA	41
	9 Conclusions and Future Work	44
	References	44
5.	CONCLUSIONS	46
	APPENDICES	47
A.	EXPERIMENT NOTES ON FEVCA, FUTURE DIRECTIONS, AND SOURCE CODE	48
	1 Experiment Notes on FEvCA, MEvCA, and FMEvCA	49
	1.1 DeJong Benchmarks	49
	1.2 DeJong Function 2	51
	1.3 DeJong Function 3	53
	1.4 DeJong Function 4	54
	1.5 Benchmark Observations	57
	2 Questions and ideas for improvement	57
	3 Future Directions	59
	4 Source Code	60
	References	80
B.	SECONDARY REFERENCES	81
C.	COPYRIGHT RELEASES	86
	ASME Press Copyright Release Letter	87
	Copyright Release Form from David Peak	88

LIST OF FIGURES

Figure		Page
2.1	Population size adapting rules from Lee and Takagi [6]	5
2.2	Crossover rate adapting rules from Xu and Vukovich [13]	7
2.3	Rules for crossover and mutation rate adaptation from Shi et al. [15]	8
2.4	Diagram of dynamic parametric genetic algorithm from Lee and Takagi [6]	15
2.5	Fuzzy rule set for crossover rate change from Lee and Takagi [16]	16
2.6	Fuzzy rule set for mutation rate change from Lee and Takagi [16]	16
2.7	Crossover operators from Herrera et al. [17]	18
2.8	Families of fuzzy connectives functions from Herrera et al. [17]	18
2.9	Offspring selection strategies from Herrera et al. [17]	19
3.1	1-D binary 2 neighbor cellular automata rule 90	27
3.2	Cellular automata rule 90 with a single cell ON in initial condition	27
3.3	Cellular automata rule 90 with a random cells ON in initial condition	28
3.4	GKL rule on a N=149 lattice with an initial condition of slightly $< 1/2$	29
3.5	GKL rule on a N=149 lattice with an initial condition of slightly $> 1/2$	29
3.6	The accuracy curve of the GKL rule with lattice size 149	30
3.7	Rule 184 on a N=149 lattice with an initial condition of slightly $> 1/2$	31
3.8	Rule 184 on a N=149 lattice with an initial condition of slightly $< 1/2$	31
4.1	1-D binary 2 neighbor cellular automata rule 90	35

		ix
4.2	Cellula automata rule 90 with a single cell ON in initial condition	36
4.3	GKL rule in action with a random initial condition	37
4.4	Initial condition significance test with scores in percentages	39
4.5	Fuzzy adaptive GA diagram	42
4.6	Rules for crossover and mutation rate adaptation from Shi et al. [16] ...	43
4.7	Performance of EvCA algorithms, fitness on Y-axis, generation on X-Axis	43
A.1	Summary of parameter settings for evolutionary cellular automata experiments	49
A.2	Stepping stone multi-population GA model	50
A.3	DeJong function 2	51
A.4	DeJong function 2 maximization performance	52
A.5	DeJong function 3	53
A.6	DeJong function 3 maximization performance	54
A.7	DeJong function 4	55
A.8	DeJong function 4 maximization performance	56
A.9	DeJong function 4 minimization performance	56

CHAPTER 1

INTRODUCTION

This thesis is an attempt to improve upon an existing approach to using the genetic algorithm to find cellular automata rules that perform computations from the very large search space of rules.

Chapter 2 is a survey of adaptive genetic algorithms, specifically those that use fuzzy logic controllers to implement the adaptation. A variety of research is summarized, and three models are looked at more closely.

Chapter 3 is a survey of the state of research in using cellular automata rules to perform density classification. Recent results and several proofs concerning density classification are presented.

Chapter 4 is a published paper that adds adaptive GAs to the evolutionary cellular automata system introduced by researchers at the Santa Fe Institute. A number of problems with the existing approach are discussed and extensions are proposed.

The Appendix contains an expanded set of experiment notes based on the experiments for Chapter 4. It also contains computer source code and a number of ideas to improve the system and answer some outstanding questions.

CHAPTER 2

REVIEW OF ADAPTIVE GENETIC ALGORITHMS

Abstract

Genetic algorithms are powerful tools that allow engineers and scientists to find good solutions to hard computational problems using evolutionary principles. The classic genetic algorithm has several disadvantages, however. Foremost among these drawbacks is the difficulty of choosing optimal parameter settings. Genetic algorithm literature is full of empirical tricks, techniques, and rules of thumb that enable GAs to be optimized to perform better in some way by altering the GA parameters. Capturing these rules of thumb in fuzzy logic systems and using such systems to dynamically control the parameters of GAs has enabled researchers to create GA systems that outperform conventional GAs. This chapter is a survey of basic parameter adaptation and different ways researchers have integrated fuzzy logic systems into GAs.

1 Introduction

The genetic algorithm (GA) is a powerful tool that allows us to find solutions to problems that are poorly understood, have many interdependencies, or are otherwise too complex to solve directly. Modeled after natural selection and the evolutionary process, the basic setup is a population of possible solutions that is analyzed for fitness and recombined to form the next generation. The process is repeated until a desirable solution is found [1].

The genetic algorithm (GA) can be configured in many ways, and these different setups can have strong effects on the solutions found. Fitness functions, crossover

operators, mutation operators, selection operators, and population size are just a few of the many parameters that are available to be optimized.

The GA literature is crowded with papers on various techniques researchers have found to set up these parameters for various problem domains. Other researchers have found interdependencies between operators and formulated helpful heuristics to follow when designing a GA system for a specific domain [1].

Many researchers agree that the classic GA has serious flaws. It can be a robust method, although when set-up improperly that property diminishes. Chief among those pitfalls is the inability of the classic GA to adapt to the changing characteristics of the solution space as the population moves around in it.

A second flaw is that the GA will often give us little or no insight into the nature of the problem space. The result is that the GA is unable to explicitly learn information about the problem space.

Third, most classic GA implementations suffer from a lack of persistence. Once a generation has completed the fitness-selection-crossover cycle, it is thrown out, thus losing any information not explicitly passed on to the next generation.

Finally, premature convergence of the population into local minima is another common problem. Afterward, it can be difficult for the populations to get out of the minima [1, 2, 3].

The pragmatic result of these issues is that the GA needs to be closely managed by people familiar with the patterns and pitfalls of the GA and the problem domain [4, 5].

Fuzzy logic (FL) is a mathematical construct allowing representation of knowledge in imprecise and nonspecific ways. This enables FL to be used to reason on knowledge that is not clearly defined or completely understood [6].

Fuzzy logic has allowed a small group of researchers to devise ways of optimizing the performance and solution quality of the GA. FL is used to incorporate the many heuristic tricks and techniques of experienced GA researchers into an FL system to control the setup of the GA [5, 6]. The goal of such systems is generally to speed up the convergence of the GA and/or obtain better quality solutions [5, 6].

Fuzzy logic systems can also be used to give good performance estimates of running GAs, resulting in a feedback mechanism enabling further enhancement of performance [7].

The goal of this chapter is to give the reader a survey of the current work done in this area and an analysis of the techniques used. Many of the techniques presented here rely on the use of fuzzy logic controllers (FLCs). Bonissone and Chaing [8] is an excellent introductory paper on FLCs. They cover the basics of fuzzy logic, including algorithms and membership functions, and compare three different types of FLCs.

2 Analysis & Comparisons

Hinterding et al. [9] is a survey paper of general GA parameter adaptation. They classify adaptation into four types and four levels. The types are:

- static (unchanging parameters)
- deterministic dynamic (parameters changed with a deterministic function)
- dynamic adaptive (parameters changed with feedback)
- dynamic self-adaptive (adaptation method encoded into chromosome).

The levels are:

- environmental (changing fitness function response to individuals with a heuristic)

- population (any parameters affecting entire population)
- individual (mutation rate, age of individuals)
- component (varying parameters for individual genes).

2.1 Population Size

Population sizing is a critical parameter for GA users. When the population is undersized, the GA may converge in too few generations to ensure a good solution. When the population is oversized, the GA is inefficient as it produces and evaluates more individuals than necessary to find an optimal result.

Goldberg [10] uses the variance of fitness to give the first formula for estimating population size. Harik et al. [11] build on that and present a formula for determining a population size for a desired convergence quality. The genome is thought of as composed of a number of smaller Building Blocks (BBs) and Gambler's Ruin model is applied to model the number of correct BBs as a random walk between correct and incorrect convergence.

Figure 2.1 shows rules from Lee and Takagi [6]. A small fuzzy if-then rule-set to govern population size based on population fitness metrics.

IF AF / BF is big	THEN increase PS
IF WF / AF is small	THEN decrease PS
IF MR is small AND PS is small	THEN increase PS

Figure 2.1: Population size adapting rules from Lee and Takagi [6].
 AF = Average Fitness, BF = Best Fitness, WF = Worst Fitness, MR = Mutation Rate,
 PS = Population Size

2.2 Crossover and Mutation

In GAs and other evolutionary algorithms (EAs), crossover denotes the process of recombining the genetic chromosomes of pairs of individuals in the population to form a new generation. Mutation refers to a stage in offspring production immediately following crossover, where the resulting gene is randomly mutated.

The settings of crossover and mutation operators during a GA run can have a definite effect on the results. Crossover is a process of local refinement of population members, often referred to as an exploiter of the problem space. Mutation is referred to as an explorer of the space. Mutation also has the effect of redistributing the population in the space. The balance between the two parameters is crucial to the success of the population [12]. Even these designations don't always hold true, as low mutations rates can be effective exploiters of the space, and crossover can often effectively explore the space if combined with an appropriate selection scheme.

As a general heuristic, a low mutation rate and high crossover rate is preferred when the population fitness is low and the population diversity is low. Conversely, when the opposite situation is in effect a high mutation-rate and low crossover-rate are preferred. The dependencies between crossover and mutation are largely a function of the problem domain and may not be precisely known [4].

Xu and Vukovich [13] outline a simple way to adapt crossover rate, mutation rate, and population size based on a fuzzy rule set dependent on the generation count. Figure 2.2 shows the rule with input of population size on the top and generation count on the left.

Arnone et al. [14] present a simple system to modify mutation rate based on a population diversity metric. The authors first present a set of statistical population

	<i>Population Size</i>		
<i>Generation</i>	<i>Small</i>	<i>Medium</i>	<i>Large</i>
<i>Low</i>	medium	small	small
<i>Medium</i>	large	large	large
<i>High</i>	very large	very large	very large

Figure 2.2: Crossover rate adapting rules from Xu and Vukovich [13]

metrics and produce a diversity function from those metrics.

A trapezoidal membership function is used to classify the diversity as either convergent or non-convergent. When a convergence state is reached the mutation rate is set to 0.6, otherwise it is set to 0.05.

In [15] the authors introduce a fuzzy if-then rule-set and corresponding membership functions to modify crossover and mutation rates based on five metrics. A complete definition of the fuzzy system is also given, defining input membership functions for the five metrics, and output membership functions for the high medium and low states. Figure 2.3 shows the rules for the fuzzy system of Shi et al. [15].

Wang et al. [2] show a system of two separate fuzzy controllers that adaptively adjust crossover and mutation rates. The crossover controller takes two inputs, population fitness at the current generation and fitness at the previous generation, and output the change in crossover rate. In the mutation rate controller, the inputs are the same as in the crossover controller, with the output being the change in mutation rate.

Nine evenly distributed membership functions with 25% neighbor overlap are applied to each input to fuzzify the input. The two classified values are then used as inputs to a 9x9 table. The resulting entry is defuzzified using the same nine membership functions. The output is then added to the current crossover or mutation rate for the next

IF BF is low	THEN MR is low and CR is high
IF BF is medium and UN is low	THEN MR is low and CR is high
IF BF is medium and UN is medium	THEN MR is medium and CR is medium
IF BF is high and UN is low	THEN MR is low and CR is high
IF BF is high and UN is medium	THEN MR is medium and CR is medium
IF UN is high and VF is medium	THEN MR is high and CR is low
IF UN is high and VF is low	THEN MR is high and CR is low
IF UN is high and VF is high	THEN MR is low and CR is low

Figure 2.3: Rules for crossover and mutation rate adaptation from Shi et al. [15]. BF = Best Fitness, UN = Number of generations since last BF change, VF = Variance of Fitness, MR = Mutation Rate, CR = Crossover Rate

generation.

Lee and Takagi [16] present extensions to earlier work [6] modifying crossover and mutation rates as well as population size. The system is implemented as a fuzzy controller with four population metrics as inputs (diversity, gravity, rank, and volume) and three outputs (crossover rate, mutation rate, and population size). A case study of this system and additional extensions are presented later in Case Study A.

In Herrera et al. [4], four distinct crossover operators are introduced along with a fuzzy controller to adjust crossover and mutation parameters. Population diversity metrics are reviewed and used as inputs to the fuzzy controller.

A fixed length generation size is divided into three stages. The first stage makes up 10% of the total generation time and all diversity measures are ignored. In the second stage, making up half of the total generation time, the crossover and mutation probabilities are varied according to a set of fuzzy rules. The goals of this stage are

ensuring good population diversity and preserving a good balance of exploration vs. exploitation.

The final stage makes up the remaining 40% of the total generation time. The goal is to subdue exploration and concentrate on exploitation and refinement of the population.

Two diversity measures, average genetic variance and average variance of selected alleles (gene groups), are used as inputs to the system. No details are given on the membership functions or defuzzification methods. The outputs are new mutation and crossover rates.

Herrera et al. continue to expand the system in other papers [17, 18]. Details are presented in Case Study B.

2.3 Incest Prohibition

Craighurst and Martin [19] present a very interesting study on ancestral based crossover prohibitions. The main idea is to prevent incest in populations during selection and crossover. The level of prohibition starts at zero (no prohibition) and continues at one (no asexual or sibling crossover) and two (no mating with parents), etc.

They surmise that incest prohibition can increase population diversity and have a positive effect on the solution quality and convergence speed of a GA. The ancestry information is passed along in a section of an individual's genome where crossover is not performed. Each unique solution is assigned an ID number and stored in the gene, depending on the specified incest prohibition level.

A caveat exists: the population size must be big enough to support the level of prohibition. If not, the number of eligible mates may not be high enough to support good diversity.

While not technically a fuzzy method, the idea has a lot of merit and could be added to a fuzzy GA system. A set number of ancestors could be kept for each individual and prohibition values could be varied with respect to population diversity measures. Descriptions of other approaches to diversity maintenance can be found in Mitchell [1].

2.4 Fuzzy Fitness Functions

Fitness functions are the engines of the GA. They decide which individuals mate and which expire. Constructing a correct fitness function will make or break the GA's ability to find accurate solutions.

Ankenbrandt et al. [20] implement a system of fuzzy fitness functions to grade the quality of chromosomes representing a semantic net. The system is used to assist in recognizing oceanic features from partially processed satellite images.

The quality of the semantic net is established by its satisfaction of a set of fuzzy predicate requirements. The sum of all satisfied predicates is the output of the fuzzy fitness function. An example predicate for this domain is the 'is far' relationship between two categories of oceanic features, which is known in general terms by the authors.

In Nishio et al. [21], a system where a user acts as a fuzzy fitness function is described. The user is presented with a collection of cartoon faces rendered by a chromosome of common facial feature variations. After rating the faces with respect to the target face, the GA proceeded normally with recombination & mutation.

One problem that surfaced was the fall off of concentration when users are required to rate every face presented to them. As an alternative, the user selects the closest matching face to the target face. This winner receives a perfect fitness score and the set of losers are assigned a bias value of less than perfect.

A refinement was then presented where the losers are rated according to a fuzzy function. Input to the fitness function are values representing distance and direction of the face from the average (or origin) face and the generation count. Although this is clever, the enhancement resulted in minimal improvements to convergence rate as compared to straight biasing.

In [22], researchers at Rolls-Royce presented a study of the use of fuzzy systems to characterize engineering judgment and its use with the GA. They demonstrate an industrial design application where a system of problem-specific engineering heuristics and hard requirements are combined to form a fitness function for the GA. The result was a gain in productivity of the engineer and quality of the solution.

The fitness function contains membership functions capturing the suitability of a range of frictional coefficients and heat-exchange flow mismatches in gas turbine engines. Based on these and other inputs, a set of fuzzy rules is applied to the fuzzified inputs. The results are defuzzified via a weighted average centroid method.

Lee and Takagi [6, 16] present the use of the DeJong [23] functions to grade the population and apply the result as feedback to the fuzzy GA control system. This is detailed further in Case Study A.

2.5 Convergence Detection

Meyer and Feng [7] concentrate on determining when to stop the GA. First, the optimal performance level is estimated using past population fitness measures and a least-mean-squares function. Next, the user defines a desired performance as a percentage of the optimal performance.

The algorithm also calculates a belief and uncertainty measure of the estimated

optimal performance level. The belief measure is based on a subjective fuzzy combination of six functions. The uncertainty measure is based on an objective relative error calculation. These calculations are not necessarily run after each generation. Using these three inputs, the algorithm determines when to stop the GA.

They claim the algorithm terminates more quickly than a reasonable fixed iteration GA. However, the exact details are vague with respect to how fuzzy logic is used, when and how the measures are calculated, and how the algorithm avoids false positive results.

2.6 Other Methods

Voigt et al. [24] present a multi-valued GA inspired by fuzzy logic. The GA's gene representation is multi-valued and all standard operators are adapted to handle this. Here, multi-valued refers to a non-binary gene value. No dynamic adjustment of parameters is used.

Voigt et al. [25] build on previous work and discuss a fuzzy crossover operator inspired by fuzzy logic. The crossover probability is governed by a bimodal distribution curve. This membership-curve results from two overlapping triangular distribution curves. Fundamentally, this is equivalent to a simple fuzzy system.

This method is applied to the Breeder GA, and the authors show linear convergence against a test suite of BGA benchmark functions.

Buckley and Hayashi [26, 27] demonstrate a method different from the others presented here. Although a standard GA is used, the difference comes in that the chromosome is a fuzzy set.

The goal is to maximize a resultant fuzzy set and a fuzzy mapping function. Their

GA uses fixed parameters with no variation. The method is applied to four applications, fuzzy linear programming, the fuzzy maximum flow problem, fuzzy regression, and fuzzy controller tuning.

Seront and Bersini [28] review two protocols for combining the GA with secondary methods to increase search performance. The first protocol involves letting the GA generate populations, then applying a hill climbing operator on each individual and finishing with a fitness and selection stage. The exact details on how the hill climbing operator optimizes the individuals are not discussed.

The second protocol describes a method of modifying the traditional GA operators (selection, mutation, crossover) to function more like classical optimization methods. They specifically discuss simplex method hybridization for real valued functions.

Next, a brief treatment of a combination of the above two hybridization methods is presented. One interesting element of this would be to probabilistically choose between different crossover methods. This type of hybridization could easily be extended by a fuzzy logic system that better controlled when alternate optimization and crossover methods are selected.

2.7 Meta-Level Genetic Algorithms

Lee and Esbensen [29] present an extension to their fuzzy GA [6]. For some problem domains, good rule of thumb knowledge about GA parameter variation is not known. They present a meta-level GA to optimize the fuzzy system, which dynamically adjusts GA parameters.

This allows researchers to automatically construct good fuzzy GA rule sets. They apply this technique specifically to multi-objective GAs, where more than one fitness

parameter is to be optimized. Full details of this system are presented in Case Study A.

2.8 Other Fuzzy Genetic Algorithm Surveys

Herrera and Lozano [3] wrote a fuzzy GA survey paper. The first half of the paper presents previous work starting with non-fuzzy adaptive methods and finishing with a thorough treatment of FLC based fuzzy GA implementations and diversity measures. The second half of the paper reviews the authors' previous work. A brief is presented here in Case Study B.

Cordon et al. [30] is a brief review and bibliography on the combination of fuzzy logic and evolutionary computation. The paper touches on fuzzy GAs, fuzzy clustering, optimization, neural networks, expert systems, fuzzy classification, genetic fuzzy systems, and a few other areas. Each topic area has a short bibliography.

3 Case Studies

3.1 Case Study A: Lee et al.

Lee and Takagi [31] show a simple scheme for optimizing fuzzy systems with genetic algorithms. They use a static GA to evolve a fuzzy rule set with a goal of performing well on the inverted pendulum task. This research is used later in meta-level GA methods.

Lee and Takagi [6] is an early and frequently referenced paper on fuzzy GAs. The Dynamic Parametric GA (DPGA) is described here as an evolutionary system optimized by a fuzzy knowledge-base applied to control population size. In comparison to a static GA for solving the inverted pendulum problem, the fuzzy GA outperforms the static GA in online and offline performance measurements.

Online and offline performance measures are derived from the DeJong function test suite [23]. Online measures gauge ongoing real-time performance, and offline measures are used to estimate convergence.

They then go on to detail a more extensive framework for studying the effects of FLC controlled crossover, mutation, and population size [16]. This expanded study first presents a comparison of static, online and offline optimized GAs. A system diagram is given in Figure 2.4, and a full listing of all membership functions and rule sets is given in Figures 2.5 and 2.6.

Next the paper presents a comparison of fuzzy GAs with some of the three parameters held static. As predicted, the fully dynamic GA strategy out performed all other combinations.

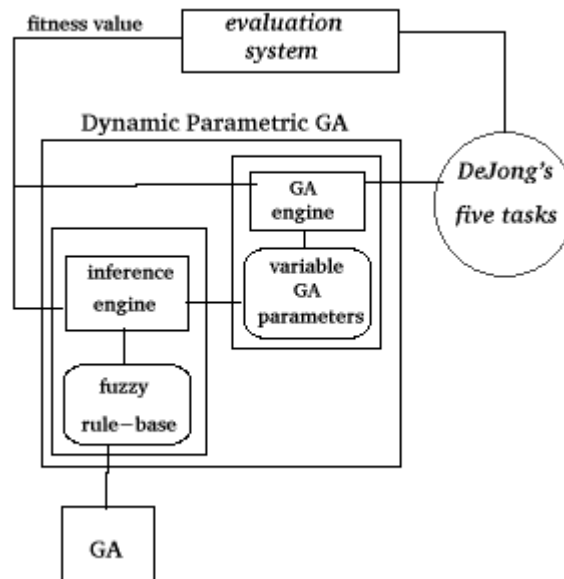


Figure 2.4: Diagram of dynamic parametric genetic algorithm from Lee and Takagi [6]. The DPGA system is composed of a fuzzy rule-base and inference engine taking input metrics from the evaluation system and adapting the GA parameters for the next generation. The Meta-GA is pictured at the bottom right and used in [29] to tune the fuzzy rule base. In this case the fitness function is one of DeJong's five optimization functions from [23].

Ave/Best	Worst/Ave	ΔBest	ΔCrossover
Small	Small	Small	Positive
Medium	Small	Small	Positive
Big	Small	Small	Positive
Small	Medium	Small	Negative
Medium	Medium	Small	None
Medium	Big	Small	Negative
Small	Small	Medium	None
Small	Medium	Medium	Negative
Medium	Medium	Medium	Negative
Big	Medium	Medium	Negative
Small	Big	Medium	None
Medium	Big	Medium	None
Big	Big	Medium	None
Small	Small	Big	Positive
Medium	Small	Big	Negative
Medium	Big	Big	Positive
Big	Big	Big	Positive

Figure 2.5: Fuzzy rule set for crossover rate change from Lee and Takagi [16].

Ave/Best	Worst/Ave	ΔBest	ΔMutation
Small	Small	Small	Negative
Big	Small	Small	None
Medium	Medium	Small	Negative
Small	Big	Small	Negative
Medium	Big	Small	Negative
Big	Big	Small	None
Small	Small	Medium	Positive
Medium	Small	Medium	None
Big	Small	Medium	Positive
Medium	Medium	Medium	Negative
Big	Medium	Medium	Negative
Small	Big	Medium	Positive
Medium	Big	Medium	None
Big	Big	Medium	None
Small	Medium	Big	Negative
Small	Small	Big	None
Medium	Small	Big	Positive
Medium	Medium	Big	None
Medium	Big	Big	Negative
Big	Big	Big	None

Figure 2.6: Fuzzy rule set for mutation rate change from Lee and Takagi [16].

With [29] and [32], Lee joins up with Esbensen and Lemaitre to extend the work. The first extension is to add the meta-GA to tune the fuzzy adaption rules [29]. In [32], DPGAs are enhanced to handle more than one optimization measure. This allows the fuzzy GA to meet several goals at once. The online and offline performance measures are combined into a hybrid function and used as feedback into the meta-level GA.

In a subsequent final paper [29], they add a user interaction module. At any point during the GA run, the user can redefine the characteristics of a good solution, switch into a simple hill-climbing mode, and individually control GA parameters.

This method was used to aid hardware engineers in Integrated Circuit (IC) placement problems. Typically, the hardware engineer is trying to meet more than one goal, such as total area, aspect ratio, and minimizing the total compute time for layout processing. The results show the system has good ability to find solutions that fulfill the different goals well.

As a side note, the authors do point out that using the meta-level GA results in a large increase in compute time. This factor may be irrelevant since it is run infrequently. It would not be necessary to run meta-GA more than once per domain. No effort is put into detecting convergence (via offline measures) and stopping the algorithm in these papers.

3.2 Case Study B: Herrera et al.

In Herrera et al. [17], they reintroduce, from [5], four fuzzy logic based crossover operators. Each is tailored to exploration or exploitation, and they are loosely defined as maximizing, minimizing, averaging, and meta-averaging.

Four methods of concurrently applying the operators to a pair of individuals are

also introduced. Each method is ranked by its diversity promotion characteristics. Each method combines three or more crossover operators concurrently and selects some number of offspring from the lot produced by the different crossover operators.

In Figures 2.7, 2.8 and 2.9, each gene in the genome is defined as an element of the range $[a,b]$. C and C' are the values of a single gene from each parent. S and S' are the C and C' values scaled to the range $[0,1]$. The 'Family' column denotes four different families of fuzzy connectives used to construct the four crossover operators. The families are compared against each other.

The paper then goes on to discuss dynamic variation of the overall crossover strategy during the GA run. The goal is to promote exploration in the early stages and exploitation in the later stages. They define several ways to accomplish this, all involve

$$\begin{aligned} \text{If } C, C' \in [a,b] \\ F(C, C') &= a + (b-a)T(S, S') \\ S(C, C') &= a + (b-a)G(S, S') \\ M(C, C') &= a + (b-a)P(S, S') \\ L(C, C') &= a + (b-a)C(S, S') \end{aligned}$$

Figure 2.7: Crossover operators from Herrera et al. [17].

Family	T-Norm, $F(c,c')$	T-conform, $S(c,c')$	Averaging Op. $M(c,c')$ ($0 \leq \lambda \leq 1$)	Gen. Comp. Op. $L(c,c')$
Logical	$T1(x,y) = \min(x,y)$	$G1(x,y) = \max(x,y)$	$P1(x,y) = (1-\lambda)x + \lambda y$	$C1(x,y) = T1^{(1-\lambda)} * G1^{\lambda}$
Hamacher	$T2(x,y) = xy / (x+y-xy)$	$G2(x,y) = (x+y-2xy) / (1-xy)$	$P2(x,y) = 1 / ((y-\lambda xy + \lambda) / (xy) + 1)$	$C2(x,y) = P2(T2, G2)$
Algebraic	$T3(x,y) = xy$	$G3(x,y) = x+y-xy$	$P3(x,y) = x^{(1-\lambda)} * y^{\lambda}$	$C3(x,y) = P3(T3, G3)$
Einstein	$T4(x,y) = xy / (1+(1-x)(1-y))$	$G4(x,y) = (x+y) / (1+xy)$	$P4(x,y) = 2 / (1 + ((2-x)/x)^{(1-\lambda)} * ((2-y)/y)^{\lambda})$	$C4(x,y) = P4(T4, G4)$

Figure 2.8: Families of fuzzy connectives functions from Herrera et al. [17].

Strategy	FCB-Crossovers	Chromosomes to be crossed	OSM	Diversity
ST1	F,S and M-crossover	$(2/3) \cdot p_c \cdot N$	All three offspring	strong
ST2	F,S and two M-crossover	$p_c \cdot N$	The two most promising	weak
ST3	F,S,M and L-crossover	$p_c \cdot N$	The two most promising	weak
ST4	F,S,M and L-crossover	$(1/2) \cdot p_c \cdot N$	All four offspring	high

Figure 2.9: Offspring selection strategies from Herrera et al. [17].

multiplying each of the four operators by a function that reinforces or suppresses the result based on the estimated total run time.

Additionally, a brief discussion of another heuristics based strategy is presented. All of the various strategies employed used a non-uniform mutation operator defined in [33]. Finally, all of the strategies are compared on a set of six functions to minimize. The results show that the methods favoring weak population diversity performed the best.

In [12], they build upon earlier research by implementing a FLC to control crossover and the selection/fitness parameter. The FLC chooses between two crossover methods. One is based on the desire to explore or exploit the search space and the other modifies the fitness parameter based on the desire to alter population diversity. Two simple rules sets and a straight forward defuzzification method govern the FLC.

In [18], they reintroduce the four crossover operators from [5, 12, 17] and add four variations for each of the four original classes of operators. The sixteen total operators are divided into four families and ranked according to diversity enhancement and promotion of exploration vs. exploitation.

Four mixed crossover operator strategies are also introduced. Each strategy is then paired with one of the four crossover operator families, resulting in sixteen different

strategy/family pairs. Each strategy applies at least three of the four operators to produce two to four offspring. The offspring are added to the population according to a unique set of rules for each strategy.

The sixteen strategy/family pairs and five other well studied crossover techniques are run on a suite of four previously published optimization functions. The results are mixed but two of the strategies emerge as clear winners, each favoring weak population diversity and allowing plenty of exploitation during the GA run.

3.3 Case Study C: Shi et al.

Shi et al. [15] discuss an application of fuzzy adaptive GAs to evolve a fuzzy classification system for the iris dataset. The paper begins with an great overview of necessary background material concerning fuzzy expert systems, GAs, fuzzy logic, evolution of fuzzy systems, and fuzzy adaptive GAs.

Good detail is given for all aspects of the experiments, including example fuzzy systems, genome encoding methods for fuzzy systems, and constructing the fitness function. The four experiments conducted are:

1. Evolve fuzzy rules with fixed membership functions using a parameter static GA
2. Evolve fuzzy rules with fixed membership functions using a parameter adaptive GA
3. Evolve fuzzy rules including membership functions using a parameter static GA
4. Evolve fuzzy rules including membership functions using a parameter adaptive GA

The fuzzy expert system used to adapt GA parameters is fully detailed. See Figure 2.3 for the set of adaptation rules from the system. The results showed that the systems with an adaptive GA performed better and that evolving the membership functions of the target fuzzy classifier was also superior to fixed membership functions.

4 Conclusions

While GA parameter adaptation in general is fairly well studied, research on using fuzzy systems to implement known adaptation techniques is limited. The size of the adaptive GA research community is comparatively small, and unfortunately, some of the better researchers have moved on to other areas of study.

General GA parameter adaptation has proven to have many benefits over parameter static GAs. The idea of using fuzzy systems to manage the adaptation of GA parameters is a powerful one and has the potential of giving all GA researchers better tools to work with. For adaptive GAs to be more commonly used, it is essential to present a number of different systems within a common framework for direct comparison.

The basic techniques developed fall into two categories: big wins and small wins. The small wins are characterized by the dynamic control of population size, mutation rate, and crossover rate. These techniques usually result in minor improvements to the performance of the GA, but don't change the overall measure of the GA's robustness.

The big wins are the fuzzy analysis of population diversity, performance estimates, crossover operators, mutation operators, and fitness functions. These techniques can result in dramatic gains in GA performance and make the GA more robust with respect to finding quality solutions.

The most exciting research is from methods using feedback techniques to optimize and improve the fuzzy logic systems dynamically. These ideas allow the growth of our understanding of the process and problem spaces. All additions and improvements to the fuzzy system are easily understandable by other researchers.

The GA pitfalls, inability to adapt with respect to the problem space, lack of

insight into the problem space, and convergence detection, are all addressed with the methods presented here. Future research directions could confront the lack of a complete method incorporating all of the techniques presented here. The addition of an inclusive method will make an evolutionary system more complex, but could also result in performance and robustness gains.

An experimental survey comparing the many rules sets in addition to formulating many ad hoc adaptation architectures into fuzzy systems for comparison would be an excellent next step.

References

- [1] M. Mitchell, *An Introduction to Genetic Algorithms (Complex Adaptive Systems Series)*. Cambridge, MA: MIT Press, 1996.
- [2] P.Y. Wang, G.S. Wang, Y.H. Song, and A.T. Johns, "Fuzzy logic controlled genetic algorithms," in *Proc. of the 1996 IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'96)*, 1996, pp. 972-979.
- [3] F. Herrera and M. Lozano, "Adaptation of genetic algorithm parameters based on fuzzy logic controllers," in *Genetic Algorithms and Soft Computing* (F. Herrera and J. L. Verdegay, Eds.) New York: Physica-Verlag, 1996, pp. 95-125.
- [4] H.Y. Xu and G. Vukovich, "A fuzzy genetic algorithm with effective search and optimization," in *Proc. of the 1993 Int. Joint Conf. on Neural Networks (IJCNN'93)*, 1993, pp. 2967-2970.
- [5] F. Herrera, E. Herrera-Viedma, A. Lozano, and J.L. Verdegay, "Fuzzy tools to improve genetic algorithms," in *Proc. of the Second European Congress on Fuzzy and Intelligent Technologies (EUFIT'94)*, 1994, pp. 1532-1539.
- [6] M. Lee and H. Takagi, "Dynamic control of genetic algorithms using fuzzy logic techniques," in *Proc. of the Fifth Int. Conf. on Genetic Algorithms (ICGA'93)*, 1993, pp. 76-83.
- [7] L. Meyer and X. Feng, "A fuzzy stop criterion for genetic algorithms using performance estimation," in *Proc. of the 1994 IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'94)*, 1994, pp. 1990-1995.
- [8] P.P. Bonissone and K.H. Chiang, "Fuzzy logic controllers: from development to

- deployment,” in *Proc. of the 1993 IEEE Int. Conf. on Neural Networks*, 1993, pp. 610-619.
- [9] R Hinterding, Z. Michalewicz, and A.E. Eiben, “Adaption in evolutionary computation: a survey,” in *Proc. of 1997 IEEE Conf. on Evolutionary Computation (ICEC'97)*, 1997, pp. 65-69.
- [10] D.E. Goldberg and M. Rudnick, “Genetic algorithms and the variance of fitness,” *Complex Systems*, vol. 5, no. 3, pp. 265-278, 1991.
- [11] G. Harik, E. Cantu-Paz, D.E. Goldberg, and B.L. Miller, “The gambler's ruin problem, genetic algorithms, and the sizing of populations,” in *Proc. of the 1997 IEEE Conf. on Evolutionary Computation (ICEC'97)*, 1997, pp. 7-12.
- [12] F. Herrera and M. Lozano, “Adaptive genetic algorithms based on fuzzy techniques,” in *Proc. of the Sixth Int. Conf. on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU)*, 1996, pp. 775-780.
- [13] H.Y. Xu and G. Vukovich, “A fuzzy genetic algorithm with effective search and optimization,” in *Proc. of the 1993 Int. Joint Conf. on Neural Networks (IJCNN'93)*, 1993, pp. 2967-2970.
- [14] S. Arnone, M. Dell'Orto, A. Tettamanzi, and M. Tomassini, “Towards a fuzzy government of genetic algorithms,” in *Proc. of the Sixth IEEE Conf. on Tools with Artificial Intelligence 1994 (TAI'94)*, 1994, pp. 585-591.
- [15] Y. Shi, R. Eberhart, and Y. Chen, “Implementation of evolutionary fuzzy systems,” *IEEE Transactions on Fuzzy Systems*, vol. 7, no. 2, pp. 109-119, 1999.
- [16] M. Lee and H. Takagi, “A framework for studying the effects of dynamic crossover, mutation, and population sizing in genetic algorithms,” in *1995 Special Issue of IEEE World Wiseperson Workshop on Fuzzy Logic and Neural Nets/Genetic Algorithms*, (T. Furuhashi Ed.) New York: Springer-Verlag, 1995, pp. 111-126.
- [17] F. Herrera, M. Lozano, and J.L. Verdegay, “Dynamic and heuristic fuzzy connectives-based crossover operators for controlling the diversity and convergence of real coded genetic algorithms,” *Int. Journal of Intelligent Systems*, vol. 11, no. 12, pp. 1013-1041, 1996.
- [18] F. Herrera, M. Lozano, and J.L. Verdegay. “Fuzzy connectives based crossover operators to model genetic algorithms population diversity,” *Fuzzy Sets and Systems*, vol. 92, no. 1, pp. 21-30, 1997.
- [19] R. Craighurst and W. Martin, “Enhancing GA performance through crossover prohibitions based on ancestry,” in *Proc. of the Sixth Int. Conf. on Genetic*

Algorithms (ICGA'93), 1994, pp. 130-135.

- [20] C.A. Ankenbrandt, BP Buckles, F.E. Petry, and M. Lybanon, "Ocean feature recognition using genetic algorithms with fuzzy fitness functions," in *Proc of the Third Annual Workshop on Space Operations, Automation and Robotics (SOAR'89)*, 1989, pp. 679-685.
- [21] K. Nishio, M. Murakami, E. Mizutani, and N. Honda. "Fuzzy fitness assignment in an interactive genetic algorithm for a cartoon face search," in *Genetic Algorithms and Fuzzy Logic Systems. Soft Computing Perspectives*, (E. Sanchez, T. Shibata, and L. A. Zadeh Eds.), River Edge, NJ: World Scientific, 1997, pp. 175-191.
- [22] R. Pearce and P.H. Cowley, "Use of fuzzy logic to describe constraints derived from engineering judgment in genetic algorithms," *IEEE Transactions on Industrial Electronics*, vol. 43, no. 5, pp. 535-540, 1996.
- [23] K. A. DeJong, "An analysis of the behavior of a class of genetic adaptive systems," *Ph.D Dissertation, University of Michigan*, University Microfilms No. 68-7556. 1975.
- [24] H.-M. Voigt, J. Born, and I. Santibanez-Koref, "A multivalued evolutionary algorithm," *International Computer Science Institute (ICSI)*, Berkeley, CA, Technical Report TR-93-022, 1993.
- [25] H.-M. Voigt and H. Muhlenbein, and D. Cvetkovic, "Fuzzy recombination for the breeder genetic algorithm," in *Proc. of the Sixth Int. Conf. on Genetic Algorithms (ICGA'93)*, 1995, pp. 15-19.
- [26] J.J Buckley and Y Hayashi, "Fuzzy genetic algorithms for optimization," in *Proc. of the 1993 Int. Joint Conf. on Neural Networks (IJCNN'93)*, 1993, pp. 725-728.
- [27] J.J Buckley and Y Hayashi, "Fuzzy genetic algorithm and applications," *Fuzzy Sets and Systems*, vol. 61, no. 2, pp. 129-136, 1994.
- [28] G. Seront and H. Bersini, "Simplex GA and hybrid methods," in *Proc. of the 1996 IEEE Conf. on Evolutionary Computation (ICEC'96)*, 1996, pp. 845-848.
- [29] M. Lee and H. Esbensen, "Automatic construction of fuzzy controllers for evolutionary multiobjective optimization algorithms," in *Proc. of the IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'96)*, 1996, pp. 1518-1523.
- [30] O. Cordón, F. Herrera, and M. Lozano, "On the combination of fuzzy logic and evolutionary computation: A short review and bibliography," *Fuzzy Evolutionary Computation*, (W. Pedrycz Ed.), New York: Kluwer Academic, 1997, pp. 33-56.
- [31] M. Lee and H. Takagi. "Integrating design stages of fuzzy systems using genetic

algorithms," in *Proc. of the IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'93)*, 1993, pp. 612-617.

- [32] M. Lee, H. Esbensen, and L. Lemairte, "The design of hybrid fuzzy evolutionary multiobjective optimization algorithms," in *Proc. of the 1995 IEEE/Nagoya University World Wisepersons Workshop (WWW95) on Fuzzy Logic and Neural Networks/Evolutionary Computation*, 1995, pp. 118-125.
- [33] Z. Michalewicz, *Genetic Algorithms + Data Structures + Evolution Programs*, 3rd ed., New York: Springer Verlag, 1996.

CHAPTER 3

REVIEW OF 1-D CELLULAR AUTOMATA AND DENSITY CLASSIFICATION

Abstract

Given an initial condition of random bits for a 1-D cellular automata with periodic boundary conditions, what is the optimal rule for determining if the initial condition contained a majority of 1s or 0s? Called the density classification and the majority problem, this problem has been studied as a starting point for understanding how locally interacting systems can perform global computation. This paper is a review of recent work.

1 Cellular Automata

Cellular automata are discrete space and time dynamical systems. They exist on a lattice of discrete cells and evolve from time step to time step using discrete rules [1]. cellular automata have been used to model a great number of scientific processes. Chemical oscillations [2], crystal growth [3], fluid dynamics [4], and biological patterns like those seen in mollusk shells [5].

One-dimensional cellular automata can be thought of as a line of cells, each in a state. The state can be binary on/off or any other discrete and finite range of values. The rules of the system define the automata. The rules list the next state for a particular cell in the next time step, given the cell's current state and the states of its neighboring cells.

For a $k = 2$, $r = 1$ system, k being the number of states and r being the number of neighbors on each side, the system has an action for each input possibility and an output defining the next state for the cell. A common scheme is for the lattice to “wrap” edge to

edge [1] like a torus.

Let's examine a $k = 2$, $r = 1$ or binary 2 neighbor rule set. The number of actions, or sub-rules, for a particular rule system is defined as $n = 2^{(2r + 1)}$. For $k = 2$, $r = 1$ system this gives us eight sub-rules. The total number of different rules for a given system is k^n . For our system this gives us 256 possible rules. The rule in Figure 3.1 is Rule 90, so called since 90 is the decimal value for 01011010, the CA rule's output states.

Let's look at Rule 90 in action. Figure 3.2 shows the evolution of the system with a single cell turned on in the initial condition or IC. Figure 3.3 shows the evolution of Rule 90 with a random IC.

Neighborhood	000	001	010	011	100	101	110	111
Output	0	1	0	1	1	0	1	0

Figure 3.1: 1-D binary 2 neighbor cellular automata rule 90.

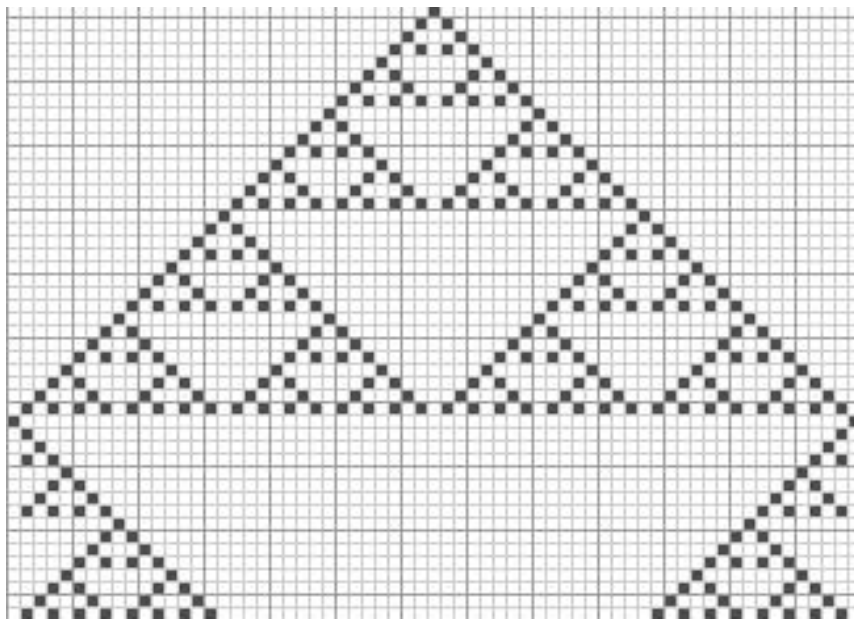


Figure 3.2: Cellular automata rule 90 with a single cell ON in initial condition.

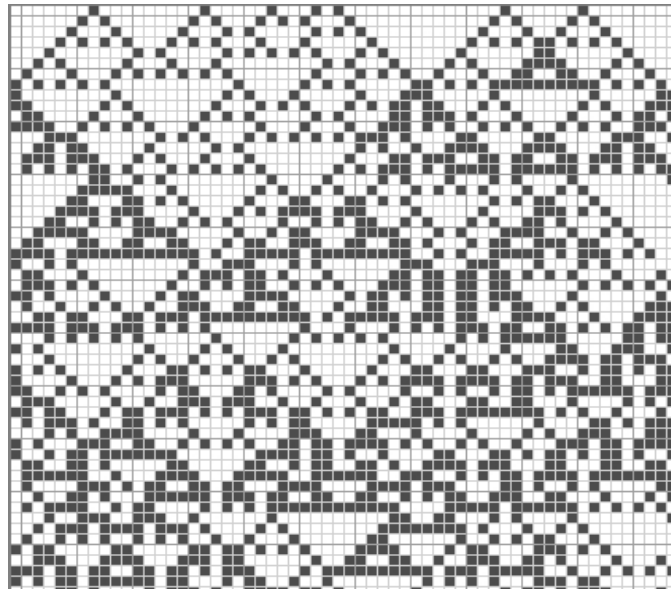


Figure 3.3: Cellular automata rule 90 with a random cells ON in initial condition.

2 The Density Classification Problem

Cellular automata are examples of an algorithm operating on a finite local neighborhood. The global output of the system can be cast as a solution to a given input. The local character of the algorithm makes it a highly advantageous model for parallel systems. The feature of CA systems that enables local interaction to achieve global computation is called “emergent computation” [6].

The Density Classification or Majority problem is stated here:

Given an arbitrary initial condition of a binary 1-D lattice:

If the density of 1s in the IC $> \frac{1}{2}$, the CA should converge to a state of all 1s.

If the density of 1s in the IC $< \frac{1}{2}$, the CA should converge to a state of all 0s.

For this problem, it is common for the size of the lattice to be an odd number so that the condition of density equal to one-half is avoided. It is worth noting here that there exists an inversion calculation that converts a rule of this type to a rule that produces

the opposite of the desired result. Thus, every binary CA rule has a complement rule that produces the opposite output where $0 \leftrightarrow 1$.

A well known and good performing solution for this problem is the Gacs-Kurdyumov-Levin rule [7]. The GKL rule is an example of a $k = 2, r = 3$ CA. See Figures 3.4 and 3.5 for the GKL rule in action.

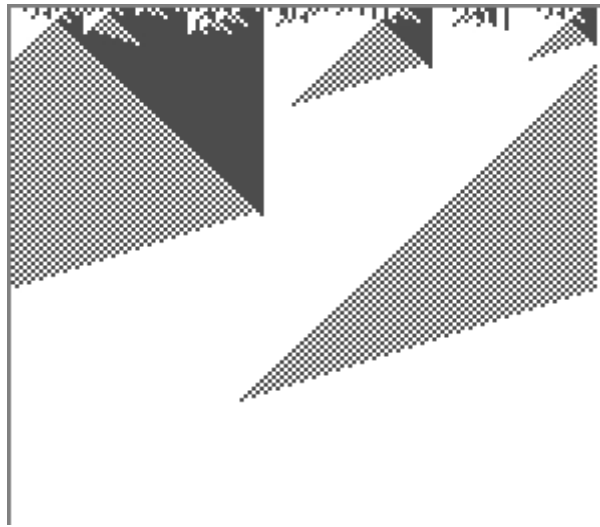


Figure 3.4: GKL rule on a $N=149$ lattice with an initial condition of slightly $< \frac{1}{2}$.



Figure 3.5: GKL rule on a $N=149$ lattice with an initial condition of slightly $> \frac{1}{2}$.

The GKL rule is reported to have an overall classification accuracy of 97.8% on a 149 cell lattice. This score is calculated by choosing a high number of random ICs from a uniform distribution. This score is in contrast to an “area” score of the space where the percentage of correct results in each of 1000 bins, evenly distributed in $[0,1]$. The GKL accuracy for this scoring method is 81.5% for 149 cell lattice.

Figure 3.6 shows us that the ICs nearest one-half are the hardest for the GKL rule to classify correctly. This curve and similar curves of other good classifiers led Land and Belew to wonder if a perfect density classifier existed.

Land and Belew [6] prove by contradiction that no rule exists which classifies density with 100% accuracy for any binary $r \geq 1$ system with finite lattice size N . This proof was done by using five lemmas illustrating various situations and showing that a particular situation produces a contradiction between two lemmas.

Capacarrere et al. [8] show that by altering the output specification, there exists an $r = 1$ binary CA rule that correctly classifies density with 100% accuracy. Rule 184 will

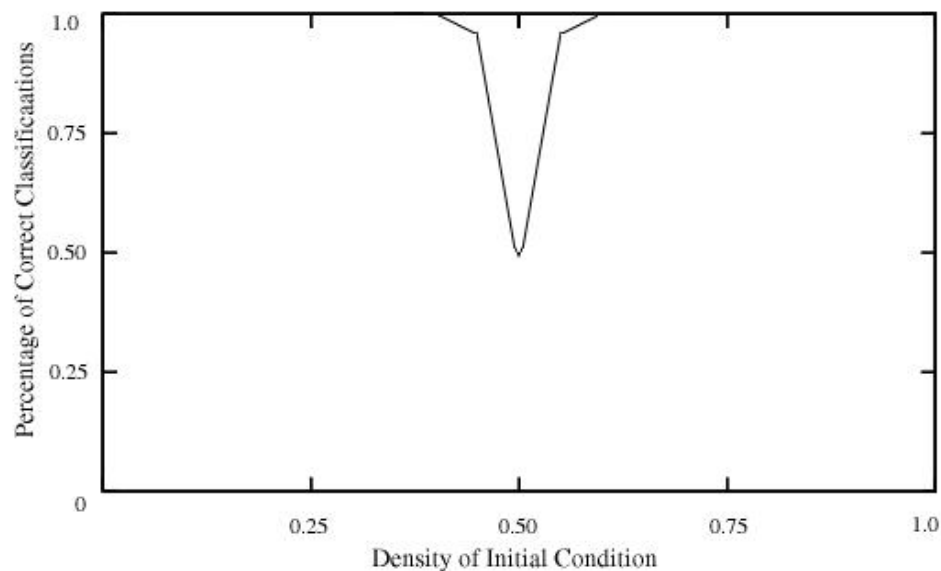


Figure 3.6: The accuracy curve of the GKL rule with lattice size 149.

converge to a checkerboard pattern of alternating 1s and 0s with blocks of repeated 1s or 0s. The state of the cells in repeated blocks define the classification. When an even lattice size is used with exactly one-half initial density, the CA converges to a perfect checkerboard pattern (see Figures 3.7 & 3.8).

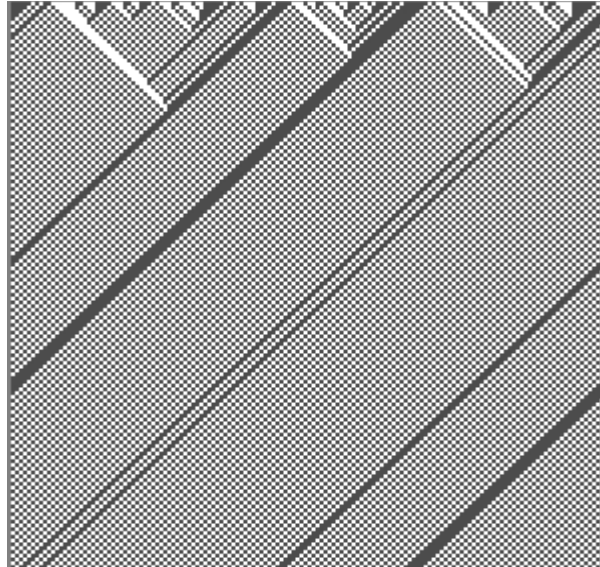


Figure 3.7: Rule 184 on a N=149 lattice with an initial condition of slightly $> \frac{1}{2}$.

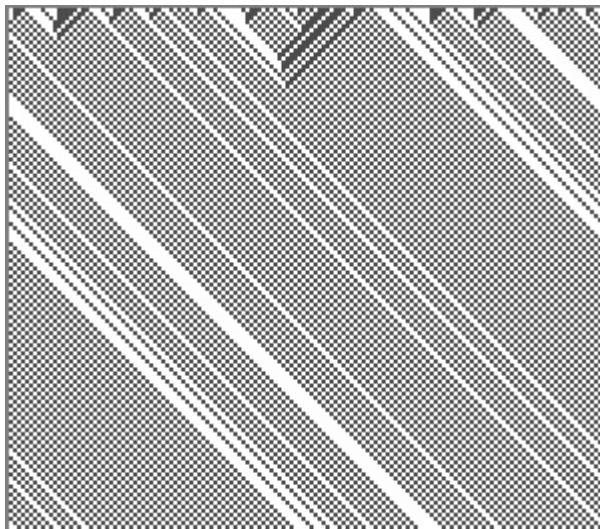


Figure 3.8: Rule 184 on a N=149 lattice with an initial condition of slightly $< \frac{1}{2}$.

Capacarrere and Sipper revisit the problem in [9] and show two necessary conditions exist for a CA to correctly classify density. The first is that the density of the IC must be preserved over time, and the second is that the density of the rule table must be one-half. This means that the proportion of ones to zeros of the rules bits should be equal. This paper generalizes [8] showing that using a checkerboard output specification will result in perfect density classifiers.

Fuks [10] showed that applying two different CA rules consecutively can also solve the density classification problem. Chau et al. [11] generalize the results of [10] and also extend the proof in [6] to show that the general n -ary density classification problem is impossible.

3 Conclusions

We've been introduced to CA systems and a well studied problem in 1-D computational CAs. There is no perfect classifier for the original problem (the solutions above altered the problem definition), and it remains to be seen how accurate a rule can be found. Note that the total rule space for a $k = 2, r = 3$ CA system is 2^{128} . This prohibits any kind of manual search for the best performing rule.

Looking at the GKL accuracy graph, there probably exists an alternate proof to [6] using substitution systems from symbolic dynamics [12]. There exist direct links between cellular automata systems and systems from symbolic dynamics. It may be that an argument exists that the peak of the accuracy curve can only be flattened as some parameter goes to infinity.

The alternate output rule from [8] and the two rule system in [10] were unexpected, but obvious in retrospect. They provide a good method of solving the

density problem.

References

- [1] S. Wolfram, *Cellular Automata and Complexity: Collected Papers*. Reading, MA: Addison-Wesley, 1994.
- [2] Y. Oono and M Kohomoto, "A discrete model of chemical turbulence," *Physical Review Letters*. vol. 55, p 2927, 1985.
- [3] A. Mackay, "Crystal symmetry," *Physics Bulletin*, vol. 27, p. 495, 1976.
- [4] U. Frisch, B. Hasslacher, and Y. Pomeau, "Lattice-gas automata for the Navier-Stokes equation," *Physical Review Letters*, vol. 56, no 14, pp. 1505-1508, 1986.
- [5] J. Campbell, B Ermentrout, and G. Oster, "A model for mollusk shell patterns based on neural activity," *The Veliger*, vol. 28 p 369, 1986.
- [6] M. Land and R.K. Belew, "No perfect two-state cellular automata for density classification exists", *Physical Review Letters*, vol. 74, no. 25, pp. 5148-5150, 1995.
- [7] P. Gacs, G. L. Kurdyumov, and L. A. Levin, "One-dimensional uniform arrays that wash out finite islands," *Problemy Peredachi Informatsii*, vol. 14, pp. 92-98, 1978.
- [8] M.S. Capcarrere, M. Sipper, and M. Tomassini, "A two-state, $r=1$ cellular automata that classifies density," *Physical Review Letters*, vol. 77, no. 24, pp. 4969-4971, 1996.
- [9] M.S. Capcarrere and M. Sipper, "Necessary conditions for density classification by cellular automata," *Physical Review E*, vol. 64, no. 3, pp. 1-4, 2001.
- [10] H. Fuks, "Solution of the density classification problem with two cellular automata rules," *Physical Review E*, vol. 55, pp. 2081-2084, 1997.
- [11] H.F. Chau, L.W. Siu, and K.K. Yan, "One dimensional n-ary density classification using two cellular automata rules," *International Journal of Modern Physics C*, vol. 10, no. 5, pp. 883-899, 1999
- [12] D. Lind and B. Marcus, *An Introduction to Symbolic Dynamics and Coding*. New York: Cambridge University Press, 1995.

CHAPTER 4

FUZZY EVOLUTIONARY CELLULAR AUTOMATA¹**Abstract**

An application of adaptive genetic algorithms to find optimal cellular automata rules to solve the density classification task is presented. A study of the statistical significance of previous results of the evolutionary cellular automata, EvCA, model is detailed, showing flaws in the fitness function. A brief review of recent work in advanced GAs and fuzzy-adaptive GAs is given. These techniques are then applied to the EvCA model to show improvement in convergence speed and more effective search of the optimization landscape.

1 Introduction

We reintroduce an application of genetic algorithms (GAs) to cellular automata., using the GA to evolve rules for performing global computations with simple localized rules. A new more accurate fitness function is introduced to compensate for inaccuracies in the old model. In addition, we extend the model to include fuzzy-logic-controlled GA parameter adaptation

2 Evolutionary Cellular Automata

The EvCA group at the Santa Fe Institute has authored many papers on using the GA to evolve cellular automata (CA) rules to perform computation [1, 2, 3, 4, 5]. The intent of the research was an initial step toward using GAs to enable decentralized computation in distributed multi-processor systems.

¹ Coauthored by James Neal Richter and David Peak

Cellular automata are discrete space and time dynamical systems with localized parallel interaction. The universe of a CA is a grid of cells, where each cell can take on one of k states. The evolution of the CA in time is determined by a set of rules. See Wolfram [6] for a more detailed background. Cellular automata systems have been used to perform a variety of computational tasks, density classification, synchronization, random number generation, etc. [2].

The simplest form of a CA is a binary state, one-dimensional model where the current state of the space is defined by the binary states of the individual cells. At each time step, the state is formed by applying a set of transition rules to the previous state. The neighborhood r is the number of cells on either side of the current cell that affect the cell's state in the next time step. The number of transition rules in such a system is defined as k^{2r+1} .

Figure 4.1 displays the $k=2$, $r=1$ CA system and table of 8 rules. This rule is referred to as Rule 90, the conversion of the 8 bit rule outputs into a decimal number. Figure 4.2 displays Rule 90 in action given a single cell ON in the initial condition of the lattice.

Neighborhood	000	001	010	011	100	101	110	111
Output	0	1	0	1	1	0	1	0



Figure 4.1: 1-D binary 2 neighbor cellular automata rule 90.

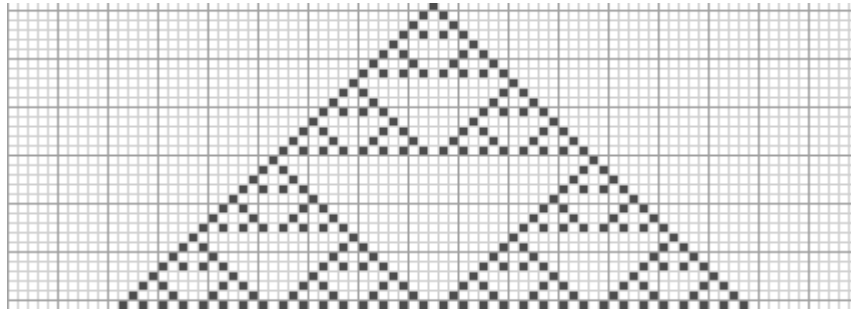


Figure 4.2. Cellular automata rule 90 with a single cell ON in initial condition.

3 Density Classification

The density classification problem is defined here. Given a CA rule applied to a randomly initialized starting state, after M time steps the output should be as follows.

Let p_0 be the percentage of the number of 1s in the initial state.

If $p_0 < 1/2$ then at $t = M$ the system is relaxed to a state of all 0s.

If $p_0 > 1/2$ then at $t = M$ the system is relaxed to a state of all 1s.

Note: $p_0 = 1/2$ is undefined, and is avoided here by using an odd lattice width N .

Land and Belew [7] proved that no binary, $r \geq 1$ CA rule can perfectly classify all possible initial configurations. Fuks [8] and Chau et al. [9] demonstrated that using two successive CA rules, perfect density classification is possible. Capcarrere et al. [10] showed that with a modification of the desired output state, a system exists that can perfectly solve the density problem.

Gacs et al. [11] presented a hand designed $k = 2$, $r = 3$ CA rule for the density classification task (for the original output specification). It appears that as $N \rightarrow \infty$, the error rate of the GKL rule decreases [2]. Later in the paper we will show a statistical analysis of the GKL rule's performance score. Figure 4.3 shows an example run of the GKL density classification rule.

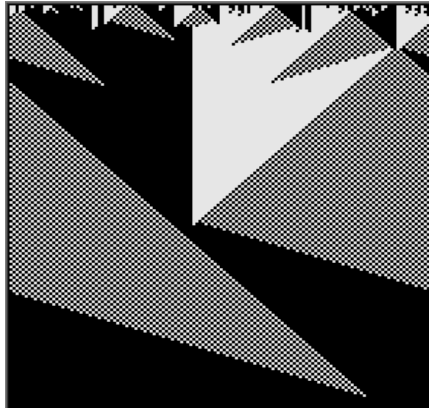


Figure 4.3: GKL rule in action with a random initial condition.

4 EvCA Algorithm

The EvCA system of SFI is reviewed here [2]. The goal of the system is to use the GA to search the space of possible rules in an attempt to find rules that perform a specific computation. Using a $k = 2$, $r = 3$, $N = 149$ CA system for the density classification task with the original output specification, a fitness function is defined as follows:

- (1) Randomly choose $I=100$ Initial Conditions (ICs) uniformly distributed over $p_0 \in [0.0,1.0]$.
- (2) Half of the ICs have $p_0 < 1/2$, the other half have $p_0 > 1/2$.
- (3) Run each rule M times where M is from a Poisson distribution with mean 320.
- (4) Performance Fitness is fraction of I ICs that produce the correct final result.

Varying M ensures that we do not evolve rules that overfit to a given M . Choosing random ICs with a coin-flip for each bit would result in the ICs being binominally distributed. As the ICs near $p_0 = 1/2$ are the most difficult to classify, this method would result in a more difficult fitness task especially in the early generations. Mitchell et al. [3]

note that the performance fitness measure produced qualitatively similar results to using proportional fitness, where partial credit is given according to the percentage of correct states in the final time step. The proportional fitness metric was used with success in other EvCA papers.

The genome is represented as the lexicographic ordering of the CA rule's bits. A $k = 2, r = 3$ CA rule is $2^7 = 128$ bits long. The bits are arranged in ascending neighborhood order, from 0000000 to 1111111. The EvCA system used 100 individuals with 20% elitism. The GA is run for 100 generations. Mutation rates in the papers varied, we chose 0.03. 50 runs of this GA system were performed.

5 Randomized Fitness Function Analysis

The total space of possible initial conditions for a $N = 149$ CA grid is 2^{149} . The EvCA experiments used 100 random ICs, or $100/2^{149} = 1.4 \times 10^{-43}$ percent of the total IC space. No justification was given for this choice other than saving computation time.

We believe that the choice of $I=100$ gives the fitness function a statistically insignificant number of ICs to test each rule. We performed a significance analysis where eight rules with varying fitness scores were tested 25 times with 100, 1000, and 10,000 random ICs and 10 times with 100,000 ICs. Figure 4.4 summarizes the results with range of variance of each rule. Note that the variance decreases as the number of ICs increases!

We can see that 100 ICs are not enough to determine fitness to a sufficient accuracy for low-scoring rules. During the initial generations of the GA run, the population is likely to have low average fitness scores. Note it is likely that with 100 ICs, the variance of the fitness scores could exceed the overall variance of the generational

	100 ICs	1000 ICs	10000 ICs	100000 ICs
GKL	98.0 ± 1.0	98.3 ± .70	98.0 ± .03	98.0 ± .01
Rule 1	92.6 ± 7.7	93.4 ± .68	93.2 ± .04	93.1 ± .02
Rule 2	90.2 ± 5.5	89.9 ± .58	89.7 ± .10	89.5 ± .03
Rule 3	87.1 ± 3.2	87.1 ± .53	87.0 ± .07	87.1 ± .02
Rule 4	81.6 ± 5.8	81.0 ± .58	81.3 ± .05	81.5 ± .01
Rule 5	77.8 ± 14.1	77.7 ± 1.2	77.6 ± .10	77.6 ± .05
Rule 6	63.7 ± 4.1	64.0 ± .23	63.8 ± .03	63.8 ± .01
Rule 7	61.7 ± 14	61.6 ± 1.4	61.3 ± .12	61.5 ± .04

Figure 4.4: Initial condition significance test with scores in percentages.

average fitness. The generational average fitness is the average score at each generation step across all runs of a single EvCA experiment.

Given this, the pre-crossover ranking of the population is probably far from accurate and we have, in effect, a semi-randomized ranking of individuals! The effectiveness of elitism in the early generations is also questionable, as individuals with a good fitness score may fall out of the top 20% due to a unlucky assignment of 100 ICs. It was also noticed that a large majority of 'winning' rules in the standard EvCA model reach high scores with a lucky assignment of Initial Conditions. Post-run validations of winning rule scores typically resulted in a several percentage point drop in score.

We believe that an insufficient number of ICs impedes the GA's explorative nature and the speedy ascent of average fitness score in the early generations. Notice that the highly fit rules are able to have accurate scores with low numbers of ICs. This means that the exploitive nature of the GA is reasonably unaffected, in so far as the variance of the generational average score does not exceed the individual fitness variances given here.

6 Adaptive EvCA

We propose a new EvCA fitness function that will ensure reasonable accuracy, as well as conserve CPU time by short circuiting the fitness evaluation when a rule is shown to be sub-standard. We used a proportional fitness scoring method, although this should make no significant difference for comparison purposes. Each individual is subjected to an initial 250 ICs. At this point the fitness score is evaluated, a target number of total ICs is then determined and the fitness function continues or quits as appropriate. Here is the fitness function:

If the preliminary average score f_p is 75% or below stop.

If $f_p \in [75, 85)$ perform an additional 250 ICs

If $f_p \in [85, 90)$ perform an additional 500 ICs

If $f_p \in [90, 95)$ perform an additional 750 ICs

If $f_p \geq 95$ perform an additional 2500 ICs

While the choice of 250 ICs does not fully address the randomized nature of fitness scoring in the early generations, it does improve accuracy at a reasonable cost. One goal of this fitness function is to help ensure that the elite individuals will be correctly identified early.

In addition, for highly fit individuals we chose to drastically increase the number of ICs. This should help ensure that the very highly fit rules in the elite population are correctly sorted during the latter exploitive generations of the GA. The need of 2500 ICs is not totally supported by the table above. We choose this number in the spirit of erring on the side of accuracy.

7 SEvCA and MevCA

The addition of a more accurate fitness function does let us make one computationally important improvement: it is not necessary to retest the elite population in each generation. Mitchell [12] outlines a GA where a percentage of the population is 'overlapping'; the remaining percentage is regenerated and evaluated for fitness. While this so called 'steady state' GA should not produce qualitatively different results, we do expect, due to subtle variations in the selection procedures, that there will be slightly different quantitative results.

Cantú-Paz [13] is an excellent survey of multipopulation GA models. We chose a coarse-grained multi-population model with frequent stepping-stone migration. The parameters are as follows: five populations with 20 individuals and five eligible for migration. An elite setting of 10 per population was also chosen, and these elite individuals were tested only once. The elite individuals are the top 10 performing individuals.

Notice that this means that there are 50 total elite individuals in each generation. While we do not claim to exactly understand the dynamics of this particular GA, the results indicate that this model performs very well in comparison to the standard EvCA at a great savings of computational time and no sacrifice of individual fitness accuracy. Fitness curves for the algorithms will be given later.

8 Fuzzy Adaptive EvCA

Here we introduce two new models for EvCA based on advanced GA models and dynamic adaptation of the GA parameters. Parameter adaptation in GAs is a much talked about but seldom utilized technique. Bäck [14] discusses a variety of parameter

adaptation issues. Lee and Takagi [15] laid early ground work for using a fuzzy logic controller to adapt population size, mutation rate, and crossover rates. Shi et al. [16] introduced a straight-forward set of fuzzy rules for adapting the mutation and crossover rates of a GA. See Figure 4.5 for a diagram of a fuzzy adaptive GA. See Figure 4.6 for a listing of the fuzzy rules.

We incorporated the Shi rule set into the SEvCA system. We defined the fuzzy parameter $BF \in [0.5, 1.0]$ divided into three overlapping triangular membership functions. The fuzzy parameter $UF \in [0, 10]$ is similarly configured. The fuzzy parameter $VF \in [0.0, .40]$ is divided between three overlapping triangular membership functions heavily skewed around the typical observed values (roughly .05 to .30).

Similar to FSEvCA, the FMEvCA system adds the multi-population model from MEvCA. Note that no additional multi-population parameter adaption is used here. No known fuzzy rule system exists for dynamically adjusting the additional multi-population GA parameters. Figure 4.7 displays the best fitness performance curves of the SEvCA (steady state), MEvCA (multi-population) algorithms, FSEvCA (fuzzy steady state), and FMEvCA systems (fuzzy multi-population).

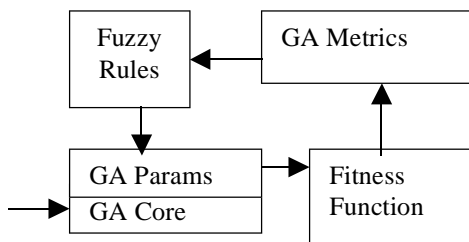


Figure 4.5: Fuzzy adaptive GA diagram.

IF BF is low	THEN MR is low and CR is high
IF BF is medium and UN is low	THEN MR is low and CR is high
IF BF is medium and UN is medium	THEN MR is medium and CR is medium
IF BF is high and UN is low	THEN MR is low and CR is high
IF BF is high and UN is medium	THEN MR is medium and CR is medium
IF UN is high and VF is medium	THEN MR is high and CR is low
IF UN is high and VF is low	THEN MR is high and CR is low
IF UN is high and VF is high	THEN MR is low and CR is low

Figure 4.6: Rules for crossover and mutation rate adaptation from Shi et al. [16]. BF = Best Fitness, UN = number of generation since last BF change, VF = Variance of Fitness, MR = Mutation Rate, CR = Crossover Rate.

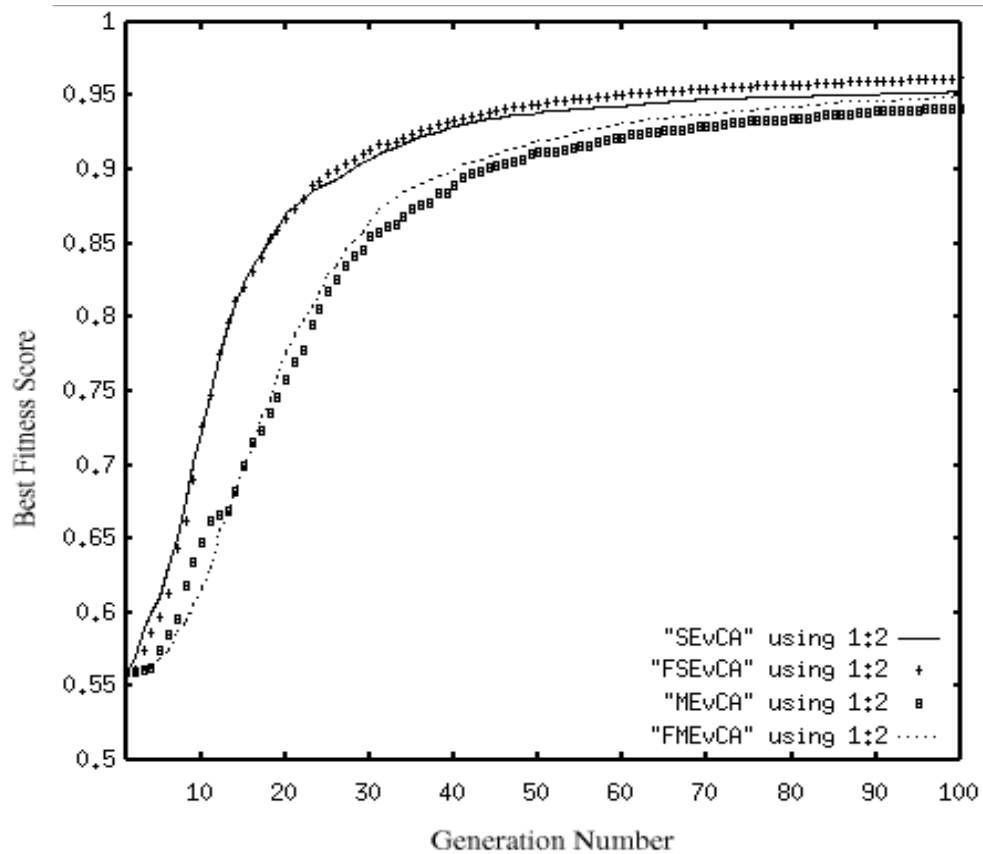


Figure 4.7: Performance of EvCA algorithms, fitness on Y-axis, generation on X-Axis.

9 Conclusions and Future Work

We have added several new techniques to the basic EvCA system with interesting results. Both the improved SEvCA and the new FEvCA system showed good results, while the MEvCA and FMEvCA system showed poor relative performance.

A closer examination of the effectiveness of the adaptive fitness function in relation to the variance of generational elite individual average scores may show that the fitness function could be adjusted to perform additional ICs for low ranking scores. It may also show if the current setting for elite percentage could be adjusted.

We also plan to evaluate a number of other Fuzzy GA rule sets against the density and other CA computational tasks. The exploration of fuzzy parameter adaption for multi-population GAs also looks interesting and fruitful. Other multi-population systems should be investigated. Serious modeling of the dynamics of such a system may be necessary before good fuzzy rules can be crafted.

References

- [1] J.P. Crutchfield and M. Mitchell, "The evolution of emergent computation," *Proceedings of the National Academy of Sciences, USA*, vol. 92, no. 23, pp. 10742-10746, 1995.
- [2] J.P. Crutchfield, M. Mitchell, and R. Das, "Evolving cellular automata to perform computations," *Handbook of Evolutionary Computation* (T. Bäck, D. Fogel, and Z. Michalewicz Eds.), New York: Oxford University Press. 1997.
- [3] M Mitchell, J.P. Crutchfield, and P.T. Hraber, "Evolving cellular automata to perform computations: Mechanisms and impediments," *Physica D*, vol. 75 pp. 361-391, 1994.
- [4] M. Mitchell, P.T. Hraber, and J.P. Crutchfield, "Revisiting the edge of chaos: evolving cellular automata to perform computations," *Complex Systems*, vol. 7, pp. 89-130, 1993.
- [5] M. Mitchell, J.P. Crutchfield, and P.T. Hraber, "Dynamics, computation, and the

- 'edge of chaos': A re-examination," in *Complexity: Metaphors, Models, and Reality*, Santa Fe Institute Studies in the Sciences of Complexity (G. A. Cowan, D. Pines, and D. Meltzer Eds.), New York: Addison-Wesley, 1994, pp. 497-513.
- [6] S. Wolfram, *Cellular Automata and Complexity: Collected Papers*. Reading, MA: Addison-Wesley, 1994.
- [7] M. Land and R.K. Belew, "No perfect two-state cellular automata for density classification exists", *Physical Review Letters*, vol. 74, no. 25, pp. 5148-5150, 1995.
- [8] H. Fuks, "Solution of the density classification problem with two cellular automata rules," *Physical Review E*, vol. 55, pp. 2081-2084, 1997.
- [9] H.F. Chau, L.W. Siu, and K.K. Yan, "One dimensional n-ary density classification using two cellular automata rules," *International Journal of Modern Physics C*, vol. 10, no. 5, pp. 883-899, 1999
- [10] M.S. Capcarrere, M. Sipper, and M. Tomassini, "A two-state, $r=1$ cellular automata that classifies density," *Physical Review Letters*, vol. 77, no. 24, pp. 4969-4971, 1996.
- [11] P. Gacs, G. L. Kurdyumov, and L. A. Levin, "One-dimensional uniform arrays that wash out finite islands," *Problemy Peredachi Informatsii*, vol. 14, pp.92--98, 1978.
- [12] M. Mitchell, *An Introduction to Genetic Algorithms (Complex Adaptive Systems Series)*, Cambridge, MA: MIT Press, 1996.
- [13] E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles, Reseaux et Systems Repartis*. vol. 10, no. 2. pp. 141-171, 1998.
- [14] T. Bäck, "Self-adaptation in genetic algorithms," in *Proc. of the First European Conf. on Artificial Life*, 1992, pp. 227-235.
- [15] M. Lee and H. Takagi. "Dynamic control of genetic algorithms using fuzzy logic techniques," in *Proc. of the Fifth Int. Conf. on Genetic Algorithms (ICGA'93)*, 1993, pp. 76-83.
- [16] Y. Shi, R. Eberhart, and Y. Chen, "Implementation of evolutionary fuzzy systems," *IEEE Transactions on Fuzzy Systems*, vol. 7, no. 2, 1999, pp. 109-119.

CHAPTER 5

CONCLUSIONS

The intent of this thesis is to introduce the reader to adaptive genetic algorithms, evolutionary cellular automata systems, and to show that combining the two results in improved performance.

The experiments in Chapter 4 and the notes in the Appendix show that adaptive GAs can offer better performance to the classic EvCA model. The finding good solutions to the density classification problem is a computationally difficult problem. These experiments produce a proof of concept, but there is much research to be done. The fitness function needs work, and other adaptive rule sets should be evaluated comparatively.

APPENDICES

APPENDIX A. EXPERIMENT NOTES ON FEVCA, FUTURE
DIRECTIONS, AND SOURCE CODE

1 Experiment Notes on FEvCA, MEvCA, and FMEvCA

In Chapter 4 it was demonstrated that several new ideas show promise. The idea of using fuzzy adaptive rules to improve upon the standard EvCA algorithm results in faster convergence time and fitter classification rules. Here we will expand upon the results presented in Chapter 4.

1.1 DeJong Benchmarks

A standard approach to introduce new GA techniques is to validate them versus the standard GA on the DeJong functions [6]. This suite is a well used benchmarking test of new GA techniques. See chapter 2 for more information.

A standard steady state GA was used with the parameters outlined in Figure A.1. As described in Chapter 3, we used the Shi [7] fuzzy adaptive rule set and ran .

GA Parameters	Fuzzy Mutation rates	Fuzzy Crossover rates
Population Size: 30	Low 0.001	Low 0.90
Crossover Rate: 0.9	Medium 0.010	Medium 0.95
Mutation: 0.001	High 0.100	High 1.00
Generations: 400		
Replacement: 0.25		

Figure A.1: Summary of parameter settings for evolutionary cellular automata experiments.

Four GAs were implemented and run with DeJong functions 1-4 as the optimization function. The multi-population GAs were implemented with 3 populations of 10 individuals with single individual migration at each generation. Each GA ran for 400 generations. A replacement factor of 25% was used for each GA.

For the following performance graphs here are the meanings of the GA acronyms:

SteadyState GA	SGA
Fuzzy SteadyState GA	FSGA
MultiPopulation GA	MGA
Fuzzy MultiPopulation GA	FMGA

The multi-population GA has multiple independent populations. Each population (deme) evolves using the steady-state GA. Migration is performed at each generation using a stepping-stone model, a number of each populations best individuals are cloned in other populations and replace the worst performing individuals of the receiving population. See [8] for a great overview of multi-population GAs. See the diagram of a three population stepping stone GA and migration routes in Figure A.2. Each population contains six binary genome individuals.

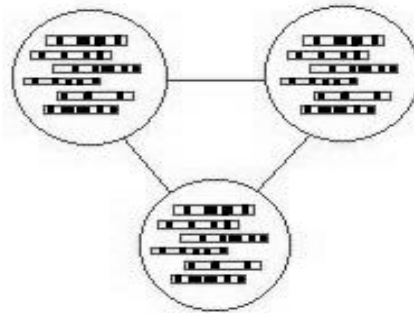
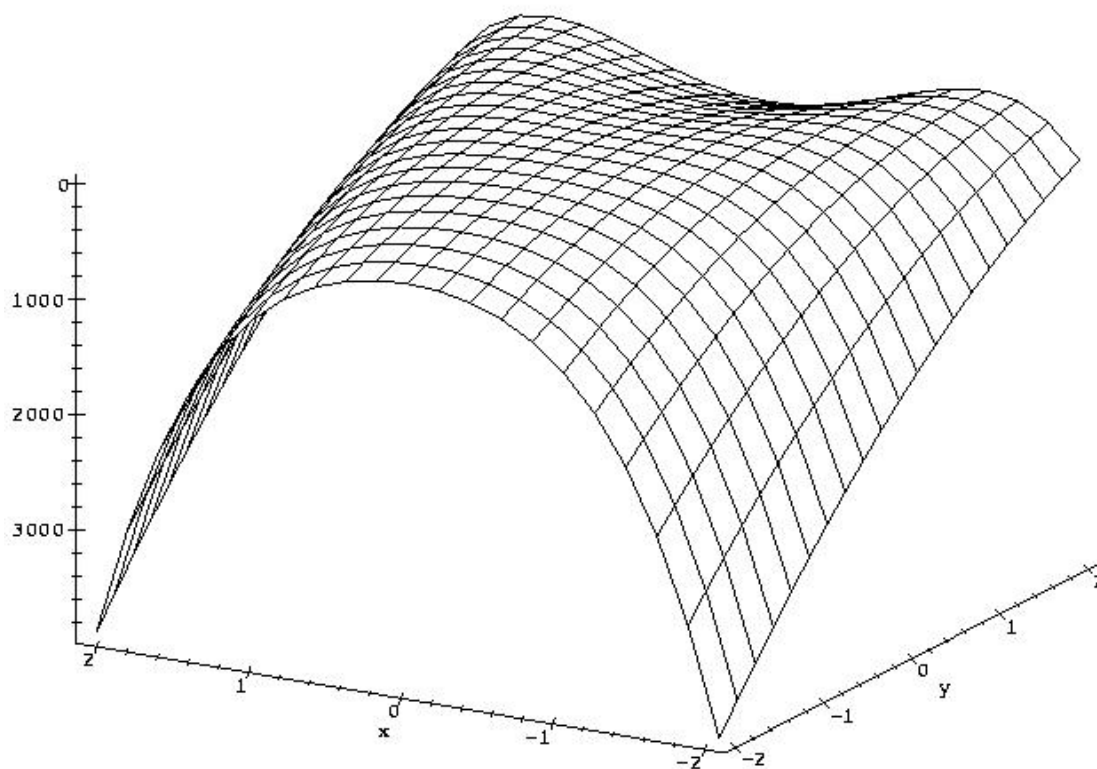


Figure A.2: Stepping stone mutli-population GA model.

The following Figures are the average best-fitness performance curves for 50 runs of each algorithm on each DeJong functions 2-4. The trivial DeJong Function 1 is not detailed here, as all algorithms did very well on it.

1.2 DeJong Function 2

The following pages discuss the experiments with DeJong Function 2. Figure A.3 defines the function and shows an inverted function graph. Figure A.4 is a performance graph of the four GAs showing convergence speed of the GA to the global maximas.



DeJong Function 2: $F2(x1,x2) = 100 * (x1*x1 - x2)^2 + (1 - x1)^2$
 where each x is in the range $[-2.048, 2.048]$.

$MAX(F2) = F2(\pm 2.048, -2.048) = 3905.93$

$MIN(F2) = F2(1,1) = 0$

Figure A.3: DeJong function 2.
 Note that the function graph is inverted for easier visualization.

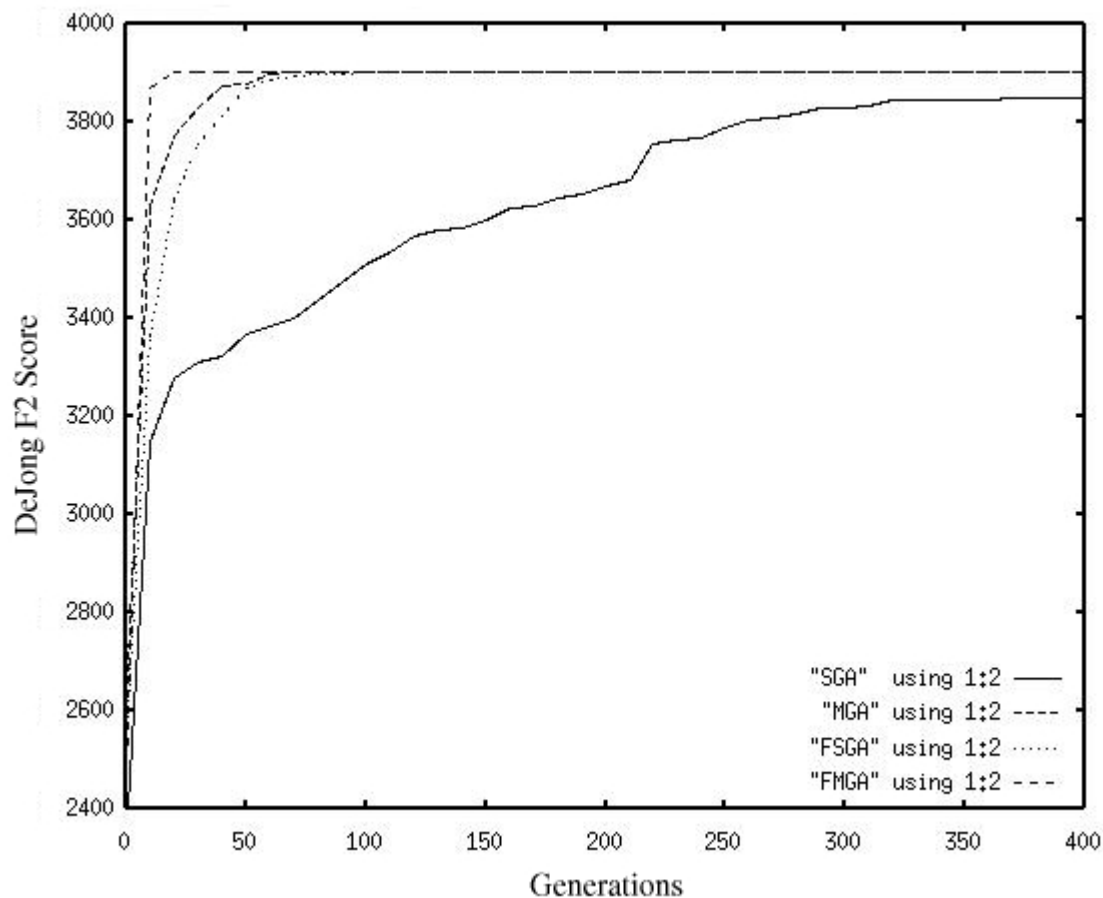


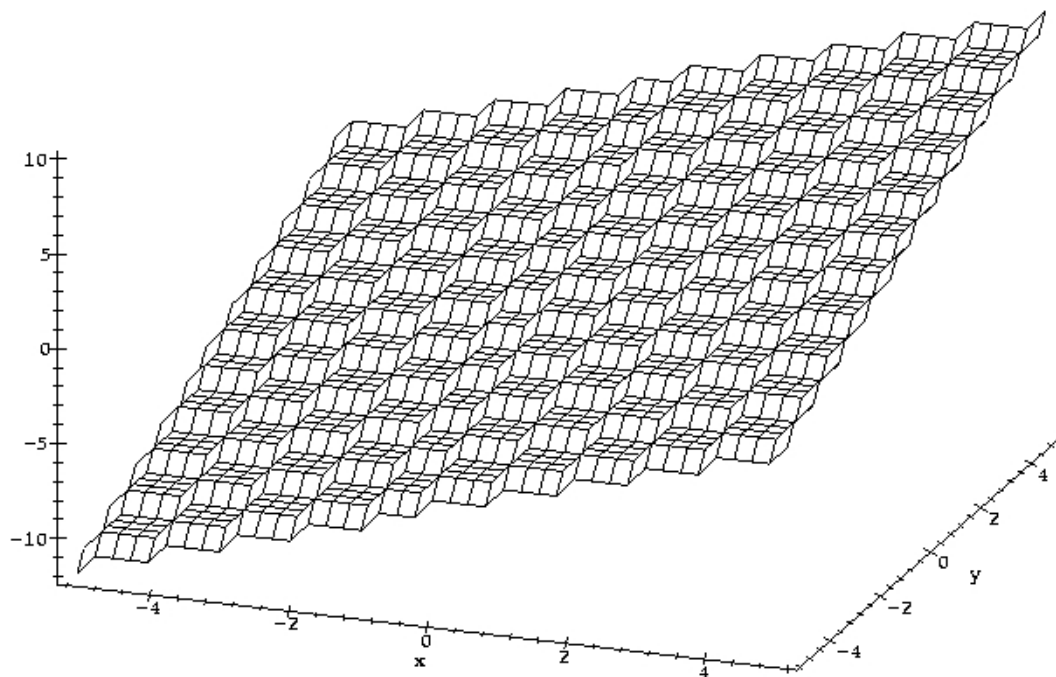
Figure A.4: DeJong function 2 maximization performance. Note there are 2 global optimas with same function value when maximizing.

The FMGA is the winner for this trial. There is little difference between the MGA and FSGA, probably less than is statistically significant. The FMGA has excellent convergence speed, finding an optima in about 25 generations. The three advanced GAs all outperform the SGA in convergence time and final solution quality. Note that in some runs of the SGA, an optimal solution was found.

1.3 DeJong Function 3

Next we look at the experiments with DeJong function 3. Figure A.5 defines the function and shows a graph of a 2-dimensional version of this function. Figure A.6 is a performance graph of the four GAs showing convergence speed of the GA to the global maxima.

Again, the FMGA outperforms all others and by a wider margin. FSGA and MGA are still so close together that their difference is likely not statistically significant. The SGA does poorly.



DeJong Function 3:

$$F3(x_1, x_2, x_3, x_4, x_5) = \lfloor x_1 \rfloor + \lfloor x_2 \rfloor + \lfloor x_3 \rfloor + \lfloor x_4 \rfloor + \lfloor x_5 \rfloor$$

where each x is in the range $[-5.12, 5.12]$.

$$\text{MAX}(F3) = F3(5.12, 5.12, 5.12, 5.12, 5.12) = 25$$

$$\text{MIN}(F3) = F3(-5.12, -5.12, -5.12, -5.12, -5.12) = -30$$

Figure A.5: Graph of a 2-D version of DeJong Function 3.
Note that the function is a 2-D version for easier visualization.

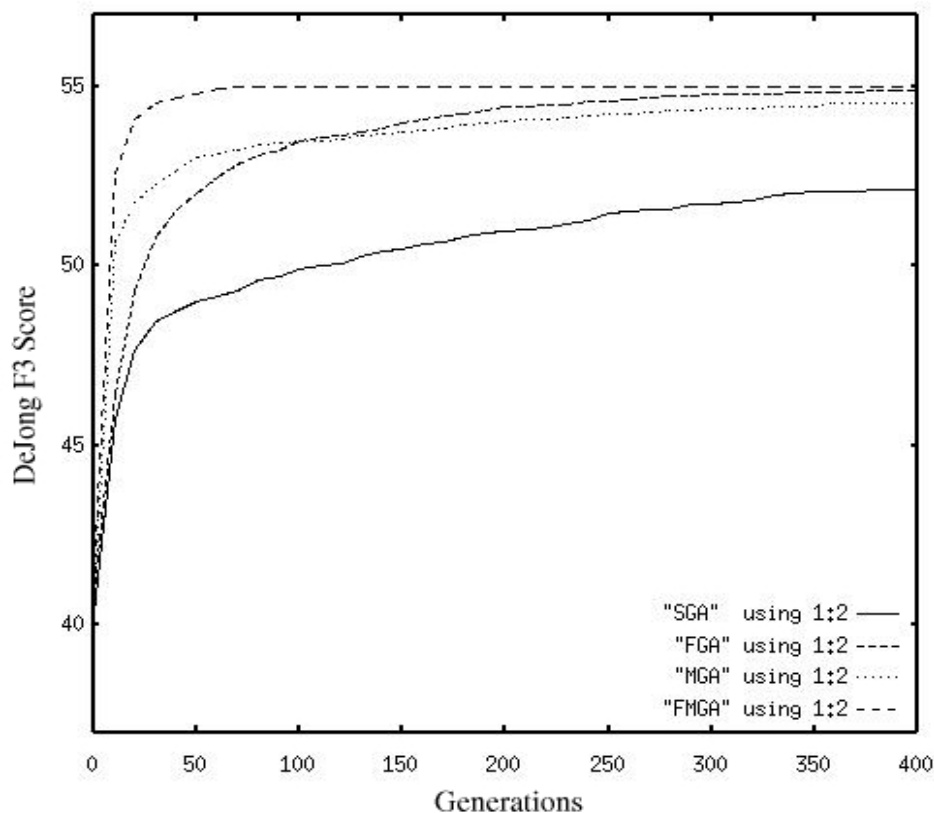
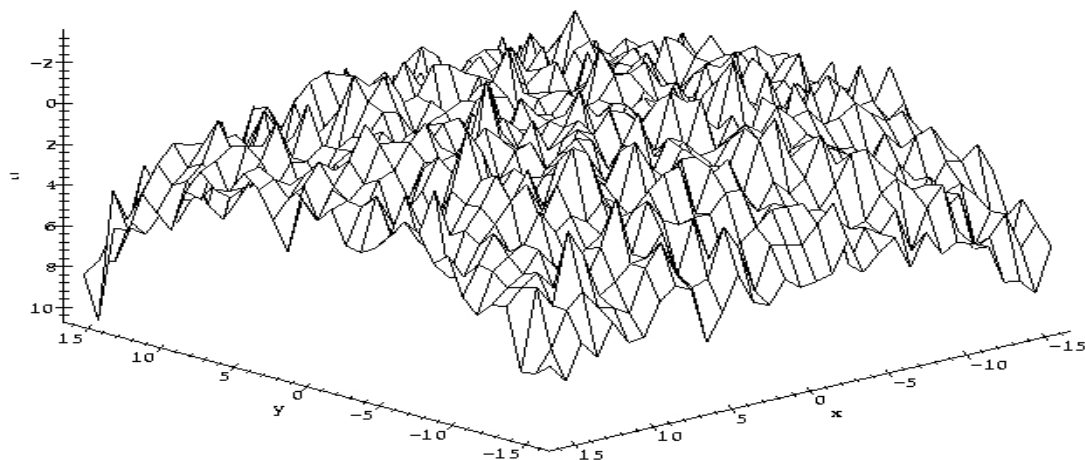


Figure A.6: DeJong function 3 maximization performance.
 Note: 25 was added to F3 function value to offset negative numbers.

1.4 DeJong Function 4

Now we turn our attention to the experiments with DeJong function 4. DeJong Function 4 is a quartic 'hill' with Gaussian noise. The GA should find the single global minima if the parameter settings are such that the population is able to explore a large enough local area to find the gradient.

Figure A.7 defines the function and shows an inverted graph of a 2-dimensional version of this function. Figure A.8 is a performance graph of the four GAs showing convergence speed of the GA to the global maximas. Figure A.9 is a performance graph of the four GAs showing convergence speed of the GA to the global minimum.



DeJong function 4

$$f_4(x_i) = \sum_{i=1}^{30} \left\{ i * x_i^4 + \text{Gauss}(0,1) \right\}$$

where each x is in the range $[-1.28, 1.28]$

$\text{MAX}(F_4) = F_4(\pm 1.28, \pm 1.28, \dots, \pm 1.28) = 1248.2$

$\text{MIN}(F_4) = F_4(0, 0, \dots, 0) = 0$

Note: There are 2^{30} optimal values when maximizing

Figure A.7: DeJong function 4.

Note that the function graph is an inverted 2-D version for easier visualization.

The maximization run again features a dominating performance by FMGA, with MGA tracking FMGA with a weaker maximum solution. Interestingly, FSGA does well in the early rounds only to converge prematurely to a low scoring maxima. The fourth DeJong function is known to be much harder than the first three. SGA's weak performance is somewhat surprising since there are 2^{30} optimal values.

The minimization run is less interesting and again is dominated by the multi-population GAs with the FMGA having a slight edge over MGA. The SGA does poorly, showing the value of using a more advanced GA model. The fuzzy adaptive methods slightly outperform their non-adaptive counterparts.

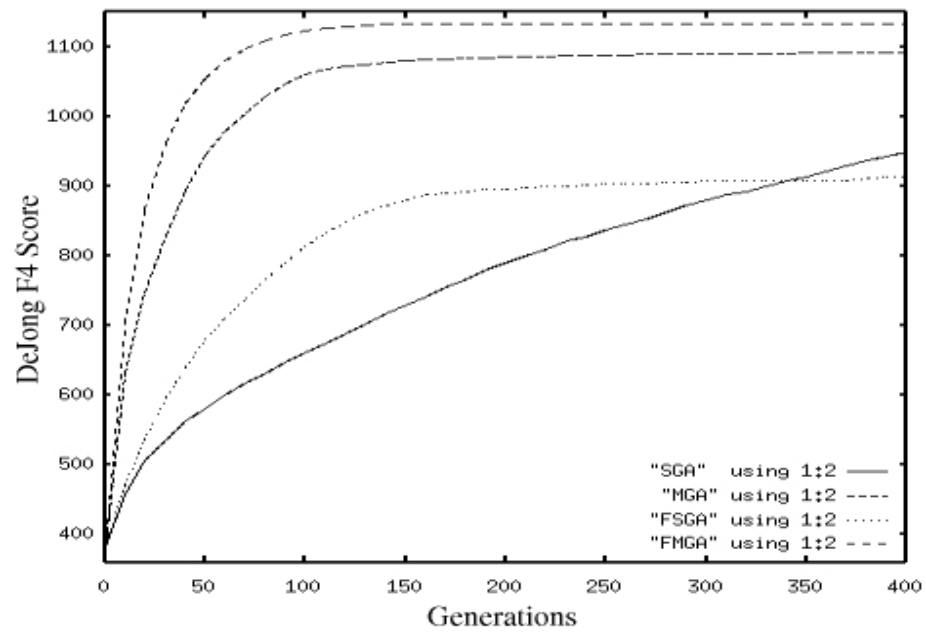


Figure A.8: DeJong function 4 maximization performance.
 Note: There are 2^{30} optimal values when maximizing.

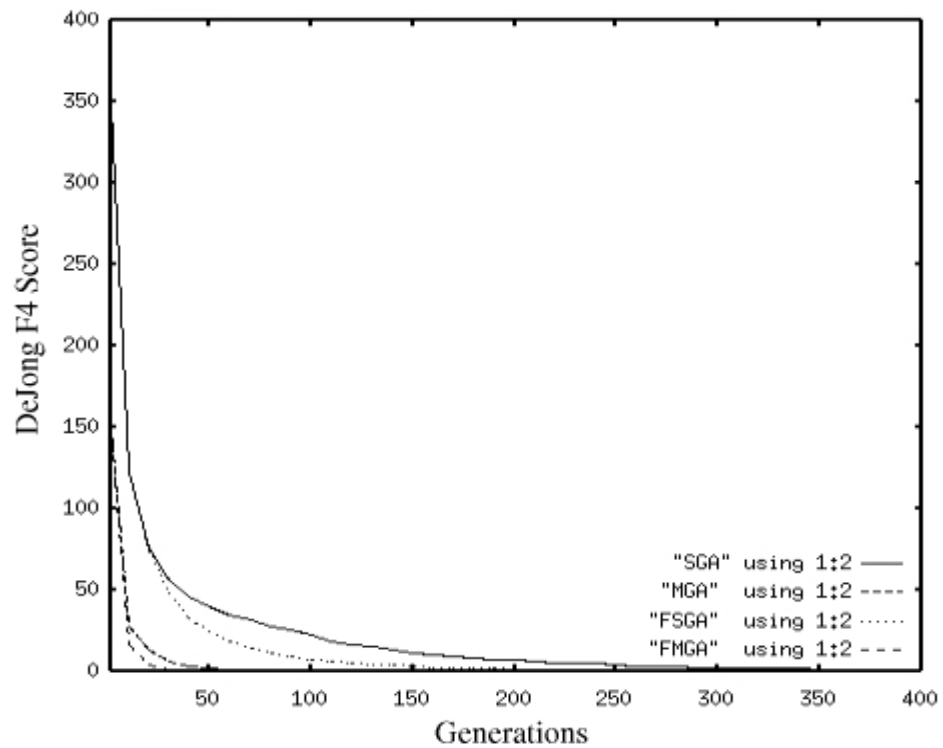


Figure A.9: DeJong function 4 minimization performance.
 Note: There is a single optima at (0,0) when minimizing.

1.5 Benchmark Observations

We see that the fuzzy adaptive GA approach is validated, as shown in many references. Note that the new fuzzy adaptive multi-population GA is a great algorithm on these benchmarks. The fuzzy adaptive GA probably needs further tuning to outperform the basic multi-population GA.

2 Questions and ideas for improvement

The FEvCA approach is not fully implemented yet. The Shi adaptive rule set worked well. There are several unanswered questions.

1. *How will the other fuzzy adaptive rule sets mentioned in Chapter 1 perform?*

The other rule sets are in some sense more 'complete' in that they probably cover more situations well. This could help the GA explore the space with a finer grained adaptive power.

2. *The short circuiting of the fitness function can be improved.*

Bad luck choosing the first set of ICs to estimate the general class of the rule for further testing can doom a good rule. In addition, it is possible with this fitness function for rules to get an easy set of ICs and slide into the elite population that is not retested. Fully testing each rule is CPU cost prohibitive. Note, however, that this fitness function does perform much better than the original EvCA function.

3. *Adaptive population sizing needs to be explored.*

A better analysis of the appropriate population size of the system should be undertaken. 100 individuals in a space as large as 2^{128} may not be adequate. And it is likely that a changing population size could both explore the space

better in complex regions and save CPU cycles in homogenous regions.

4. *A more detailed survey should be done of non-fuzzy adaptive GA systems.*

There are many other papers detailing experiences and heuristics that may be missing from the reviewed fuzzy rule sets.

5. *Enforcing population diversity directly may be valuable.*

Indirect methods such as incest prohibition are interesting. Direct methods using a suite of statistical metrics from [1] warrant study and incorporation into the feedback loop.

6. *The MEvCA system had a disappointingly poor performance. Why?*

Multi-population GAs are excellent algorithms that do well on a variety of benchmark functions. They are also easily parallelizable and should be good sources of implicit population diversity. Is it possible that the migration patterns of the population degrade diversity?

7. *Several enhancements to the choice of ICs are worth exploring.*

The idea of switching the random distribution to binomial or partially binomial for high scoring rules should help refine the accuracy of the high fitness selection process and result in better final rules. Care must be taken however to ensure that the enhancement does not kick in too soon and give the GA a harder hill to climb. The alternate 'area' scoring metric may be worth exploring as it would subject each rule to known suite of random ICs that cover the density space well.

3 Future Directions

Here are some discussion points and questions for further study:

1. *What does the EvCA fitness space look like?*

It may be possible to use a self-organizing map to form a 2-D projection of the fitness landscape. Tracking the evolving population in this projection would be interesting.

2. *Study Ph.D. thesis' of R. Das (Colorado State 1998), W. Hordijk (U of New Mexico 1999), and E. van Nimwegen (former Santa Fe Inst. Fellow).*

3. *What do the SFI EvCA group's later papers on statistical mechanics of evolutionary populations have to offer FEvCA?*

It seems likely that much of the analysis in these later papers on populations may have insights that would improve FEvCA.

4. *Fuzzy Adaptation in CoEvCA?*

[3] and [4] introduce Coevolutionary EvCA. They successfully find highly fit rules. Can it be done better/faster?

5. *What about Genetic Programming?*

Andre, et al. showed in [2] that genetic programming is successful for finding CA rules that classify density.

6. *Fuzzy Adaptive Genetic Programming?*

In a conversation with Koza at GECCO-99, he showed hesitation at the idea. I have not seen any papers on this. Adaptive crossover in GP will probably not be fruitful, as the crossover functions are very complex. Aggressive mutation operators are also trouble for GP. What about adaptive population size with population diversity metrics?

7. *Fuzzy Adaptive Evolutionary Strategies?*

How about adding fuzzy adaptation to the classic mutation-only Evolutionary Computation model? Evolutionary Strategies techniques have long used adaptive mutation schemes. How do these stack up against fuzzy adaptive techniques?

8. *Fuzzy Adaptive Co-Evolutionary Systems?*

The idea in general shows promise. Static parameters are shown to be substandard in comparison to a well constructed parameter adaptive system.

9. *What is in Wolfram's new CA book, A New Kind of Science?*

4 Source Code

The following pages contain some of the source code used for the experiments. Listed here is the complete source for the FMEvCA experiment. The source code for other experiments is essentially the same except they used different genetic algorithm C++ classes to implement the specific GA model desired.

Please see the GALib genetic algorithm library documentation for further information. GALib WWW address: <http://lancet.mit.edu/ga/>

All Source Code, unless otherwise specified, is Copyright © 2003 James Neal Richter

and is subject to the following license:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

```

/*****
                                ca.h - description
                                -----
begin                          : Thu Sep 28 2000
copyright                      : (C) 2000 by Neal Richter
email                          : jnr@cc.usu.edu

DERIVED FROM
1 Dimensional Boolean Cellular Automata
Written by Paul Bourke September 1997
http://www.swin.edu.au/astronomy/pbourke/fractals/ca/
(translated from C to a C++ object)
*****/

#include <fstream.h>
#include <gsl_rng.h>
#include <gsl_randist.h>

#ifndef TRUE_FALSE
#define TRUE 1
#define FALSE 0
#endif

#ifndef CA_H
#define CA_H

//As used at SFI EvCA group
#define POISSON_MEAN 320

//these can be 'OR'ed together
#define NO_DISPLAY 0
#define TERMINAL_DISPLAY 1
#define GNUPLOT_DISPLAY 2

/**
 *@author Neal Richter
 */

class SimpleCA {
public:
    unsigned long int niterations;
    unsigned long int maxiterations;
    unsigned long int length;

    unsigned long int num_neighbors;

    //initial value counters
    unsigned long int iv_num_zeros;
    unsigned long int iv_num_ones;
    double iv_percent_ones;

    int *state;
    int *newstate;
    int *rule;

    gsl_rng *rng;

    ofstream outfile;

    //niterations, length, rule_len, a_rule[]
    SimpleCA(int, int, int, int, int[], char *);
    ~SimpleCA();

    void Go(int);
    void TerminalDisplayState();
    void GnuPlotDisplayState();

    int rule_lookup(int);
};

#endif

```

```

/*****
                                ca.cpp - description
                                -----
begin                          : Thu Sep 28 2000
copyright                       : (C) 2000 by Neal Richter
email                            : jnr@cc.usu.edu

DERIVED FROM
1 Dimensional Boolean Cellular Automata
Written by Paul Bourke September 1997
http://www.swin.edu.au/astronomy/pbourke/fractals/ca/
(translated from C to a C++ object)
*****/

#include "stdio.h"
#include "stdlib.h"
#include "math.h"
#include "sys/types.h"
#include "time.h"
#include <sys/timeb.h>

#include <gsl_rng.h>
#include <gsl_randist.h>

#include "ca.h"

//constructor
SimpleCA::SimpleCA(int n, int len, int neighbors, int rule_len, int a_rule[], char *
filename)
{
    struct timeb mytime;

    maxiterations = n;
    length = len;
    num_neighbors = neighbors;

    // get memory
    state = new int[length];
    newstate = new int[length];
    rule = new int[rule_len];

    //initialize memory for safety
    for (int i = 0; i < length; i++)
    {
        state[i] = 0;
        newstate[i] = 0;
    }

    //initialize and seed random number generator
    rng = gsl_rng_alloc(gsl_rng_mt19937);
    ftime(&mytime);
    gsl_rng_set(rng, (unsigned int)mytime.millitm);

    //copy the rule
    for(int i = 0; i < rule_len; i++)
        rule[i] = a_rule[i];

    //open the file
    if (filename != NULL)
    {
        outfile.open(filename);
        if (!outfile) {
            cout << "Output file cannot be opened.\n";
            exit(1);
        }
    }
}

//destructor
SimpleCA::~SimpleCA()
{
    //free memory
    delete state;
    delete newstate;
    delete rule;

    //free rng
    gsl_rng_free(rng);
}

```

```

    //close the file
    outfile.close();
}

//random first state
//processes the rule to make new state
//until niterations
void SimpleCA::Go(int display)
{
    int i,j,k;
    int stop_flag = 0;
    int ones_counter = 0;
    long secs;
    struct timeb mytime;

    /* set up GSL RNG MT19937 */
    //rng = gsl_rng_alloc(gsl_rng_mt19937);
    //seed
    ftime(&mytime);
    gsl_rng_set(rng, (unsigned int)mytime.millitm);
    /* end of GSL setup */

    /* Initialize the state randomly*/
    time(&secs);
    srand(secs);
    iv_num_ones = 0;
    iv_num_zeros = 0;
    iv_percent_ones = 0.0;

    //beware of binomial distribution. Randomly initializing
    //the IC cells individually will result in a binomial distribution.
    //uniform distribution of ICs is desired

    iv_num_ones = gsl_rng_uniform_int(rng, length);
    iv_num_zeros = length - iv_num_ones;
    iv_percent_ones = iv_num_ones/length;

    //alternate implementation of above
    //iv_percent_ones = gsl_rng_uniform(rng);
    //iv_num_ones = (int)((length * iv_percent_ones) + 0.5);
    //iv_num_zeros = length - iv_num_ones;

    for (i = 0; i < length; i++)
        state[i] = 0;

    while (ones_counter < iv_num_ones)
    {
        i = gsl_rng_uniform_int(rng, length);

        if (state[i] != 1)
        {
            state[i] = 1;
            ones_counter++;
        }
    }

    // display initial state
    if(display & TERMINAL_DISPLAY)
        TerminalDisplayState();

    // Poisson Distribution for number of iterations.. mean = 320
    // as used in the EvCA groups papers at SFI

    maxiterations = gsl_ran_poisson(rng, POISSON_MEAN);

    for (niterations=0;niterations<maxiterations; niterations++)
    {
        /* Erase the old state, not really necessary */
        for (j=0;j<length;j++)
            newstate[j] = 0;

        /* Create the next state */
        for (j=0;j<length;j++)
        {
            k = rule_lookup(j);
            newstate[j] = rule[k];
        }
    }
}

```

```

    /* Update the current state */
    for (j=0;j<length;j++)
        state[j] = newstate[j];

    /* Display the results */
    if(display & TERMINAL_DISPLAY)
        TerminalDisplayState();
    /* Plot the results */
    if(display & GNUPLOT_DISPLAY)
        GnuPlotDisplayState();
}

} //end void SimpleCA::Go(int)

void SimpleCA::TerminalDisplayState()
{
    int i;

    for (i=0;i<length;i++)
    {
        if (state[i] == 1)
            cout << '*';
        else
            cout << ' ';
    }
    cout << endl;
} //end void SimpleCA::DisplayState()

// plot [0:81][61:0] "junk" with points pointtype 3 pointsize 1.5
// needs modifier to fill boxes
void SimpleCA::GnuPlotDisplayState()
{
    int i;

    for (i=0;i<length;i++)
    {
        if (state[i] == 1)
            outfile << i+1 << " " << niterations+1 << endl;
    }
} //end void SimpleCA::GnuPlotDisplayState()

int SimpleCA::rule_lookup(int j)
{
    int index = 0;

    switch(num_neighbors)
    {
        case 1:
            index = 4*state[(j-1+length)%length] + 2*state[j] + state[(j+1)%length];
            break;
        case 2:
            index = 16*state[(j-2+length)%length] + 8*state[(j-1+length)%length];
            index += 4*state[j];
            index += 2*state[(j+1)%length] + state[(j+2)%length];
            break;
        case 3:
            index = 64*state[(j-3+length)%length] + 32*state[(j-2+length)%length] +
16*state[(j-1+length)%length];
            index += 8*state[j];
            index += 4*state[(j+1)%length] + 2*state[(j+2)%length] + state[(j+3)%length];
            break;
        default:
            index = -1;
    }

    return(index);
}

```

```

/* -----
mevca.cxx

Neal Richter
Utah State University

Copyright (c) 2000, 2001, 2002 Neal Richter

DESCRIPTION:
  Multi-Population EvCA algorithm for density classification

  Uses GALib library and example source as a guide/template
  Copyright (c) 1995-199 Massachusetts Institute of Technology and Matthew Wall.
  http://lancet.mit.edu/ga/
  Thanks MATT!
----- */
#include <stdio.h>
#include <iostream.h>
#include <time.h>
#include <math.h>
#include <ga/GALDBinStrGenome.h>
#include <ga/GADemeGA.h>
#include <ga/GASimpleGA.h>

#include "ca.h"
#include "fuzzy.h"

#define NUM_POPULATIONS      5
#define POPULATION_SIZE     20 // NUM_POPULATIONS * POPULATION_SIZE = total
                               individuals
#define NUM_ELITE            5 //should be less than POPULATION_SIZE!
#define NUM_MIGRATION       2 //should be less than POPULATION_SIZE!

#define NUM_GENERATIONS     100
#define GENOME_LEN          128

#define NUM_SAMPLES_MAX     2500
#define NUM_SAMPLES_MIN     250
#define NUM_SAMPLES_DEFAULT 500

#define FUZZY_HIGH          3
#define FUZZY_MED           2
#define FUZZY_LOW           1

//global variables -- for convenience
int current_generation;
float current_score_mean;
float current_score_stddev;
float average_samples;
float average_samples_counter;

int supreme_fit_count;
int vhigh_fit_count;
int high_fit_count;
int med_fit_count;
int low_fit_count;
int vlow_fit_count;

//functions
float Objective(GAGenome &);
int get_optimum_num_samples(float);
void shi_fuzzy_adapt(float *, float *, float, float, int);

int
main(int argc, char** argv)
{
  time_t time_now;
  int total_individuals = 0;
  int unchanged_best_fitness_count = 0;
  float new_pcrossover = 0;
  float new_pmutate = 0;
  float old_best_fitness = 0;
  float current_best_fitness = 0;
  float current_score_variance = 0;

  cout << "FMEvCA Algorithm with Intelligent Sampling" << endl;
  cout << "Shi-Like Fuzzy Rules to adapt Mutation and Crossover rates" << endl;

```

```

cout << "k=2 r=3 Density Classification Task" << endl;
cout << "Num CA Interations: Poisson Distribution with mean=" << POISSON_MEAN << endl;
cout << "GADemeGA (parrallel populations with migration)";
cout << "Agressive Exploration. See mutations rates." << endl;
cout << endl;

// See if we've been given a seed to use (for testing purposes). When you
// specify a random seed, the evolution will be exactly the same each time
// you use that seed number.

unsigned int seed = 0;
for(int ii=1; ii<argc; ii++) {
    if(strcmp(argv[ii++], "seed") == 0) {
        seed = atoi(argv[ii]);
    }
}

GAlDBinaryStringGenome genome(GENOME_LEN, Objective);
GADemeGA ga(genome);

ga.nPopulations(NUM_POPULATIONS);
ga.populationSize(POPULATION_SIZE);
ga.nGenerations(NUM_GENERATIONS);
cout << "nPopulations = " << ga.nPopulations() << endl;
cout << "PopulationSize = " << ga.populationSize();
cout << "Num Generations = " << ga.nGenerations() << endl;
total_individuals = ga.nPopulations() * ga.populationSize();

ga.nReplacement(GADemeGA::ALL, (POPULATION_SIZE-NUM_ELITE));
ga.nMigration(NUM_MIGRATION);
cout << "Replacement = ALL" << (POPULATION_SIZE-NUM_ELITE) << ", Migration = " <<
ga.nMigration() << endl;

ga.recordDiversity(gaTrue);
ga.pMutation(0.03);
ga.pCrossover(1.0);

ga.scoreFrequency(1);
ga.flushFrequency(1);

ga.parameters(argc, argv);

time_now = time(NULL);
cout << "Init Time: " << ctime(&time_now) << endl;
cout << "Initializing" << endl;
ga.initialize(seed);
cout << "Evolving" << endl;
time_now = time(NULL);
cout << "Evolve Start Time: " << ctime(&time_now) << endl;

while(!ga.done())
{
    supreme_fit_count = 0;
    vhigh_fit_count = 0;
    high_fit_count = 0;
    med_fit_count = 0;
    low_fit_count = 0;
    vlow_fit_count = 0;

    average_samples = 0;
    average_samples_counter = 0;

    ga.step();
    time_now = time(NULL);
    cout << "Time: " << ctime(&time_now) << endl;
    current_generation = ga.generation();
    cout << "Generation:" << current_generation << endl;

    cout << "Average Number of Samples: " <<
(int)((average_samples/total_individuals)+0.5);
    cout << " ( " << average_samples_counter << " )" << endl;

    cout << "best individual is: \n" << ga.statistics().bestIndividual() << "\n";
    cout << endl << ga.statistics() << endl;
    cout << endl;
    cout << "#supreme_fit_count " << supreme_fit_count << endl;
    cout << "#vhigh_fit_count " << vhigh_fit_count << endl;
    cout << "#high_fit_count " << high_fit_count << endl;
    cout << "#med_fit_count " << med_fit_count << endl;
    cout << "#low_fit_count " << low_fit_count << endl;
    cout << "#vlow_fit_count " << vlow_fit_count << endl;
}

```

```

cout.flush();

current_score_mean = ga.statistics().current(GAStatistics::Mean);
current_score_stddev = ga.statistics().current(GAStatistics::Deviation);
current_best_fitness = ga.statistics().current(GAStatistics::Maximum);

//get adaptive step parameters ready

if (current_best_fitness == old_best_fitness)
    unchanged_best_fitness_count++;
else
    unchanged_best_fitness_count = 0;

old_best_fitness = current_best_fitness;

new_pcrossover = ga.pCrossover();
new_pmutate = ga.pMutation();

//scale & calculate variance
current_score_variance = current_score_stddev*100;
current_score_variance *= current_score_variance;
current_score_variance *= (float)1/100;

//adaptive step
shi_fuzzy_adapt(&new_pcrossover, &new_pmutate, current_best_fitness,
                current_score_variance, unchanged_best_fitness_count);

//adjust the output parameters
ga.pCrossover(new_pcrossover);
ga.pMutation(new_pmutate);

cout << "pMutation " << ga.pMutation() << endl;
cout << "pCrossover " << ga.pCrossover() << endl;

}
cout << endl;

cout << "---" << endl;
cout << "Done with generations.";
cout << "best individual is: \n" << ga.statistics().bestIndividual() << "\n";
cout << "\n" << ga.statistics() << "\n";
cout << endl << endl << "EvCA done." << endl;
cout << "-----" << endl;

return 0;
}

float
Objective(GAGenome& g)
{
    GALDBinaryStringGenome & genome = (GALDBinaryStringGenome &)g;
    float single_score = 0.0;
    float avg_score = 0.0;
    float temp_avg_score = 0.0;
    int gene_len = genome.length();
    int i = 0, j = 0;
    int stop_flag = FALSE;
    int num_samples = 0;
    int optimum_num_samples = 0;
    int *gene_rule = new int[gene_len];

    //copy the gene/rule
    for (i = 0; i < gene_len; i++)
        gene_rule[i] = genome.gene(i);

    SimpleCA myca(149, 149, 3, gene_len, gene_rule, NULL);

    //decide on maximum number of samples based on generation
    num_samples = NUM_SAMPLES_DEFAULT;

    i = 0;
    while((i < num_samples) && (stop_flag == FALSE))
    {
        i++;
        myca.Go(FALSE);
        single_score = 0;
        for(j=0; j<myca.length; j++)
            single_score += myca.state[j];
    }
}

```

```

    if (myca.iv_num_ones > myca.iv_num_zeros)
        single_score /= myca.length;
    else
        single_score = (myca.length - single_score)/myca.length;

    avg_score += single_score;

    if (i == NUM_SAMPLES_MIN)
    {
        temp_avg_score = avg_score/(float)i;

        optimum_num_samples = get_optimum_num_samples( temp_avg_score );

        if (optimum_num_samples <= i)
            stop_flag = TRUE;
        else
            num_samples = optimum_num_samples;
    }
}

average_samples_counter++;
average_samples += i;

avg_score /= (float)i;

delete [] gene_rule;

//population counters
if ( avg_score < .60 )
    vlow_fit_count++;
else if ( avg_score < .75 )
    low_fit_count++;
else if ( avg_score < .85 )
    med_fit_count++;
else if ( avg_score < .90 )
    high_fit_count++;
else if ( avg_score < .95 )
    vhigh_fit_count++;
else
    supreme_fit_count++;

return ( avg_score );
}

//----- get_optimum_num_samples(..) -----
//
// This function takes the preliminary score and returns the number of
// suggested total samples to test this individual.
//
// This function is a loose heuristic based on Significance tests of a few
// CA-Density classification rules
//
// These are approximate values with a conservative round up
//
// 100 iterations = ~8% error
// 500 iterations = ~4% error
// 1000 iterations = ~1% error
// 5000 iterations = ~0.5% error
//
// Also note that for very high scoring individuals, the error is much lower
// (on the order of less than 1%), so excessive testing is unneeded.
//
// The idea of this heuristic is to not test low scoring individuals more than
// needed, while ensuring that accurate comparisons are made between high
// scoring individuals.
//
// Also note that this kind of heuristic should ensure that the error in the
// scoring is much less than the standard deviation of the populations scores.
//
//-----
int get_optimum_num_samples(float prelim_score)
{
    int ret = NUM_SAMPLES_DEFAULT; //safe default

    if ( prelim_score < .50 )
        ret = 100;
    else if ( prelim_score < .75 )
        ret = 250;
    else if ( prelim_score < .85 )

```

```

        ret = 500;
    else if ( prelim_score < .90 )
        ret = 750;
    else if ( prelim_score < .95 )
        ret = 1000;
    else
        ret = 2500;

    return(ret);
}

//note the Rice fuzzy library returns zero membership values for calls with
//vals above or below limit like this
// ftriangle(13, 6, 12, 12);
//so make sure that the upper & lower membership functions 'span' the range of
//valid values!
void shi_fuzzy_adapt(float * pcross, float * pmut, float bf, float vf, int uf)
{
    int fz_bf = 0;
    int fz_vf = 0;
    int fz_uf = 0;
    int fz_mr = 0;
    int fz_cr = 0;
    float temp1 = 0;
    float temp2 = 0;

    //get fuzzy values

    //best fitness
    //adjusted/scaled to .5 <-> 1.0
    temp1 = ftriangle(bf, 0.0, 0.5, 0.7);
    fz_bf = FUZZY_LOW;
    temp2 = ftriangle(bf, 0.65, 0.775, 0.9);
    if(temp2 > temp1)
    {
        fz_bf = FUZZY_MED;
        temp1 = temp2;
    }
    temp2 = ftriangle(bf, 0.85, 1.0, 1.0);
    if(temp2 > temp1)
        fz_bf = FUZZY_HIGH;

    //variance of fitness -
    //adjusted for empirical values observed
    temp1 = ftriangle(vf, 0.0, 0.0, 0.12);
    fz_vf = FUZZY_LOW;
    temp2 = ftriangle(vf, 0.1, 0.16, 0.22);
    if(temp2 > temp1)
    {
        fz_vf = FUZZY_MED;
        temp1 = temp2;
    }
    temp2 = ftriangle(vf, 0.20, .32, 1);
    if(temp2 > temp1)
        fz_vf = FUZZY_HIGH;

    //unchanged fitness
    temp1 = ftriangle(uf, 0.0, 0.0, 6);
    fz_uf = FUZZY_LOW;
    temp2 = ftriangle(uf, 3, 6, 9);
    if(temp2 > temp1)
    {
        fz_uf = FUZZY_MED;
        temp1 = temp2;
    }
    temp2 = ftriangle(uf, 6, 12, NUM_GENERATIONS);
    if(temp2 > temp1)
        fz_uf = FUZZY_HIGH;

    //fuzzy rule base
    if (fz_bf == FUZZY_LOW)
    {
        fz_mr = FUZZY_MED;
        fz_cr = FUZZY_HIGH;
    }
    if ((fz_bf == FUZZY_MED) && (fz_uf == FUZZY_LOW))
    {
        fz_mr = FUZZY_LOW;
        fz_cr = FUZZY_HIGH;
    }
}

```

```

if ((fz_bf == FUZZY_MED) && (fz_uf == FUZZY_MED))
{
    fz_mr = FUZZY_MED;
    fz_cr = FUZZY_MED;
}
if ((fz_uf == FUZZY_HIGH) && (fz_vf == FUZZY_MED))
{
    fz_mr = FUZZY_HIGH;
    fz_cr = FUZZY_LOW;
}
if ((fz_bf == FUZZY_HIGH) && (fz_uf == FUZZY_LOW))
{
    fz_mr = FUZZY_LOW;
    fz_cr = FUZZY_HIGH;
}
if ((fz_bf == FUZZY_HIGH) && (fz_uf == FUZZY_MED))
{
    fz_mr = FUZZY_MED;
    fz_cr = FUZZY_MED;
}
if ((fz_uf == FUZZY_HIGH) && (fz_vf == FUZZY_LOW))
{
    fz_mr = FUZZY_HIGH;
    fz_cr = FUZZY_LOW;
}
if ((fz_uf == FUZZY_HIGH) && (fz_vf == FUZZY_HIGH))
{
    fz_mr = FUZZY_LOW;
    fz_cr = FUZZY_LOW;
}

//defuzzify output parameters

//mutation rate
//altered to pick MED as 0.03 (original EvCA value)
//not symmetric!
if (fz_mr == FUZZY_LOW)
    *pmut = 0.03;
else if (fz_mr == FUZZY_MED)
    *pmut = 0.10;
else if (fz_mr == FUZZY_HIGH)
    *pmut = 0.16;

//crossover rate
//altered to pick HIGH as 1.00 (original EvCA value)
//not symmetric! and more conservative than SHI original values
if (fz_cr == FUZZY_LOW)
    *pcross = 0.90;
else if (fz_cr == FUZZY_MED)
    *pcross = 0.95;
else if (fz_cr == FUZZY_HIGH)
    *pcross = 1.00;

cout << "bf:" << bf << " [fz_bf:" << fz_bf << "]" << endl;
cout << "vf:" << vf << " [fz_vf:" << fz_vf << "]" << endl;
cout << "uf:" << uf << " [fz_uf:" << fz_uf << "]" << endl;
cout << "fz_mr:" << fz_mr << " pmut:" << *pmut << endl;
cout << "fz_cr:" << fz_cr << " pcross:" << *pcross << endl;

}

```



```
/* general norm operators */
float *fnorm(int len, float *dest, float *src, float (*norm)(float, float));
float *fxnorm(int len, float *dest, float *src,
              float (*norm)(float, float, float), float par);

/* intersection operators */
float *fzand(int len, float *dest, float *src);
float *fland(int len, float *dest, float *src);
float *fpand(int len, float *dest, float *src);

/* union operators */
float *fzor(int len, float *dest, float *src);
float *flor(int len, float *dest, float *src);
float *fpor(int len, float *dest, float *src);

/* negation operator */
float *fnot(int len, float *dest);

/* height */
float fhgt(int len, float *src);

/* alpha-cut and strong alpha-cut */
float *fcut(int len, float *dest, float *src, float cut);
float *fscut(int len, float *dest, float *src, float cut);

#endif /* _FUZZY_H_ */
```



```

    return 0.0;
}

/* hard membership function(s) */
float ftrapezium(float val, float p1, float p2, float p3, float p4)
{
    if(p2 <= val && val <= p3)
        return 1.0;
    if(val <= p1 || p4 <= val)
        return 0.0;
    if(p1 < val && val < p2)
        return (val - p1)/(p2 - p1);
    if(p3 < val && val < p4)
        return (p4 - val)/(p4 - p3);

    return 0.0;
}

/* indexed-centre-of-gravity */
float ficog(int len, float *set, float lim)
{
    register int i;
    float up = 0.0, low = 0.0;

    for(i = 1; i <= len; set++, i++)
        if(*set >= lim)
            {
                up += *set*i;
                low += *set;
            }

    up -= 1.0;

    if(low == 0.0)
        return -1.0;

    return up/low;
}

/* centre-of-gravity */
float fcog(int len, float *set)
{
    register int i;
    float up = 0.0, low = 0.0;

    for(i = 1; i <= len; set++, i++)
        {
            up += *set*i;
            low += *set;
        }

    up -= 1.0;

    if(low == 0.0)
        return -1.0;

    return up/low;
}

/* mean-of-maximum */
float fmom(int len, float *set)
{
    int low = 0;
    float up = 0.0, top;

    top = fhgt(len, set);
    set += len;

    while(len)
        {
            if(*set == top)
                {
                    up += len;
                }
        }
}

```

```

        low++;
    }
    set--;
    len--;
}

up -= 1.0;

if(low == 0)
    return -1.0;

return up/low;
}

/* general norm operators */

float *fnorm(int len, float *dest, float *src, float (*norm)(float, float))
{
    float *save = dest;

    if(len < 0)
        while(len++)
        {
            *dest = (*norm)(*dest, *src);
            dest++;
        }
    else
        while(len--)
        {
            *dest = (*norm)(*dest, *src);
            dest++;
            src++;
        }

    return save;
}

float *fxnorm(int len, float *dest, float *src,
              float (*norm)(float, float, float), float par)
{
    float *save = dest;

    if(len < 0)
        while(len++)
        {
            *dest = (*norm)(*dest, *src, par);
            dest++;
        }
    else
        while(len--)
        {
            *dest = (*norm)(*dest, *src, par);
            dest++;
            src++;
        }

    return save;
}

/* intersection operators */

float *fzand(int len, float *dest, float *src)
{
    float *save = dest;

    if(len < 0)
        while(len++)
        {
            *dest = MIN(*dest, *src);
            dest++;
        }
    else
        while(len--)
        {
            *dest = MIN(*dest, *src);
            dest++;
            src++;
        }
}

```

```

    return save;
}

float *fpand(int len, float *dest, float *src)
{
    float *save = dest;

    if(len < 0)
        while(len++)
        {
            *dest = *dest * *src;
            dest++;
        }
    else
        while(len--)
        {
            *dest = *dest * *src;
            dest++;
            src++;
        }

    return save;
}

float *fland(int len, float *dest, float *src)
{
    float *save = dest;

    if(len < 0)
        while(len++)
        {
            *dest = MAX(*dest + *src - 1.0, 0.0);
            dest++;
        }
    else
        while(len--)
        {
            *dest = MAX(*dest + *src - 1.0, 0.0);
            dest++;
            src++;
        }

    return save;
}

/* union operators */

float *fzor(int len, float *dest, float *src)
{
    float *save = dest;

    if(len < 0)
        while(len++)
        {
            *dest = MAX(*dest, *src);
            dest++;
        }
    else
        while(len--)
        {
            *dest = MAX(*dest, *src);
            dest++;
            src++;
        }

    return save;
}

float *fpor(int len, float *dest, float *src)
{
    float *save = dest;

    if(len < 0)
        while(len++)
        {
            *dest = *dest + *src - *dest * *src;
            dest++;
        }
}

```

```

    else
        while(len--)
        {
            *dest = *dest + *src - *dest * *src;
            dest++;
            src++;
        }
    return save;
}

float *flor(int len, float *dest, float *src)
{
    float *save = dest;

    if(len < 0)
        while(len++)
        {
            *dest = MIN(*dest + *src, 1.0);
            dest++;
        }
    else
        while(len--)
        {
            *dest = MIN(*dest + *src, 1.0);
            dest++;
            src++;
        }
    return save;
}

/* negation operator */
float *fnot(int len, float *dest)
{
    float *save = dest;

    while(len--)
    {
        *dest = 1.0 - *dest;
        dest++;
    }

    return save;
}

/* height */
float fhgt(int len, float *set)
{
    float value = 0.0;

    while(len--)
    {
        value = MAX(value, *set);
        set++;
    }

    return value;
}

/* alpha-cut and strong alpha-cut */
float *fcut(int len, float *dest, float *src, float cut)
{
    float *save = dest;

    while(len--)
    {
        *dest = (*src >= cut) ? *src : 0.0;
        dest++;
        src++;
    }

    return save;
}

```

```
float *fscut(int len, float *dest, float *src, float cut)
{
    float *save = dest;

    while(len--)
    {
        *dest = (*src > cut) ? *src : 0.0;
        dest++;
        src++;
    }

    return save;
}

#ifdef TEST_FUZZY
#include <stdio.h>

int main()
{
    float val;
    val = fpi(0.1, 1.0, 2.0, 3.0, 4.0);
    printf("val = %f\n", val);
    val = fpi(1.1, 1.0, 2.0, 3.0, 4.0);
    printf("val = %f\n", val);
    val = fpi(1.6, 1.0, 2.0, 3.0, 4.0);
    printf("val = %f\n", val);
    val = fpi(2.1, 1.0, 2.0, 3.0, 4.0);
    printf("val = %f\n", val);
    val = fpi(3.1, 1.0, 2.0, 3.0, 4.0);
    printf("val = %f\n", val);
    val = fpi(3.6, 1.0, 2.0, 3.0, 4.0);
    printf("val = %f\n", val);
    val = fpi(4.1, 1.0, 2.0, 3.0, 4.0);
    printf("val = %f\n", val);
    return 0;
}

#endif
```

References

- [1] M. Capcarrere, M. Tomassini, A. Tettamanzi, and M. Sipper, "A statistical study of a class of cellular evolutionary algorithms," *Evolutionary Computation*, vol. 7, no. 3, pp. 255-274, 1999.
- [2] D. Andre, F.H. Bennet, and J.R. Koza, "Evolution of intricate long-distance communication signals in cellular automata using genetic programming," in *Proc. of Artificial Life V*, 1996, pp. 513-520.
- [3] J. Paredis, "Coevolving cellular automata: be aware of the red queen!," in *Proc. of Seventh Int. Conf. on Genetic Algorithms (ICGA97)*, 1997, pp. 393-400.
- [4] H. Juillé and J.B. Pollack, "Coevolving the 'ideal' trainer: Application to the discovery of cellular automata rules," in *Proc. Third Ann. Conf. Genetic Programming*, 1998, pp. 519-527.
- [5] S. Wolfram, *A New Kind of Science*, Champaign, IL: Wolfram Media, 2002.
- [6] K. A. DeJong, "An analysis of the behavior of a class of genetic adaptive systems," *Ph.D Dissertation, University of Michigan*, University Microfilms No. 68-7556. 1975.
- [7] Y. Shi, R. Eberhart, and Y. Chen, "Implementation of evolutionary fuzzy systems," *IEEE Transactions on Fuzzy Systems*, vol. 7, no. 2, 1999, pp. 109-119.
- [8] M. Nowostawski and R. Poli, "Parallel genetic algorithm taxonomy," in *Proc. of the Third Int. Conf. on Knowledge-based Intelligent Information Engineering Systems (KES'99)*, 1993, pp. 93-98.

APPENDIX B. SECONDARY REFERENCES

Note: The following appendix contains references gathered during research that relate to adaptive genetic algorithms. These references were not used in this thesis. They may prove to be a useful bibliography for further study by the reader.

Secondary References

- [1] A. Bergman, W. Burgard, and A. Hemker, "Adjusting parameters of genetic algorithms by fuzzy control rules," *New Computer Techniques in Physics Research III* (Becks K. H. and Gallix D. P. Eds.), Singapore: World Scientific Press, 1994, pp 235-240.
- [2] A. Di Nola and V. Loia, "Fuzzy evolutionary framework for adaptive agents," in *1999 Proc. of the ACM Symposium on Applied Computing*, 1999, pp. 233-237.
- [3] A.G. Tettamanzi, "Evolutionary algorithms and fuzzy logic: A two integration," in *Proc. of Second Joint Conf. on Information Sciences*, 1995, pp. 464-467.
- [4] B. Freisleben and S. Strelen "Hybrid genetic algorithm/fuzzy logic approach to manufacturing process control," in *1995 Proc. of the IEEE Conf. on Evolutionary Computation*, 1995, pp. 837-841.
- [5] B. Kosko, *Neural Networks and Fuzzy Systems*. New York: Prentice-Hall. 1992.
- [6] C. Karr, "Design of an adaptive fuzzy logic controller using a genetic algorithm," in *Proc. of the Fourth Int. Conf. on Genetic Algorithms*, 1991, pp. 450-457.
- [7] C.L. Karr and E.J. Gentry, "Application of fuzzy control techniques to a chaotic system," *1993 Symposium on Emerging Computer Techniques for the Minerals Industry*, 1993, pp. 371-376.
- [8] D. Bhandari, S.K. Pal, and M.K Kundu, "Image enhancement incorporating fuzzy fitness function in genetic algorithms," in *1993 Proc. IEEE Int. Conf. on Fuzzy Systems*, 1993, pp. 1408-1413.
- [9] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. New York: Morgan Kaufmann. 1988.
- [10] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," *Foundations of Genetic Algorithms (FOGA '90)* (G. Rawlins, Ed.), New York: Morgan Kaufmann, 1991, pp 69-93.
- [11] D. Srinivasan and A. Tettamanzi, "Heuristics-guided evolutionary approach to multiobjective generation scheduling," *IEEE Proc. Generation, Transmission and Distribution*, vol. 143, pp. 553-559. 1996.
- [12] D. Whitley, "The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best," in *Proc. Of the Third Int. Conf. on Genetic Algorithms (ICGA '89)*. 1989, pp. 116-123.
- [13] D. E. Goldberg, K. Deb, and J. H. Clark, "Genetic algorithms, noise and the sizing of populations," *Complex Systems*, vol. 6, pp. 333-362, 1992.

- [14] E.P. Dadios and D.J. Williams, "Fuzzy-genetic controller for the flexible pole-cart balancing problem," in *1996 Proc. of the IEEE Conf. on Evolutionary Computation*, 1996, pp. 223-228.
- [15] F. Herrera, M. Lozano, and J.L. Verdegay. "Tackling fuzzy genetic algorithms," *Genetic Algorithms in Engineering and Computer Science*, (G. Winter, J. Periaux, M. Galan, P. Cuesta Eds.), New York: John Wiley, 1995, pp. 167-189.
- [16] G. Klir and T. Folger, *Fuzzy Sets, Uncertainty, and Information*. New York: Prentice Hall, 1988.
- [17] G.S.K. Fung, J.N.K. Liu, K.H. Chan, and R.W.H. Lau, "Fuzzy genetic algorithm approach to feature selection problem," in *1997 IEEE Int. Conf. on Fuzzy Systems*, 1997, pp. 441-446.
- [18] H. Bersini, G. Seront, "In search of a good evolution-optimization crossover," *Parallel Problem Solving from Nature II*, (R. Männer and B. Manderick, Eds.), New York: Elsevier, 1992, pp. 479-488.
- [19] H.Y. Xu and G. Vukovich, "Fuzzy evolutionary algorithms and automatic robot trajectory generation," in *1994 Proc. IEEE Conf. on Evolutionary Computation*, 1994, pp. 595-600.
- [20] H.-M. Voigt and H. Muehlenbein, "Gene pool recombination and utilization of covariances for the breeder genetic algorithm," in *1995 Proc. of the IEEE Conf. on Evolutionary Computation*, 1995, pp. 172-177.
- [21] J. D. Shaffer and A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms," in *Proc. of the Second Int. Conf. on Genetic Algorithms (ICGA'87)*, 1987, pp. 36-40, 1987.
- [22] J. Eshelman and J.D. Schaffer, "Preventing premature convergence in genetic algorithms by preventing incest," in *Proc. of the Fourth Int. Conf. on Genetic Algorithms (ICGA'91)*, 1991. pp. 115-122.
- [23] J. E. Baker, "Adaptive selection method for genetic algorithms," in *Proc. of an Int. Conf. on Genetic Algorithms (ICGA'85)*, 1985, pp. 101-111.
- [24] J. H. Holland, *Adaptation in Natural and Artificial Systems (second edition)*. Cambridge, MA: MIT Press, 1992.
- [25] J. Yen, B. Lee, and J.C. Liao, "Using fuzzy logic and a hybrid genetic algorithm for metabolic modeling," *AAAI/IAAI*, vol. 1, pp. 743-749, 1996.
- [26] J.D. Shaffer and A. Morishima, "An adaptive crossover distribution mechanism for genetic algorithms," in *Proc. of the Second Int. Conf. of Genetic Algorithms*,

(ICGA '87), 1987, pp. 36-40.

- [27] J.J Grefenstette, "Optimization of control parameters for genetic algorithms," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-16, pp. 122-128, 1986.
- [28] M.A. Lee and H. Takagi, "Embedding apriori knowledge into an integrated fuzzy system design method gased on genetic algorithms," in *Proc. of the Fifth Int. Fuzzy Systems Association Congress, (IFSA '93)*, 1993, pp. 1293-1296.
- [29] P. Thrift, "Fuzzy logic synthesis with genetic algorithms," in *Proc. Fourth Int. Conf. on Genetic Algorithms (ICGA'91)*, 1991, pp. 509-513.
- [30] R. Jang, "Fuzzy controller design without domain experts," in *Proc. IEEE Int. Conf. on Fuzzy Systems (FUZZ-IEEE'92)*, 1992, pp. 289-296.
- [31] R. Jang, "Self-learning fuzzy controller based on temporal back propagation," *IEEE Trans. on Neural Networks*, vol 3, no. 5, pp. 714-723, 1992.
- [32] R. Subbu, A. Sanderson, and P. Bonissone, "Fuzzy logic controlled genetic algorithms versus tuned genetic algorithms: An agile manufacturing application," in *1998 Proc. IEEE Int. Symposium on Intelligent Control*, 1993, pp. 434-440.
- [33] S. McClintock, T. Lunney, and A. Hashim, "Fuzzy logic controlled genetic algorithm environment," *1997 Proc. of the IEEE Int. Conf. on Systems, Man and Cybernetics*, 1997, pp. 2181-2186.
- [34] S. McClintock, T. Lunney, and A. Hashim, "Using fuzzy logic to optimize genetic algorithm performance," in *1997 Proc. IEEE Int. Conf. on Intelligent Engineering Systems*, 1997, pp. 271-275.
- [35] S.K. Pal and D. Bhandari, "Genetic algorithms with fuzzy fitness function for object extraction using cellular networks," *Fuzzy Sets and Systems*, vol. 65, no. 2-3, pp. 129-139, 1994.
- [36] T. Back, "Optimal mutation rates in genetic search," in *Proc. Of the Fifth Int. Conf. on Genetic Algorithms (ICGA-93)*, 1993. pp. 2-8.
- [37] T. C. Fogarty, "Varying the probability of mutation in the genetic algorithm," in *Proc. of the Third Int. Conf. on Genetic Algorithms (ICGA'89)*, 1989, pp. 104-109.
- [38] T.J. Ross. *Fuzzy Logic with Engineering Applications*. New York: McGraw-Hill, 1995.
- [39] Y.H. Song, G.S. Wang, Q Xia, and A.T. Johns, " Fuzzy logic controlled genetic algorithms," in *1996 IEEE Int. Conf. on Fuzzy Systems*, 1996, pp. 972-979.

- [40] Y.H. Song, G.S. Wang, Q Xia, and A.T. Johns, "Environmental/economic dispatch using fuzzy logic controlled genetic algorithms," *IEEE Trans. on Generation, Transmission and Distribution*, vol. 144 no. 4, pp. 377-382, 1997.
- [41] Y.H. Song, G.S. Wang, Q Xia, and A.T. Johns, "Improved genetic algorithms with fuzzy logic controlled crossover and mutation," *Proc. of the 1996 UKACC Int. Conf. on Control*, 1996, pp. 140-144.

APPENDIX C. COPYRIGHT RELEASES

ASME Press Copyright Release Letter

Subject: Re: Copyright Release Form
Date: Tue, 03 Dec 2002 13:19:44 -0500
From: Beth Darchi <darchib@asme.org>
To: jnr@cc.usu.edu

Dear Mr. Richter:

It is our pleasure to grant you permission to use the following ASME paper "Fuzzy Evolutionary Cellular Automata", by J Neal Richter & David Peak, Proceedings of ANNIE-2002, Intelligent Engineering Systems Through Artificial Neural Networks, Volume 12, ISBN: 0-7918-0191-8 ASME PRESS 2002, cited in your letter for use as part of a Master's Thesis at Utah State University.

As is customary, we ask that you ensure full acknowledgment of this material, the author(s), and ASME as original publisher on all printed copies being distributed.

Many thanks for your interest in ASME publications.

Sincerely,

Beth Darchi
Technical Publishing
ASME International
Three Park Avenue, 22S1
New York, NY 10016-5990
tel: (212) 591-7700
fax: (212) 591-7292
DarchiB@asme.org

Copyright Release Form from David Peak

I, David Peak, hereby give James Neal Richter permission to use the following information in his Master's thesis at Utah State University:

"Fuzzy Evolutionary Cellular Automata," by J Neal Richter & David Peak, Proceedings of ANNIE-2002, Intelligent Engineering Systems Through Artificial Neural Networks, Volume 12, ISBN: 0-7918-0191-8 ASME PRESS 2002.

This permission is granted with the understanding that I will receive full bibliographic credit for all material used.

Signed _____

Date _____