

**Genetic Algorithms
and
Fitness Variance
with an
Application to the Automated Design
of Artificial Neural Networks**

W. Michael Rudnick
Bachelor of Science — Mathematics
Portland State University
1972

A dissertation submitted to the faculty of the
Oregon Graduate Institute of Science & Technology
in partial fulfillment of the
requirements for the degree
Doctor of Philosophy
in
Computer Science and Engineering

April 1992

© Copyright 1992 by W. Michael Rudnick
All Rights Reserved

The dissertation "Genetic Algorithms and Fitness Variance with an Application to the Automated Design of Artificial Neural Networks" by W. Michael Rudnick has been examined and approved by the following Examination Committee:

Robert G. Babb II
Associate Professor
Thesis Research Co-Advisor

David E. Goldberg
Associate Professor
University of Illinois at Urbana-Champaign
Thesis Research Co-Advisor

Dan Hammerstrom
Associate Professor

Todd Leen
Assistant Professor

Kent Spackman
Oregon Health Sciences University

Acronyms

Acronym	Meaning	First Use
GA	genetic algorithm	1
SNR	signal-to-noise ratio	34
END	evolutionary network design	66
ANN	artificial neural network	66
GAND	genetic algorithms for network design	66

Notation

Notation	Meaning	First Use
i, j, k	index variables	—
t	time	11
\mathbf{x}	a genotype	4
x_i	i^{th} most significant bit position	4
f or $f(\mathbf{x})$	fitness of genotype \mathbf{x}	4
k	arity of representation alphabet	7
l	length of genotype or representation	4
\mathbf{h}	a schema	8
h	a schema template	8
$*$	wild card (don't care) schema template character	8
$o(\mathbf{h})$	order of schema \mathbf{h}	8
$o(\mathbf{J})$	order of competition partition \mathbf{J}	10
$s(h)$	schema generated from schema template h	8
$\delta(\mathbf{h})$	defining length of schema \mathbf{h}	8
\mathbf{J}	a competition partition	10
J	competition partition number	10
J_i	competition partition index set	10
$N(\mathbf{x})$	number of copies of genotype \mathbf{x} in population	11
$f(\mathbf{h})$	schema \mathbf{h} 's fitness	11
$f(\mathbf{J})$	partition \mathbf{J} 's fitness	11
$P(\mathbf{h})$	proportion of population in schema \mathbf{h}	11
n	population size	11
\bar{f}	average (or expected) fitness of population	11
G	the search space, or space of all genotypes	11
$P(\mathbf{h}, t)$	proportion of population in schema \mathbf{h} at time t	12
$E(m)$	expectation of m	12
p_c	crossover probability	12

Notation	Meaning	First Use
p_m	mutation probability	12
$\psi_j(\mathbf{x})$	Walsh function j of \mathbf{x}	12
$\psi_j(\mathbf{h})$	Walsh function j of schema \mathbf{h}	12
$\psi_{j,k}(\mathbf{h})$	two-dimensional Walsh function j, k of schema \mathbf{h}	12
$\beta(i)$	function converting *s to 0s	12
w_j	Walsh coefficient j	13
$ \mathbf{h} $ or $ \mathbf{J} $	size of corresponding set	13
$J_n(\mathbf{h})$	number of partition containing \mathbf{h}	14
X	random variable	18
S	sample space of random variable	18
\bar{X}	expected or average value of X	18
$P(X)$	probability density function of X	18
$\text{var}()$	variance function	18
(j, k)	index pair	19
$J_i^2(\mathbf{h})$	Cartesian cross product of $J_i(\mathbf{h})$	19
$S(\mathbf{h}, j, k)$	Walsh function summation	20
$J_{\oplus}^2(\mathbf{h})$	index set for nonzero terms in $S(\mathbf{h}, j, k)$	21
w_1'	order-one Walsh terms of equal value	28
α	significance level	28
z	standardized normal distribution random variable	29
μ	mean	29
k	a schema's order	30
$\hat{f}(\mathbf{h})$	sample mean fitness of schema \mathbf{h}	30
S	a sampling distribution	30
σ^2	variance	30
M	twice schema's order	30
$\overline{J_i(\mathbf{J})}$	complement of set $J_i(\mathbf{J})$	38
$S(\mathbf{J})$	competition partition's signal	35
$C(\mathbf{J})$	competition partition's noise	38
$R(\mathbf{J})$	competition partition's signal-to-noise ratio	39
a_i	a constant	42
P	proportion of 1s at a locus in a population	53
c	number of loci in converged region	53
p'_m	probability of mutation(s) in converged region	54
P_{ss}	proportion of 1s at steady state	54

Notation	Meaning	First Use
D_i	proportion of tournaments available to be decided at locus i	55
d_i	expected proportion of tournaments decided at locus i	55
P_i	proportion of 1s at locus i	55
$P_{i,t}$	proportion of 1s at locus i at time t	58
W_1	index set of order-one Walsh coefficients	63
$J_S(\mathbf{J})$	signal index set for competition partition \mathbf{J}	63
$J_C(\mathbf{J})$	noise index set for competition partition \mathbf{J}	63
\mathbf{w}	weight vector	68
O	node output	68
n	number of input connections to a node	68
$O(n)$	worst-case computational complexity	71
k	number of clumps of 1s in input field	84
k_0	threshold for number of clumps of 1s in input field	84
I	input vector	84
$\%G$	generalization performance (on test set)	85
$\%L$	performance on training set	85
s	hidden node receptive field size	85
X	sampling distribution of genotype fitnesses	97
n	number of independent trials	97
$f_L(\mathbf{x})$	link-adjusted performance	101
n_L	number of links between input and hidden layers	101
f_{avg}	average fitness of new population	104
f_g	average fitness of the genotype	104
ϕ	growth ratio	108
P_t^*	proportion of the best genotype at time t	108
f_i	a constant fitness value	108
$P_{i,t}$	proportion of genotypes with fitness greater than f_i at time t	108
γ	net growth factor	108
ϵ	schema disruption probability	108
$g(\sigma_f^2)$	fitness function's additive Gaussian noise	109
b	linear function's displacement	109
$f'(\mathbf{x})$	linear fitness function with Gaussian noise	109
$\hat{f}'(\mathbf{h})$	noise-augmented sample mean fitness of \mathbf{h}	109
ρ	ratio of fitness noise variance to Walsh magnitude	110
C_l	convergence level	127
C_i	instantaneous convergence rate	127
$\hat{f}^{(i)}$	i^{th} order approximation to \hat{f}	129
λ	logistic function's exponential scaling factor	131

Dedication

Dedicated to the pursuit of truth,
wherever it may be found.

Acknowledgements

To be human is to be part of the group animal, which is simply to say that no one ever does anything alone. So although the research presented here is my work (except, of course, where noted in the text), it is really a joint accomplishment of a group of people. It is with pleasure that I take this opportunity to express my debt and thanks to those who, in ways large and small, have significantly contributed to both my growth and these results.

First and foremost, I wish to thank Dave Goldberg. His time, trust, tolerance, and financial support of both me and the research presented here have made this dissertation possible. He taught me not only much about genetic algorithms and how to do theoretical analysis, but also life skills such as writing and thinking. His demanding standards and personal example stretched my capacities, resulting in considerable personal growth.

Second, I'd like to thank Dan Hammerstrom. As my advisor during my initial three years at Oregon Graduate Institute (OGI), Dan taught me how to think, how to approach a problem (often the most difficult part), and how to do research. Although my interests eventually diverged from Dan's, I feel he is an outstanding advisor, genuine human being, and exemplary manager. I'd also like to thank Dan for his research assistanceship support during my first three years at OGI.

Third, I thank Robbie Babb, who served as my OGI advisor. His "yes, can do" attitude and ever-present humor helped me when I was feeling overwhelmed by the many potential obstacles in pursuing an 'outside' dissertation research path.

Each of these three people played a key role and were necessary to the successful completion of my Ph.D. and this dissertation. I am in your debt. It reminds me of the man stuck on the interstate with a flat tire and no jack — let's say a tall man with a

shaggy beard. After trying to flag down help for some time, someone finally stops and loans him a jack. After the spare is in place and the jack returned, the erstwhile-stranded man says, "I'd like to pay you something for the help," to which the Good Samaritan responds, "Just pass it along." I intend to pass along the help you've all given me.

In addition to my three key helpers, I'd like to thank the others who have given me significant help. Todd Leen gave me a new perspective on what mathematical analysis is about, and worked with me during my difficult "searching for a dissertation subject" period.

I especially thank my friends Kalyanmoy Deb, Dirk Thiersen, and Rob Smith, for their keen insights during our lengthy technical discussions and for their emotional support. I also thank Rob Smith separately for his system administrative help at the University of Alabama, Tuscaloosa. I thank my friends Andrew Horner and Andy Assad for their friendship and emotional support during my stay in Illinois.

I thank Marie La Bonte, Kelly Atkinson, Jo Ann Binkerd, Karon Ticknor, Anita Creche, Patty Stewart, Kelsey Milman, Dwight Todd, Sydney Cromwell, and Vince Weatherill for their secretarial and administrative help and support. Phil Barrett also helped me repeatedly in his capacity as OGI/CSE departmental administrative assistant.

Until I had the opportunity to experience technical libraries at other graduate schools, I didn't understand how truly exceptional the Oregon Graduate Institute library staff is. It is with considerable pleasure I acknowledge the extensive help I received from OGI's library staff, and especially Maureen Sloan, Gretta Siegel, Julianne Williams, Chris Lightcap, Kris Roley, and Mary Vatne.

Kevin Carmody, Nick Nafpliotis, Nick Horton, Marion Hakanson, Bruce Jerrick, John Pochmara, Bob Shair, Ahmed Kassem, and Randy Cetin gave me various and substantial systems administration help.

I thank Dick Kieburtz, Dan Hammerstrom, and the rest of the Oregon Graduate Institute Computer Science and Engineering faculty, and especially Robbie Babb as my OGI advisor, for giving me — despite some initial doubts and skepticism — their permission, trust, and support (both financial and moral) to work with someone outside

the department when my interests led me into an area in which OGI didn't have in-house expertise. I believe their willingness to experiment by allowing me to pursue a dissertation research path outside the customary route speaks well of their innovative spirit.

I'd like to thank those upon whose work my work is based, especially Albert Bethke, whose Walsh schema work established the foundation upon which I build. Every step taken up the path of human progress is taken from the shoulders of those who have gone before (who said this?). I also thank Scott Fahlmam for sharing his Quickprop code and Terry Rieger for sharing his C version of Quickprop.

I acknowledge support from National Science Foundation grants CTS-8451610 and ECS-9022007, U.S. Army Contract DASG60-90-C-0153, and departmental support from Oregon Graduate Institute's Department of Computer Science and Engineering.

Last, and most importantly, I want to thank and acknowledge those closest to me for their ever-constant, loving and emotional (and sometimes financial) support — my parents, Bill and Janet; my sister, Linda; her husband and my good friend, Mike Free; my sister Roberta; and finally, Emma Carter, my dearest friend.

Mike Rudnick
Portland, Oregon
January, 1992

Contents

Acronyms	iv
Notation	v
Dedication	viii
Acknowledgements	ix
Abstract	xviii
1 Introduction	1
1.1 The Simple GA	2
1.2 GA Theory Objects	7
1.3 Schema Theorem	11
1.4 Walsh-Schema Transform	12
1.5 A Note on GA Convergence	16
2 Variance in Genetic Algorithms	17
2.1 Computing Schema Fitness Variance	17
2.2 Applications and Extensions	26
2.3 Fitness Variance Based Population Sizing	27
3 Signal Versus Noise	34
3.1 Overview of GA Signal and Noise	34
3.2 Signal	35
3.3 Noise	38
3.4 Signal-to-Noise Ratio (SNR)	39
3.5 Discussion	40

4	Domino Convergence	42
4.1	Simulation	42
4.2	Analysis of Convergence Window Width	47
4.3	Analysis of Convergence Stall	52
4.3.1	Simple Model	52
4.3.2	Refined and Streamlined Models	55
4.4	Domino Convergence and the Signal-to-Noise Ratio	62
4.5	Mutation and Convergence Stall	65
5	Evolutionary Network Design (END)	66
5.1	Artificial Neural Networks	67
5.2	Network Design Problem	72
5.3	Previous Work	74
5.3.1	Dress's Artificial Insect	75
5.3.2	Mjolsness's Recursive Network Definition	75
5.3.3	Hinton & Nowlan's Work	76
5.3.4	Miller's Connection Matrix	77
5.3.5	Harp's Area Blueprint	79
5.3.6	Kitano's Graph L-System	80
5.3.7	Other Related Work	80
5.4	GAND Overview	81
5.4.1	GAND Design	81
5.4.2	The Test Problem	84
5.4.3	Training and Testing Data Sets	86
5.4.4	Genotype Representation	87
5.5	GAND Results	92
5.5.1	Back-Propagation	95
5.5.2	Objective Function Noise	95
5.5.3	Initial Investigation of Population Size	99
5.5.4	Link Tax	101
5.5.5	Elitism	103
5.6	Population Sizing with Fitness Noise	109
5.7	END Discussion	111
5.7.1	Normal Forms	112
5.7.2	Future GAND Research	118
5.8	GAND Conclusion	121

6	Discussion	122
6.1	SNR, Schema Theorem, and GA Convergence	122
6.2	GA Decision-Making	124
6.3	Future Research	126
6.3.1	Fitness Variance and GA Convergence	127
6.3.2	Other Signal-to-Noise Ratio Extensions	129
6.3.3	Population Sizing	129
6.3.4	Domino Convergence	130
6.4	Conclusion	130
 Appendices		
A	Characterizing Contiguity Problem	131
B	Quickprop versus Back-propagation	133

List of Tables

1.1	A simple GA run.	5
1.2	Schemata and competition partitions for $l = 3$ problems.	9
1.3	Walsh basis schema fitness sums for $l = 3$ problems.	14
2.1	Walsh variance component matrices.	23
2.2	Walsh variance component matrices for $J(*ff)$ and $J(f*f)$, $l = 3$	24
2.3	Walsh variance component matrices for $J(ff*)$, $l = 3$	25
2.4	Normal deviates for various confidence levels.	31
4.1	Two locus selection analysis.	50
4.2	Tournament pairing probabilities.	53
4.3	Example signal, noise, and signal-to-noise ratio values.	64
5.1	Solla's generalization performance results.	86

List of Figures

1.1	Simple GA run — average population fitnesses.	6
2.1	Fitness distributions for two schemata, \mathbf{h}_1 and \mathbf{h}_2	27
2.2	Sampling distribution probability, $P(\hat{f}(\mathbf{h}_{best}) - \hat{f}(\mathbf{h}_{2^{nd}best})$, versus standardized z score.	29
4.1	Convergence for various mutation rates.	44
4.2	Convergence standard deviations for various mutation rates.	45
4.3	Results from a single run.	48
4.4	Loci converge vs time, 90% convergence, various mutation rates.	49
4.5	Context of locus i for simplified convergence stall analysis.	52
4.6	Steady-state convergence level vs locus — empirical and simple model. . .	56
4.7	Locus decision tree — combined model.	59
4.8	Locus decision tree — streamlined model.	60
5.1	Generic artificial neural network node.	69
5.2	GAND design.	82
5.3	Connection matrix for a contiguity problem solution.	85
5.4	General feedforward connection matrix representation.	88
5.5	Single-layer connection matrix representation.	89
5.6	Initial GAND run — generation vs generalization.	91
5.7	GAND performance — baseline.	94
5.8	Solla's vs GAND's back-propagation results.	96
5.9	GAND performance — averaging out noise.	98
5.10	GAND performance — population $N = 150$	100
5.11	GAND performance — link tax.	102
5.12	GAND performance — elitism.	105
5.13	Intermediate population showing elitist “better half” effect.	106
5.14	Typical GAND solution networks.	107
5.15	How two identical networks can produce non-viable offspring.	113

5.16 Left-most normal form representation.	114
5.17 Left-most largest normal form representation.	116
5.18 Left-most single receptive field normal form.	117
B.1 Performance differences — Quickprop vs back-propagation.	134

Abstract

**Genetic Algorithms
and
Fitness Variance
with an
Application to the Automated Design
of Artificial Neural Networks**

**W. Michael Rudnick, Ph.D.
Oregon Graduate Institute of Science & Technology, 1992**

Supervising Professor: Robert G. Babb II

Existing genetic algorithm (GA) theory addresses how schema fitness serves as a measure of the expected increase or decrease of schema representation within the population. The work presented here considers how schema fitness variance affects schema representation through GA decision-making.

It has long been known that the more significant bits of binary coded parameters converge before bits of lesser significance. This phenomenon, called *domino convergence*, is explored using the identity problem, $f(\mathbf{x}) = x$. Sometimes convergence stops prematurely (a phenomenon called *convergence stall*), depending upon the relative magnitude of the mutation rate and the length of the encoding string. Analyses and models are presented exploring various aspects of these phenomena.

GA convergence occurs in competition partitions. Each partition has an associated

signal (measure of the force tending towards correct decision-making within that partition) and *noise* (measure of the force hindering correct decision-making). Which has the upper hand within a particular partition determines if the GA chooses correctly between competing schemata, which in turn determines convergence in the partition. Signal, noise, and the signal-to-noise ratio (SNR) are each defined in terms of fitness variance, with the SNR reconciling the conflicting effects of signal and noise. Formulas for the flat-population schema fitness variance, signal, noise, and SNR are derived using the Walsh basis. Both domino convergence and convergence stall are examined from the signal versus noise perspective.

Designing an artificial neural network (ANN) for a specified problem can be difficult. Since the design of biological neural networks is a result of evolution, evolutionary search techniques may be well suited to network design. Back-propagation is known to generalize well on the contiguity problem (counting the number of clumps of 1s in a binary input field) when hidden layer receptive fields are narrow, but with high performance variance (noise) due to local minima.

Evolutionary network design is used as a case study in applying GAs to a difficult, noisy problem. A program called GAND, genetic algorithms for network design, is described and tested on the contiguity problem. A number of techniques are presented that allow GAND, starting with randomly generated network interconnections, to evolve architectures rivaling the best produced by hand.

Chapter 1

Introduction

The genetic algorithm (GA) is a population-based search technique abstracting the paradigm of natural evolution (Holland, 1975b; Goldberg, 1989c; Davis, 1991; Brady, 1985; Casti & Karlqvist, 1986; Davis, 1987; Grefenstette, 1985; Grefenstette, 1987; Schaffer, 1989; Holland, 1984). Thus, like artificial neural networks, GAs are biologically motivated.

Building blocks, which are short-defining-length, low-order schemata of above-average fitness containing optima or near-optima, are rightly recognized as one of the keys to GA function. If the population is rich in building blocks, crossover can combine them to produce good solutions. Holland's schema theorem (Holland, 1975b; Goldberg, 1989c) provides a lower bound on the expected representation of each schema in the next generation's population based on its representation in the current population, and thereby serves as a measure of when building blocks are likely to grow. However, the schema theorem is only a result in *expectation*. Because of stochastic influences (Jong, 1975; Goldberg & Segrest, 1987), even when its inequality is satisfied the schema theorem does not guarantee a building block will grow. The present work addresses these stochastic deviations by defining schema fitness variance and examining how it affects a schema's increase or decrease in representation.

The dissertation is organized as follows. This chapter introduces the GA through a simple example, establishes notational conventions and definitions, and reviews relevant GA theory, including Walsh functions and the Walsh schema transform. Chapter 2 reviews the GA fitness variance literature and then derives an expression for schema

fitness variance in the Walsh basis. Chapter 3 defines competition partition signal and noise measures. The conjecture is presented that the entity in which convergence occurs in the GA is the competition partition, and that signal and noise together control the order in which partitions converge. The signal-to-noise ratio is then defined as a measure reconciling the antagonistic effects of signal and noise. The Walsh basis expression for schema fitness variance from Chapter 2 is used to derive expressions for signal, noise, and the signal-to-noise ratio. Chapter 4 then defines and demonstrates the phenomena of domino convergence, the convergence window, and convergence stall as signal-to-noise ratio effects. Various analyses of these phenomena are performed. Chapter 5 presents an evolutionary artificial neural network architecture design application. The application is both difficult and possesses a noisy fitness function; thus, both general GA theory and the schema fitness variance work may be usefully employed. Of course, evolutionary network design is also interesting in its own right. Finally, Chapter 6 discusses the work and outlines opportunities for further investigation.

The remainder of this chapter is organized as follows. First the GA is introduced and an example of running the GA on a simple problem is presented. Next, the various objects important to GA theory are defined and notational conventions are established. Then, Walsh functions and the Walsh schema transform are reviewed. Finally, a brief discussion of what constitutes GA convergence is presented.

1.1 The Simple GA

The genetic algorithm is a form of guided blind search. It is guided because a scalar reinforcement signal is used. It is blind because the GA does not access the ‘insides’ of the process producing the reinforcement signal, only the resulting evaluations of utility. In effect, the model being optimized is treated as a black box.

The GA is a population sampling technique. The GA uses the reinforcement signal to guide its population sampling toward more rewarding regions of the search space.

The GA is robust in that it can be applied to any well-defined problem. Of course

as with all search techniques, the match, or lack thereof, between the structure of the search technique and the structure of the objective function determines the potential effectiveness of the search (Ackley, 1987).

It is worth noting that from an algorithmic perspective the GA has near-perfect parallelism in that the evaluation of each candidate solution can be performed in parallel. It is thus an ideal candidate for execution on highly parallel computer architectures. This is especially true when, as in the neural network design problem presented in Chapter 5, the bulk of the computational burden results from the evaluation of each candidate solution.

Using the GA is fairly simple. First, a representation for candidate solutions must be defined, often a binary string encoding. Second, some comparison criteria must be established to evaluate the relative 'goodness' of different candidate solutions. A fitness function inducing a total order on candidate solutions is usually defined for this purpose. Finally, the GA operators and parameters are selected. These include deciding what kind of crossover, mutation, and selection to use; setting their associated parameters such as crossover and mutation rates; and setting the value of other GA parameters such as population size.

The operation of the GA is also simple. First, an initial population of candidate solution strings, or *genotypes*, is generated, often by generating random strings. Then, generation cycles are performed repeatedly until a stopping criteria is reached. Stopping criteria are usually based on a fixed number of generations, a heuristic measure of population status, or a measure of population diversity.

A generation cycle consists of selection, crossover, and mutation. During selection, genotypes having above-average fitness are chosen to be parents. During crossover, the strings for two selected genotypes are combined to produce offspring. During mutation, with small probability each bit position of an offspring genotype is independently flipped.

In the remainder of this section, an easy binary integer problem will be used to illustrate the simple GA. The simple GA may be thought of as 'vanilla' GA — nothing fancy, just good, everyday GA. The simple GA has none of the customized enhancements or

operators that are sometimes beneficial for tackling certain problems. For an introductory, yet fairly comprehensive treatment of the care and feeding of GAs, see Goldberg (1989c).

The problem consists of using a binary string as a coding and then valuing the goodness of a solution by interpreting a binary string as an integer. Thus, its fitness function is

$$f(\mathbf{x}) = \sum_{i=0}^{l-1} 2^i x_i, \quad (1.1)$$

where x_i denotes the value of the i^{th} most significant bit position of genotype \mathbf{x} and l is the length of the representation in bits. A genotype string length of three and population size of six will be used to keep this example short.

Table 1.1 shows the resulting GA run of 10 generations plus the initial population of randomly generated genotypes. Column one shows the generation number, where generation zero is the random population. Column two allows referring to individual genotypes in the population by providing each with a number unique within the generation. Column three shows each genotype's fitness. Columns four, five, and six show respectively, for each genotype, its number of mutations during the current generation, its crossover partner (if it mated), and its parent in the preceding generation. Finally, column seven shows each genotype as a binary string.

First examine generation zero, the random population. Since selection, crossover, or mutation are not done on the seed population, the corresponding column entries are empty. Note genotype three has the highest fitness in the initial population. We were slightly unlucky in this, since for our simple, length-three problem, the probability that a perfect individual (genotype 111 has the highest possible fitness for our three bit problem) will occur in a randomly selected population of size six is 0.55. Figure 1.1 shows average population fitness at each generation. The expected average fitness for a randomly generated population is 3.5; thus the 3.67 actually obtained is close to the expected value.

Tournament selection is used. The idea is that candidate solutions in the current

gen	I Num	$f(x)$	m	c	p	string
0	0	5.0				101
	1	2.0				010
	2	2.0				010
	3	6.0				110
	4	3.0				011
	5	4.0				100
gen	I Num	$f(x)$	m	c	p	string
1	0	6.0		3	5	110
	1	4.0		5	3	100
	2	6.0		3	0	110
	3	1.0	1	0	3	001
	4	7.0	1		0	111
	5	5.0	1		5	101
2	0	7.0		3	1	111
	1	6.0		1	3	110
	2	5.0		5	5	101
	3	5.0		5	5	101
	4	6.0		3	1	110
	5	7.0		1	3	111
3	0	7.0		3	1	111
	1	6.0		1	3	110
	2	5.0		5	5	101
	3	5.0		5	5	101
	4	6.0		3	1	110
	5	7.0		1	3	111
4	0	1.0	1		2	001
	1	7.0			5	111
	2	7.0		0		111
	3	5.0	1	0		101
	4	6.0			1	110
	5	7.0			5	111
5	0	7.0		1	3	111
	1	1.0	1	3	1	001
	2	5.0		3	5	101
	3	7.0		5	3	111
	4	7.0		2	5	111
	5	7.0		5	2	111
6	0	7.0		5	0	111
	1	7.0		0	5	111
	2	6.0	1	5	2	110
	3	7.0		2	5	111
	4	5.0			2	110
	5	7.0			0	111
7	0	7.0		5	0	111
	1	7.0		0	5	111
	2	6.0	1	5	2	110
	3	7.0		2	5	111
	4	6.0	1		2	110
	5	7.0			0	111
8	0	6.0		2		110
	1	7.0		3		111
	2	7.0		0	0	111
	3	7.0		0	0	111
	4	7.0		5	3	111
	5	7.0		3	5	111
9	0	7.0		3	5	111
	1	7.0		5	3	111
	2	7.0		1	1	111
	3	7.0		1	1	111
	4	7.0		5	2	111
	5	5.0	1	2	5	101
10	0	7.0		0	3	111
	1	7.0		3	0	111
	2	7.0		4	2	111
	3	7.0		2	4	111
	4	7.0		2	0	111
	5	7.0		0	2	111

Table 1.1: Generations zero through ten of the simple GA on the binary integer problem.

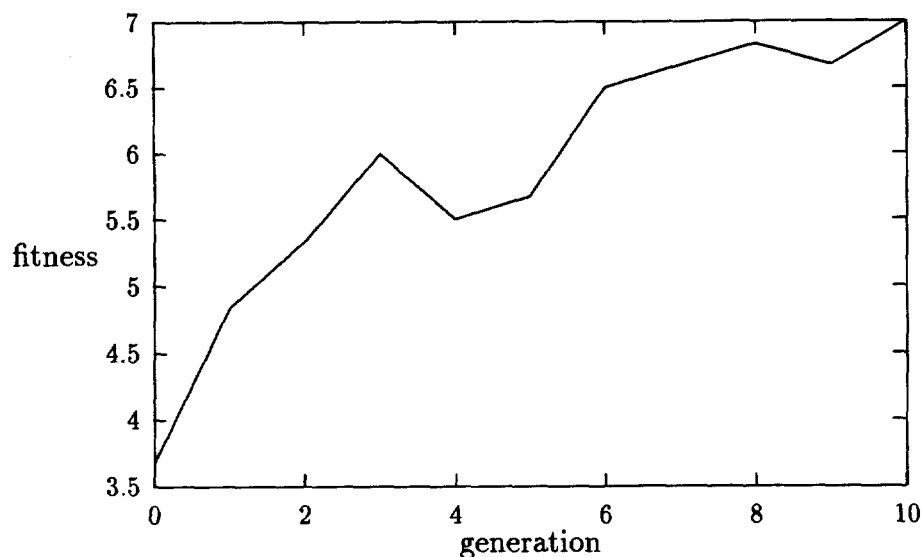


Figure 1.1: Generation number versus average population fitness.

population engage in tournaments with each other. The winner of each tournament is the candidate solution with the higher fitness. Its genotype is copied into the next generation's population, where it undergoes crossover and mutation. Consider the population at generation 1. From column 'p' we can see genotype numbers 0, 3, and 5 from the previous generation each won two tournaments. Such a tournament selection outcome is sensible, since they are the three best genotypes in the previous generation.

Two-point crossover occurs with probability 0.75. Two positions are selected at random, and the string segments between them are exchanged. Again, consider generation 1. For each member of the population, column 'c' shows the crossover partner's number from the previous generation. From it we can see that genotype number 0 in generation 1 was the result of a crossover between genotypes 5 and 3 in generation 0. Genotype number 1 in generation 1 is the other side of the same crossover. Similarly, genotype numbers 3 and 0 in generation 0 were crossed to produce genotype numbers 2 and 3 in generation 1¹. No crossover occurred for genotype numbers 4 or 5 in generation 1.

¹Because of where the crossovers occurred, the similarity between the genotypes crossed, and the fact that the genotype length is so short, each crossover has the effect of swapping the two genotypes.

Mutation occurs with probability 0.1. For generation 1 from column ‘m’ we can see that only genotype numbers 3, 4, and 5 had a mutation. Note that genotype number 4 in generation 1 is an optimal solution to the problem because it has the maximum possible fitness value, 7. This happened because genotype number 0 from generation 0 was chosen as the parent, no crossover occurred, and the single 0-bit in the string was mutated into a 1-bit.

The creation of each generation may be followed in a similar fashion. Note how the optimal genotype gradually takes over, reducing genotypic variation in successive populations. As seen in both Table 1.1 and Figure 1.1, by generation 6 the GA has essentially reached equilibrium, or converged (what constitutes GA convergence is discussed in Section 1.5). Because there are a total of 18 bit positions in the population and the mutation rate is 0.1, about two 0s are expected to be generated by mutation in an all-1s population. In each generation after the GA has reached convergence, selection removes the 0s created by mutation in the previous generation, and mutation in the current generation creates new 0s. In effect, selection is “cleaning up” the population at each generation, only to have mutation “dirty it up” again. Thus, it is only happenstance that generation 10 contains no 0s, since it is only happenstance that no mutations occurred at generation 10.

1.2 GA Theory Objects

Conceptually, GAs may be thought of as searching the space of possible solutions by combining partial solutions into complete solutions. A complete solution, or more precisely the specification of a complete solution, is known as a *genotype*. Although in general a genotype may be variable length (for example, see Goldberg, Deb, and Korb (1989; 1990)) or use a k -ary alphabet, for simplicity a genotype will here be limited to length l binary strings. The space of all possible solutions, or the search space, is then simply all binary strings of length l .

Although the GA works directly only with a collection of complete solutions, the

population, each genotype is an exemplar of many partial solutions. A *schema*, \mathbf{h} , is a set of all complete solutions sharing a specific partial solution. The plural of schema is schemata. Sharing a partial solution means fixing, or holding constant, the bit position values specifying the partial solution in all members of the schema. All other bit positions vary freely. Since it is the partial solution which all members of a schema have in common, a schema will be referred to as a partial solution. Schemata may be specified by a *schema template*, h , a string from the alphabet $\{0, 1, *\}$. The $*$ character is used to denote a varying position, and a 0 or 1 is used at each fixed position. Thus, for a length $l = 3$ binary coding, the schema template $0 * 1$ denotes the schema $s(0 * 1) = \{001, 011\}$, where s is a function taking a schema template and returning the associated schema. Thus, $s(100) = \{100\}$, $s(*1*) = \{010, 011, 110, 111\}$, and $s(***) = \{000, 001, 010, 011, 100, 101, 110, 111\}$. Note that the notation for schemata and schema templates are distinguishable, since a boldface \mathbf{h} denotes a schema, while a plain text h denotes a schema template.² *Schema order*, $o(\mathbf{h})$, is the number of fixed positions in a schema, thus $o(s(0 * 1)) = 2$. Finally, *schema defining length*, $\delta(\mathbf{h})$, is the distance between the outermost fixed positions in the schema's schema template. Table 1.2 lists the $3^l = 27$ unique schemata for any $l = 3$ binary alphabet encodings (it also lists competition partitions, which will be dealt with shortly). Note that the size (number of elements) of a schema is $|\mathbf{h}| = 2^{l-o(\mathbf{h})}$.

All schemata taken together may be thought of as forming a lattice, or hierarchy, of schema with respect to the subset relation, \subset (Vose & Liepins, 1991). The greater the order of the schema, the smaller it is, the more complete is the partial solution it specifies, and the lower it is in the lattice. Thus for our $l = 3$ example, $s(***)$ is at the top of the lattice, since it contains every other schema, and $s(000)$, $s(001)$, $s(010)$, $s(011)$, $s(100)$, $s(101)$, $s(110)$, and $s(111)$ occupy the bottom level of the lattice, since

²This distinction will be followed generally: a bold-faced character will be used to denote the mathematical object, itself, while a non-bold-faced character will be used to denote a function returning such an object or a representation for such an object. Thus, a competition partition is denoted by \mathbf{J} while the function given a schema and returning a competition partition is denoted by J .

h	\mathbf{h}	\mathbf{J}	J	J_i
***	{000, 001, 010, 011, 100, 101, 110, 111}	{{000, 001, 010, 011, 100, 101, 110, 111}}	0	{0}
**0	{000, 010, 100, 110}	{{000, 010, 100, 110}, {001, 011, 101, 111}}	1	{0, 1}
**1	{001, 011, 101, 111}			
0	{000, 001, 100, 101}	{{000, 001, 100, 101}, {010, 011, 110, 111}}	2	{0, 2}
1	{010, 011, 110, 111}			
0**	{000, 001, 010, 011}	{{000, 001, 010, 011}, {100, 101, 110, 111}}	4	{0, 4}
1**	{100, 101, 110, 111}			
*00	{000, 100}	{000, 100}, {001, 101}, {010, 110}, {011, 111}	3	{0, 1, 2, 3}
*01	{001, 101}			
*10	{010, 110}			
*11	{011, 111}			
0*0	{000, 010}	{000, 010}, {001, 011}, {100, 110}, {101, 111}	5	{0, 1, 4, 5}
0*1	{001, 011}			
1*0	{100, 110}			
1*1	{101, 111}			
00*	{000, 001}	{000, 001}, {010, 011}, {100, 101}, {110, 111}	6	{0, 2, 4, 6}
01*	{010, 011}			
10*	{100, 101}			
11*	{110, 111}			
000	{000}	{000}, {001}, {010}, {011}, {100}, {101}, {110}, {111}	7	{0, 1, 2, 3, 4, 5, 6, 7}
001	{001}			
010	{010}			
011	{011}			
100	{100}			
101	{101}			
110	{110}			
111	{111}			

Table 1.2: Schema templates, schemata, partitions, partition numbers, and partition index sets for $l = 3$ problems.

they contain no other schema.³

A *competition partition*, denoted by \mathbf{J} , is a set of non-intersecting schemata, or partial solutions, fixing the same bit positions in their associated schema templates. For example, $\mathbf{J} = J(s(11*)) = \{s(00*), s(01*), s(10*), s(11*)\}$ is the competition partition containing all schemata fixing the two leftmost bit positions, or positions $l-1$ and $l-2$, where binary strings are labeled $x_{l-1}, x_{l-2}, \dots, x_0$. The order of a partition, $o(\mathbf{J})$, is the same as the order of the schemata it contains, or $o(\mathbf{J}) = o(\mathbf{h}), \mathbf{h} \in \mathbf{J}$. Every complete solution, for example, a genotype in a population, belongs to exactly one schema within each partition. In effect, the partial solutions within a partition compete for representation in the GA's population of complete solutions. Hence the name, competition partition — a partition covering the search space in which the schemata in the partition compete with each other for representation in the GA's population. An assumption underlying much of the work presented here is that competition partitions are the basic unit in which GA convergence occurs. Column \mathbf{J} of Table 1.2 shows the competition partition for each of the schemata of $l = 3$ problems.

Each competition partition may be identified by its *partition number*, J . The partition number may be generated by starting with the schema template of the schema in the partition whose fixed positions are all 1s, replacing each $*$ in the schema template by 0, and interpreting the resulting binary string as an integer. Column J in Table 1.2 lists partition numbers.

Each competition partition also has an associated *partition index set*, J_i , which will be used shortly in the Walsh schema transform. The elements of a partition's index set may be generated by taking the schema template of each schemata in the partition, replacing each $*$ in the template by 0, and interpreting the resulting binary string as an integer. Column J_i in Table 1.2 shows partition index sets.

³ Actually, the null set may be sensibly viewed as occupying the bottom of the lattice with the order- l schema occupying the next higher level of the lattice.

1.3 Schema Theorem

Now that basis terms have been defined, the main theoretical result about whether or not any particular partial solution will thrive is reviewed.

Schema fitness, $f(\mathbf{h})$, the average fitness of the members of the population belonging to \mathbf{h} , is

$$f(\mathbf{h}) = \frac{\sum_{\mathbf{x} \in \mathbf{h}} f(\mathbf{x})N(\mathbf{x})}{\sum_{\mathbf{x} \in \mathbf{h}} N(\mathbf{x})}, \quad (1.2)$$

where $f(\mathbf{x})$ is the genotype fitness function and $N(\mathbf{x})$ is the number of copies of genotype \mathbf{x} in the population. Similarly, each partition has a *partition fitness*, $f(\mathbf{J})$, the average fitness of its elements, that is to say, the average of its schema fitnesses. Partial solutions within each partition will, in general, have a spread in their fitnesses, which can be thought of as the partition's convergence signal. As will be shown in Chapter 3, a partition's signal is related to how much convergence occurs within that partition.

Schema fitness is central to determining whether the proportion of a population belonging to a schema will grow or shrink in the succeeding generation. The proportion of a population belonging to schema \mathbf{h} is

$$P(\mathbf{h}) = \sum_{\mathbf{x} \in \mathbf{h}} \frac{N(\mathbf{x})}{n}, \quad (1.3)$$

where $N(\mathbf{x})$ is the number of copies of genotype \mathbf{x} in the population and n is the size of the population. Define \bar{f} as the average fitness of the population, or

$$\bar{f} = \frac{\sum_{\mathbf{x} \in G} f(\mathbf{x})N(\mathbf{x})}{\sum_{\mathbf{x} \in G} N(\mathbf{x})}, \quad (1.4)$$

where G is the search space. When no crossover or mutation is performed, the expected proportion of \mathbf{h} in the population, $E(P(\mathbf{h}))$, will grow or shrink as $f(\mathbf{h}) > \bar{f}$ or $f(\mathbf{h}) < \bar{f}$, respectively. These notions have been generalized by Holland (1975b). He proved a lower bound, widely known as the *schema theorem*, on the expected proportion of schema elements in the successor population based on the current proportion, the relative magnitude of the current schema fitness to the current population fitness, and

the expected maximum disruption due to crossover and mutation. The schema theorem may be stated mathematically as

$$E(P(\mathbf{h}, t+1)) \geq P(\mathbf{h}, t) \frac{f(\mathbf{h})}{\bar{f}} \left[1 - p_c \frac{\delta(\mathbf{h})}{l-1} - p_m o(\mathbf{h}) \right], \quad (1.5)$$

where E is expectation, $P(\mathbf{h}, t)$ is the proportion of the population at time t belonging to schema \mathbf{h} , p_c is crossover probability, p_m is mutation probability. It is both a powerful theoretical tool and useful in practical applications. However, it also has limitations both because it is a bound and because it is an expectation.

1.4 Walsh-Schema Transform

Although genotype fitness is usually expressed in the binary basis, there is no intrinsic reason for doing so. As was first pointed out by Bethke⁴ (1980), Walsh functions have advantages for use as a basis for expressing schema average fitness. Walsh functions may be defined as

$$\psi_j(\mathbf{x}) = \prod_{i=0}^{l-1} (-1)^{x_i j_i}, \quad (1.6)$$

where \mathbf{x} is a binary string, and x_i and j_i denote bit i of the binary representation of each integer. Continuing our three-bit example, $\psi_0(000) = 1$, $\psi_0(011) = 1$, $\psi_1(011) = -1$, $\psi_3(011) = 1$, $\psi_5(011) = -1$, $\psi_7(011) = 1$, and $\psi_7(111) = -1$. In effect, the value of Walsh function j is the product of a bitwise exclusive-or between the binary representation of j and the binary representation of the function's argument. Likewise, a two-dimensional Walsh function may be defined (for later use) as

$$\psi_{j,k}(\mathbf{x}) = \prod_{i=0}^{l-1} (-1)^{x_i (j_i + k_i)}. \quad (1.7)$$

Likewise, $\psi_j(\mathbf{h})$ and $\psi_{j,k}(\mathbf{h})$ may be defined as

$$\psi_j(\mathbf{h}) = \prod_{i=0}^{l-1} (-1)^{\beta(h_i) j_i} \quad (1.8)$$

⁴Apparently working from a suggestion by Andy Barto.

and

$$\psi_{j,k}(\mathbf{h}) = \prod_{i=0}^{l-1} (-1)^{\beta(h_i)(j_i+k_i)}, \quad (1.9)$$

where β converts characters in the schema template associated with \mathbf{h} into bits by replacing *s with 0s.

Walsh functions provide a basis for real valued functions with integer domains expressed as binary strings. Thus, any real fitness function over binary strings, or integers, may be replaced by a linear combination of the Walsh functions,

$$f(\mathbf{x}) = \sum_{j=0}^{2^l-1} w_j \psi_j(\mathbf{x}), \quad (1.10)$$

in effect simply rewriting the fitness function in the Walsh basis, where the Walsh coefficients may be expressed as

$$w_j = \sum_{\mathbf{x}=0}^{l-1} f(\mathbf{x}) \psi_j(\mathbf{x}). \quad (1.11)$$

Schema average fitness is conventionally expressed as

$$f(\mathbf{h}) = \frac{1}{|\mathbf{h}|} \sum_{\mathbf{x} \in \mathbf{h}} f(\mathbf{x}), \quad (1.12)$$

where $|\mathbf{h}|$ is the size of \mathbf{h} . Since schema average fitness is a real function with binary domain, it may be expressed in the Walsh basis. Bethke (1980) has done this, and Goldberg (1989a) reviews and extends Bethke's work. The main result states that, under the flat population assumption, the fitness of a schema may be expressed as

$$f(\mathbf{h}) = \sum_{j \in J_i(\mathbf{h})} w_j \psi_j(\mathbf{h}), \quad (1.13)$$

where the partition index set $J_i(\mathbf{h})$ is the index set of the competition partition containing \mathbf{h} . In effect, the index set contains those terms that “make up” schema average fitness in the sense that associated Walsh functions determine parity within the fixed positions of the schema. The *flat population assumption* says that the population being considered is the complete search space, or equivalently in this case, that Equation 1.13 gives the expected fitness of a schema for a randomly generated population of genotypes. As will

\mathbf{h}	$J_i(\mathbf{h})$	$J_n(\mathbf{h})$	Walsh basis schema fitness sum
***	{0}	0	w_0
**0	{0, 1}	1	$w_0 + w_1$
**1			$w_0 - w_1$
0	{0, 2}	1	$w_0 + w_2$
1			$w_0 - w_2$
0**	{0, 4}	4	$w_0 + w_4$
1**			$w_0 - w_4$
*00	{0, 1, 2, 3}	3	$w_0 + w_1 + w_2 + w_3$
*01			$w_0 - w_1 + w_2 - w_3$
*10			$w_0 + w_1 - w_2 - w_3$
*11			$w_0 - w_1 - w_2 + w_3$
0*0	{0, 1, 4, 5}		$w_0 + w_1 + w_4 + w_5$
0*1			$w_0 - w_1 + w_4 - w_5$
1*0			$w_0 + w_1 - w_4 - w_5$
1*1			$w_0 - w_1 - w_4 + w_5$
00*	{0, 2, 4, 6}	6	$w_0 + w_2 + w_4 + w_6$
01*			$w_0 - w_2 + w_4 - w_6$
10*			$w_0 + w_2 - w_4 - w_6$
11*			$w_0 - w_2 - w_4 + w_6$
000	{0, 1, 2, 3, 4, 5, 6, 7}	7	$w_0 + w_1 + w_2 + w_3 + w_4 + w_5 + w_6 + w_7$
001			$w_0 - w_1 + w_2 - w_3 + w_4 - w_5 + w_6 - w_7$
010			$w_0 + w_1 - w_2 - w_3 + w_4 + w_5 - w_6 - w_7$
011			$w_0 - w_1 - w_2 + w_3 + w_4 - w_5 - w_6 + w_7$
100			$w_0 + w_1 + w_2 + w_3 - w_4 - w_5 - w_6 - w_7$
101			$w_0 - w_1 + w_2 - w_3 - w_4 + w_5 - w_6 + w_7$
110			$w_0 + w_1 - w_2 - w_3 - w_4 - w_5 + w_6 + w_7$
111			$w_0 - w_1 - w_2 + w_3 - w_4 + w_5 + w_6 - w_7$

Table 1.3: Walsh basis schema fitnesses for $l = 3$ problems.

be seen later, the elements of the index set are identical for every schema in any particular competition partition. Thus, a schema's average fitness may be calculated as a partial, signed sum of the Walsh coefficients, where the sign of each coefficient is determined by the parity of the schema template at the positions fixed by the particular Walsh term's index.

To make this concrete, consider once again our three-bit example. Table 1.3 shows the Walsh coefficients, including the sign produced by the associated Walsh function, in the Walsh basis schema fitness sum for all possible three-bit schemata and $J_n(\mathbf{h})$ is the

number of the partition containing \mathbf{h} . Note that Table 1.3 is applicable to all three-bit problems, regardless of coding and fitness function used, since all the coding and fitness function information is captured by the Walsh coefficients in Equation 1.13.

For example, the expected fitness of the schema $s(*1*)$ may be figured as follows. First as can be seen from Table 1.3, the index set for the summation in Equation 1.13 is $J_i(*1*) = \{0, 2\}$; thus there are two terms in the sum, $w_0\psi_0(\mathbf{h})$ and $w_2\psi_2(\mathbf{h})$. Because the parity of any schema over no fixed positions (associated with ψ_0) is even, $\psi_0(*1*) = 1$; and because parity of $*1*$ over bit position x_2 (associated with ψ_2) is odd, $\psi_2(*1*) = -1$. Thus, the Walsh basis schema fitness sum is $f(*1*) = w_0 - w_2$. Similarly, the expected fitness of schemata $*10$ and 110 are $f(*10) = w_0 + w_1 - w_2 - w_3$ and $f(110) = w_0 + w_1 - w_2 - w_3 - w_4 - w_5 + w_6 + w_7$, respectively.

So why use the Walsh basis? There are two main reasons. The first is computational. For GAs to work, the population must become rich in building blocks, so that these small partial solutions may be combined into larger partial solutions. For this enrichment to happen, the building block's fitness must be greater than that of the population as a whole. The Walsh basis sum for schema fitness contains $2^{o(\mathbf{h})}$ terms. The binary basis expression for schema fitness, Equation 1.12, contains $2^{l-o(\mathbf{h})}$ terms. Thus when $o(\mathbf{h})$ is small relative to l , as it is for building blocks, the Walsh basis expression has fewer terms. The second reason has to do with structure of the Walsh basis, schemata, and competition partitions. The Walsh basis schema fitness sum for the each schemata in a competition partition all contain the same Walsh coefficients, only the sign of the coefficients change according to the placement of 1-bits among the schema's fixed positions. The structure of the Walsh basis matches the structure of the competition partitions and schemata — the Walsh basis respects competition partitions. Thus, the Walsh basis is used to calculate the variance of schema fitness in the next chapter.

1.5 A Note on GA Convergence

Although it is not the purpose of the present work to analyze or even attempt to formally define GA convergence, it is worthwhile to characterize what is meant by GA convergence. Because the GA is a population-based search technique, convergence may be conceptualized as the reduction of diversity in the population over time (generations). It is relatively easy to tell when there is little diversity left in the population, and hence, when convergence has occurred. Usually the more important question is, can we determine how good is the solution to which the GA converged? The issues of convergence, computational complexity, and quality of solution are often formally dealt with in a convergence proof — a formal proof characterizing both the nature of convergence and the quality of the solution to be found by the search technique. Although some progress has been made towards a formal GA convergence proof (Goldberg & Segrest, 1987; Eiben et al., 1990; Goldberg, 1990b; Davis & Principe, 1991; Nix & Vose, year unknown), no one has yet established a convergence proof for the simple GA.

Several heuristic convergence measures relating to lack of population genotype diversity have been defined; they are often used as GA stopping criteria. De Jong (1975) used alleles lost as the basis for a convergence measure. Goldberg (1983) has used comparison of average fitness to maximum fitness for convergence determination and has explicitly computed the fitness variance (Goldberg, 1991a). Wilson (1987) has used an entropy-based measure of diversity for the modification of control parameters. Both a normalized measure of the average proportion of 0s versus 1s at each genotype position and average fitness have been used as convergence indicators in the evolutionary network design work presented in Chapter 5. Chapter 4 also uses the proportion of 1s versus 0s at each genotype position to indicate the relative convergence of each position. Generally speaking, when these measures become stationary, convergence is said to have occurred.

Chapter 2

Variance in Genetic Algorithms

As shown by the schema theorem, the expected fitness of a building block is an important quantity because it indicates whether, in a particular problem, the GA will be able to find optimal or near-optimal points through recombination of building blocks. On the other hand, because most GAs depend upon statistical sampling, knowing expected schema average fitness is not enough; the statistical variation, or distribution, of fitness must also be considered to determine the amount of sampling required to reliably accept or reject a building block with respect to one of its competitors (Holland, 1973; Holland, 1975a; Jong, 1975; Goldberg & Segrest, 1987; Goldberg et al., 1989; Davidor, 1991). Towards this end, it is desirable to calculate the *variance* of schema average fitness, or *collateral noise*, and for the reasons cited in Section 1.4 it is useful to do so in the Walsh basis. In the remainder of this chapter and following the work of Goldberg and Rudnick (1990), an expression for schema fitness variance in the Walsh basis is derived. The derivation is followed by a brief overview of several applications and extensions in Section 2.2, and a detailed derivation of a population size using fitness variance in Section 2.3.

2.1 Computing Schema Fitness Variance

As in the Walsh schema transform computation of schema average fitness, a flat population is assumed. Thus, the necessity of dealing with population sampling error, which incidentally can be viewed merely as an additional source of variance, is avoided. Further, the fitness function is assumed to be deterministic, so that no variance derives from

f itself.

Variance is a statistical measure of dispersion (Ross, 1987; Beck & Arnold, 1977). Its definition is

$$\text{var}(\mathbf{x}) = \sum_{\mathbf{x} \in S} P(\mathbf{x})(\mathbf{x} - \bar{\mathbf{x}})^2, \quad (2.1)$$

where the random variable \mathbf{x} ranges over discrete sample space S with probability density function $P(\mathbf{x})$, and $\bar{\mathbf{x}}$ denotes the expected value of \mathbf{x} . Thus, the variance of fitness within schema \mathbf{h} is

$$\text{var}(f(\mathbf{h})) = \frac{1}{|\mathbf{h}|} \sum_{\mathbf{x} \in \mathbf{h}} (f(\mathbf{x}) - \overline{f(\mathbf{x})})^2, \quad (2.2)$$

where $|\mathbf{h}|$ denotes the size of schema \mathbf{h} . Expanding the quadratic and simplifying yields

$$\text{var}(f(\mathbf{h})) = \overline{f^2(\mathbf{x})} - \overline{f(\mathbf{x})}^2, \quad (2.3)$$

where the sample space over which \mathbf{x} ranges in the expectations $\overline{f^2(\mathbf{x})}$ and $\overline{f(\mathbf{x})}^2$ is understood to be \mathbf{h} . The notation $\overline{f^2(\mathbf{h})}$ denotes the expectation of f^2 , and the notation $\overline{f(\mathbf{h})}^2$ denotes the square of the expected value of f .

As with the Walsh-schema transform presented in Section 1.4, both $\overline{f(\mathbf{x})}^2$ and $\overline{f^2(\mathbf{x})}$ will be derived in the Walsh basis, then Equation 2.3 will be rewritten substituting these expressions.

First, consider the equation for $\overline{f(\mathbf{x})}$ from equation (3.6) of Goldberg's (1989a) Walsh-schema paper,

$$\overline{f(\mathbf{x})} = \sum_{j \in J_i(\mathbf{h})} w_j \psi_j(\mathbf{h}), \quad (2.4)$$

where $J_i(\mathbf{h})$ is as defined in Section 1.2. Given Equation 2.4, $\overline{f(\mathbf{x})}^2$ may be written as

$$\begin{aligned} \overline{f(\mathbf{x})}^2 &= \left[\sum_{j \in J_i(\mathbf{h})} w_j \psi_j(\mathbf{h}) \right]^2 \\ &= \sum_{j, k \in J_i(\mathbf{h})} w_j w_k \psi_j(\mathbf{h}) \psi_k(\mathbf{h}). \end{aligned} \quad (2.5)$$

Recognizing that $\psi_j(\mathbf{h}) \psi_k(\mathbf{h})$ is simply the two-dimensional Walsh function,

$$\overline{f(\mathbf{x})}^2 = \sum_{j, k \in J_i(\mathbf{h})} w_j w_k \psi_{j,k}(\mathbf{h}). \quad (2.6)$$

Finally, converting the summation index from independent indices to index pairs,

$$\overline{f(\mathbf{x})}^2 = \sum_{(j,k) \in J_i^2(\mathbf{h})} w_j w_k \psi_{j,k}(\mathbf{h}), \quad (2.7)$$

where $J_i^2(\mathbf{h}) = J_i(\mathbf{h}) \times J_i(\mathbf{h})$.

Counting the number of quadratic terms is enlightening. There are $|J(\mathbf{h})|^2 = 2^{2o(\mathbf{h})}$ possibly nonzero terms in the sum. As will soon be seen, it is interesting that this number is never more than the number of terms in $\overline{f^2(\mathbf{h})}$.

The Walsh-schema transform form of $\overline{f^2(\mathbf{x})}$ is next derived. We start with the definition

$$\overline{f^2(\mathbf{x})} = \frac{1}{|\mathbf{h}|} \sum_{\mathbf{x} \in \mathbf{h}} f^2(\mathbf{x}), \quad (2.8)$$

substitute the Walsh expansion for $f(\mathbf{x})$ from Equation 1.13,

$$\overline{f^2(\mathbf{x})} = \frac{1}{|\mathbf{h}|} \sum_{\mathbf{x} \in \mathbf{h}} \left(\sum_{j=0}^{2^l-1} w_j \psi_j(\mathbf{x}) \right)^2, \quad (2.9)$$

expand the quadratic,

$$\overline{f^2(\mathbf{x})} = \frac{1}{|\mathbf{h}|} \sum_{\mathbf{x} \in \mathbf{h}} \sum_{j=0}^{2^l-1} \sum_{k=0}^{2^l-1} w_j \psi_j(\mathbf{x}) w_k \psi_k(\mathbf{x}), \quad (2.10)$$

move the sum over \mathbf{x} in and the w_j and w_k terms out,

$$\overline{f^2(\mathbf{x})} = \frac{1}{|\mathbf{h}|} \sum_{j=0}^{2^l-1} \sum_{k=0}^{2^l-1} w_j w_k \sum_{\mathbf{x} \in \mathbf{h}} \psi_j(\mathbf{x}) \psi_k(\mathbf{x}), \quad (2.11)$$

and finally, replace the product of Walsh functions with the 2-D Walsh function,

$$\overline{f^2(\mathbf{x})} = \frac{1}{|\mathbf{h}|} \sum_{j=0}^{2^l-1} \sum_{k=0}^{2^l-1} w_j w_k \sum_{\mathbf{x} \in \mathbf{h}} \psi_{j,k}(\mathbf{x}). \quad (2.12)$$

Equation 2.12, along with Equation 2.7, may be substituted directly into Equation 2.3 to produce

$$\text{var}(f(\mathbf{h})) = \frac{1}{|\mathbf{h}|} \sum_{j=0}^{2^l-1} \sum_{k=0}^{2^l-1} w_j w_k \sum_{\mathbf{x} \in \mathbf{h}} \psi_{j,k}(\mathbf{x}) - \sum_{(j,k) \in J_i^2(\mathbf{h})} w_j w_k \psi_{j,k}(\mathbf{h}), \quad (2.13)$$

a closed form expression for the fitness variance of \mathbf{h} . However, as will next be shown, and depending upon which schema is being considered, many of the (i, j) index terms in the innermost summation of $\overline{f^2(\mathbf{x})}$ may be zero, allowing further simplification of Equations 2.13 and 2.12.

Consider the 2-D Walsh function summation

$$S(\mathbf{h}, j, k) = \sum_{\mathbf{x} \in \mathbf{h}} \psi_{j,k}(\mathbf{x}), \quad (2.14)$$

which is analogous to $S(\mathbf{h}, j)$ in Goldberg's (1989a) Walsh-schema transform derivation. Each summand of Equation 2.14 is $+1$ or -1 , since each is a 2-D Walsh function. Further, the sum is bounded by $\pm|\mathbf{h}|$ since there are $|\mathbf{h}|$ terms in the sum. In fact as will next be shown, each term of the sum will be one of $+|\mathbf{h}|$, $-|\mathbf{h}|$, or 0 .

The Walsh function indices each correspond to the competition partition index numbers $J_i(\mathbf{h})$ and $\psi_{i,j}(\mathbf{x})$ functions as a mask. Consider Equation 2.14 expressed using the product form definition of the 2-D Walsh function as defined in Equation 1.7,

$$S(\mathbf{h}, j, k) = \sum_{\mathbf{x} \in \mathbf{h}} \prod_{i=1}^l (-1)^{\mathbf{x}_i(j_i+k_i)}. \quad (2.15)$$

As was mentioned earlier, $\psi_{j,k}(\mathbf{x})$ is simply the product of the bitwise exclusive-or between each bit of \mathbf{x} and the bitwise sum (no carry) of the two indices. Because \mathbf{h} is a schema rather than just an arbitrary subset of the search space, it has structure. It is the structure of \mathbf{h} interacting with the j and k indices which results in $S(\mathbf{h}, j, k)$ exclusively assuming one of the values $+|\mathbf{h}|$, $-|\mathbf{h}|$, or 0 . Whenever a $*$ character at some position i in the schema template of \mathbf{h} matches a 1-bit at position i in *exactly one* of the two indices (j or k) in Equation 2.15, there will be an equal number of $+$ terms and $-$ terms in the sum and a zero sum results. That is to say, zero sums result whenever any $*$ 'd position in \mathbf{h} occurs where the j and k indices differ. This happens because the $*$ at position i in \mathbf{h} results in an equal number of 0s and 1s at position i in the elements of $s(\mathbf{h})$. Thus, half of the summands are $+1$ and half are -1 . And when no don't-care ($*$ 'd) position in \mathbf{h} matches a difference in the indices, nonzero sums will result.¹

Consider the j and k values for which $S(\mathbf{h}, j, k) \neq 0$. Nonzero values occur exactly when each schema template don't-care (*) character matches either 0s in both indices or 1s in both, which results in even exponents for the -1 in Equation 2.15 and 1s in the product (which leave the product unchanged). Then, since fixed positions in h are fixed throughout all elements of \mathbf{h} , all terms in Equation 2.15's sum are either $+1$ or all terms are -1 , and since there are $|\mathbf{h}|$ terms in the sum, the resulting sum is one of $\pm|\mathbf{h}|$. The sign is determined by the number of 1s in h matching 1s in index j , and likewise the number of 1s in h matching 1s in index k . If the sum of these two numbers is *even*, then the sum in Equation 2.15 is *positive* and equal to $+\mathbf{h}$; if the sum of these two numbers is *odd*, Equation 2.15's sum is *negative* and equal to $-\mathbf{h}$. Again, each term of $S(\mathbf{h}, j, k)$'s sum simply performs a compound exclusive-or between each index and the schema's naming string.

Let $J_{\oplus}^2(\mathbf{h})$ be the set of index pairs corresponding to nonzero values of $S(\mathbf{h}, j, k)$. Then by recognizing that the $\frac{1}{|\mathbf{h}|}$ in Equation 2.12 is canceled out by the $|\mathbf{h}|$ resulting from the nonzero terms of Equation 2.15, Equation 2.12 may be rewritten as

$$\overline{f^2(\mathbf{x})} = \sum_{(j,k) \in J_{\oplus}^2(\mathbf{h})} w_j w_k \psi_{j,k}(\mathbf{h}). \quad (2.16)$$

Counting the number of terms in this sum is also enlightening. Thinking of the terms as being arrayed in a matrix with the j index naming rows and the k index naming columns, if we fix a row (if we fix j) there are at most $|J_i(\mathbf{h})|$ nonzero terms in the row. Each row has the same number of terms, because addition modulo-2 can do no more than translate each term to another position. Since there are 2^l rows, there are a total of $2^l |J_i(\mathbf{h})| = 2^{l+o(\mathbf{h})}$ possibly nonzero terms. This is never less than the number of terms in the $\overline{f(\mathbf{h})}^2$ sum, and as will soon be seen, the relationship is actually much closer.

¹Even though a nonzero sum results from Equation 2.15, if one of the associated Walsh coefficients in Equation 2.13 is zero the associated contribution to the sum of Equation 2.13 will be zero.

Now Equation 2.3 may be rewritten using Equations 2.16 and 2.7, producing

$$\text{var}(f(\mathbf{h})) = \sum_{(j,k) \in J_{\oplus}^2(\mathbf{h})} w_j w_k \psi_{j,k}(\mathbf{h}) - \sum_{(j,k) \in J_i^2(\mathbf{h})} w_j w_k \psi_{j,k}(\mathbf{h}). \quad (2.17)$$

Noting the two summations are identical except for the summand indices, and that $J_{\oplus}^2(\mathbf{h}) \supseteq J_i^2(\mathbf{h})$ for all \mathbf{h} , yields

$$\text{var}(f(\mathbf{h})) = \sum_{(j,k) \in J_{\oplus}^2(\mathbf{h}) - J_i^2(\mathbf{h})} w_j w_k \psi_{j,k}(\mathbf{h}), \quad (2.18)$$

where the minus sign in the summation index denotes set difference. In effect, a difference of summations has been converted to a set difference of index sets.

Consider Tables 2.1, 2.2, and 2.3, where each row shows a diagrammatic representation of the schema fitness variance computation for a length-three binary alphabet problem. The elements of each matrix dimension are indexed from 0 through 7, with 0 being in the upper left-hand corner. The first column denotes the schema. The second column shows the two-dimensional Walsh function for each schema, as defined in Equation 1.9. Each element is denoted by a plus or minus, which stand for 1 or -1 . The third column shows the nonzero elements of the two-dimensional Walsh function sum of Equation 2.14. Each element is denoted by a plus, minus, or blank space, standing for 1, -1 , and 0, respectively, and corresponding to the Walsh coefficient product pair adding, subtracting, or not participating in the sum. The fourth column shows the cross-product, $J_i^2(\mathbf{h})$, of the Walsh schema transform's index set, $J_i(\mathbf{h})$, from Equation 1.13 and is defined in Section 1.2. $J_i^2(\mathbf{h})$ participates in the Walsh variance computation through a set difference, identifying elements of $J_{\oplus}^2(\mathbf{h})$ that do not participate. Thus, elements of $J_i^2(\mathbf{h})$ are denoted by o, indicating they are 'zeroed' from the index set. That these terms should not participate makes sense since they contribute to average schema fitness, and average fitness bears no relationship to fitness variance. Finally, the fifth column shows the Walsh product pairs participating in the schema variance sum using the notation of column three.

For example from Table 2.1, $\text{var}(f(s(**))) = w_1^2 + w_2^2 + w_3^2 + w_4^2 + w_5^2 + w_6^2 + w_7^2$ because $J_{\oplus}^2(s(**)) = \{(0,0), (1,1), (2,2), (3,3), (4,4), (5,5), (6,6), (7,7)\}$ and $J_i(s(**)) = \{0\}$,

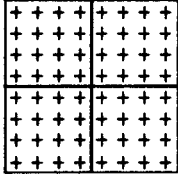
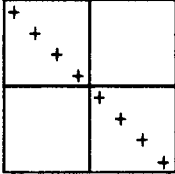
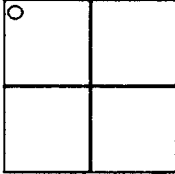
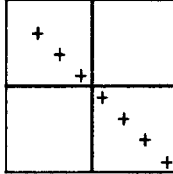
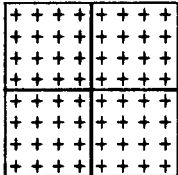
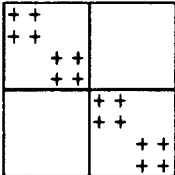
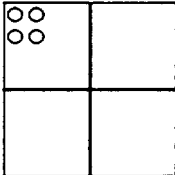
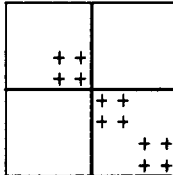
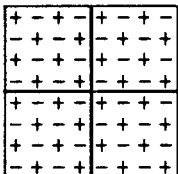
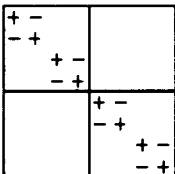
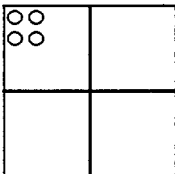
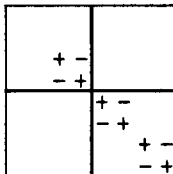
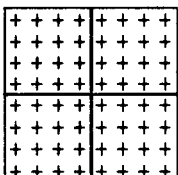
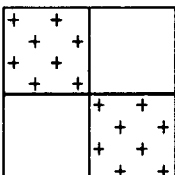
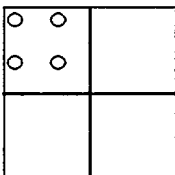
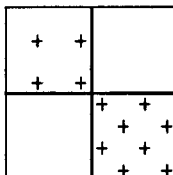
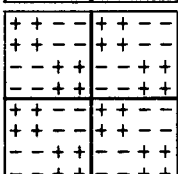
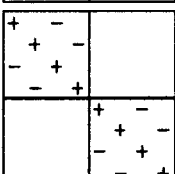
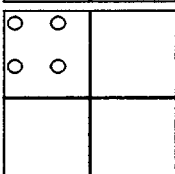
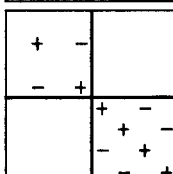
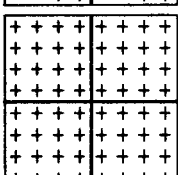
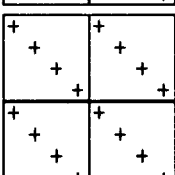
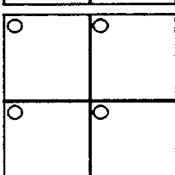
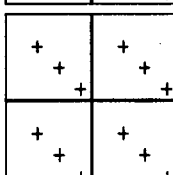
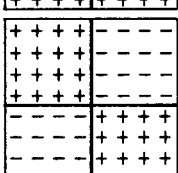
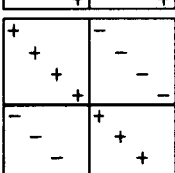
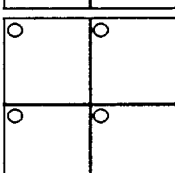
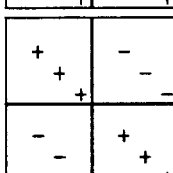
\mathbf{h}	$\psi_{j,k}(\mathbf{h})$	$S(\mathbf{h}, j, k)$	$J^2(\mathbf{h})$	$\text{var}(f(\mathbf{h}))$
$\mathbf{h}(***)$				
$\mathbf{h}(**0)$				
$\mathbf{h}(**1)$				
$\mathbf{h}(*0*)$				
$\mathbf{h}(*1*)$				
$\mathbf{h}(0**)$				
$\mathbf{h}(1**)$				

Table 2.1: Walsh function, summation, exclusion, and variance matrices for order-zero and order-one competition partitions for length-three binary encoding.

\mathbf{h}	$\psi_{j,k}(\mathbf{h})$	$S(\mathbf{h}, j, k)$	$J^2(\mathbf{h})$	$\text{var}(f(\mathbf{h}))$
$\mathbf{h}(*00)$				
$\mathbf{h}(*01)$				
$\mathbf{h}(*10)$				
$\mathbf{h}(*11)$				
$\mathbf{h}(0*0)$				
$\mathbf{h}(0*1)$				
$\mathbf{h}(1*0)$				
$\mathbf{h}(1*1)$				

Table 2.2: Walsh function, summation, exclusion, and variance matrices for order-two competition partitions $J(*ff)$ and $J(f*f)$ for length-three binary encoding.

\mathbf{h}	$\psi_{j,k}(\mathbf{h})$	$S(\mathbf{h}, j, k)$	$J^2(\mathbf{h})$	$\text{var}(f(\mathbf{h}))$
$\mathbf{h}(00*)$	<div> <div>++ ++</div> <div>++ ++</div> <div>++ ++</div> <div>++ ++</div> <div>++ ++</div> <div>++ ++</div> <div>++ ++</div> <div>++ ++</div> </div>	<div> <div>+ + + +</div> <div>+ + + +</div> <div>+ + + +</div> <div>+ + + +</div> <div>+ + + +</div> <div>+ + + +</div> <div>+ + + +</div> <div>+ + + +</div> </div>	<div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> </div>	<div> <div>+ + + +</div> <div>+ + + +</div> <div>+ + + +</div> <div>+ + + +</div> <div>+ + + +</div> <div>+ + + +</div> <div>+ + + +</div> <div>+ + + +</div> </div>
$\mathbf{h}(01*)$	<div> <div>+ + - -</div> <div>+ + - -</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> </div>	<div> <div>+ - + -</div> <div>+ - + -</div> <div>- + - +</div> <div>- + - +</div> <div>- + - +</div> <div>- + - +</div> <div>- + - +</div> <div>- + - +</div> </div>	<div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> </div>	<div> <div>+ - + -</div> <div>+ - + -</div> <div>- + - +</div> <div>- + - +</div> <div>- + - +</div> <div>- + - +</div> <div>- + - +</div> <div>- + - +</div> </div>
$\mathbf{h}(10*)$	<div> <div>++ ++</div> <div>++ ++</div> <div>++ ++</div> <div>++ ++</div> <div>- - - -</div> <div>- - - -</div> <div>- - - -</div> <div>- - - -</div> </div>	<div> <div>+ + - -</div> <div>+ + - -</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> </div>	<div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> </div>	<div> <div>+ + - -</div> <div>+ + - -</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> </div>
$\mathbf{h}(11*)$	<div> <div>+ + - -</div> <div>+ + - -</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> <div>- - + +</div> </div>	<div> <div>+ - - +</div> <div>+ - - +</div> <div>- + + -</div> <div>- + + -</div> <div>- + + -</div> <div>- + + -</div> <div>- + + -</div> <div>- + + -</div> </div>	<div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> <div>o o o o</div> </div>	<div> <div>+ - - +</div> <div>+ - - +</div> <div>- + + -</div> <div>- + + -</div> <div>- + + -</div> <div>- + + -</div> <div>- + + -</div> <div>- + + -</div> </div>

Table 2.3: Walsh function, summation, exclusion, and variance matrices for order-two competition partition $J(ff*)$ for length-three binary encoding.

so only the index pair $J_i^2(s(* * *)) = \{(0, 0)\}$ is excluded from the sum. Likewise from Table 2.3, $J_{\oplus}^2(s(10*))$ is all index pairs either both even or both odd, and $J_i(s(10*)) = \{0, 2, 4, 6\}$, so $J_i^2(h)$ is all index pairs which are both even. Thus, the Walsh schema fitness variance sum's index set, $J_{\oplus}^2(s(10*)) - J_i^2(h)$, is all index pairs which are both odd, resulting in a Walsh schema fitness variance of $\text{var}(f(s(10*))) = w_1^2 + 2w_1w_3 - 2w_1w_5 - 2w_1w_7 + w_3^2 - 2w_3w_5 - 2w_3w_7 + w_5^2 + 2w_5w_7 + w_7^2$. The other 25 schema fitness variances for a length-three problem representation are similarly computed.

Counting the number of possibly nonzero terms is once again useful. The total number of nonzero terms in the overall sum is $2^{o(h)+l} - 2^{2o(h)} = 2^{o(h)} (2^l - 2^{o(h)})$. Of course when the schemata are genotypes themselves (when $o(h) = l$), the index set becomes nil and the sum vanishes, as it must, since the fitness function is assumed to be deterministic. Note also that the Walsh variance computation may require more or less computation than a direct calculation of variance using the binary basis. Of course, fitness variance may always be calculated directly using the binary basis if that is more convenient, but the insight gained by understanding the relationship between partitions is well worth the price of admission.

2.2 Applications and Extensions

Based on the Walsh schema fitness variance computation from the previous section and related to John Holland's (1973) bandit theory analysis of the optimal allocation of trials, Goldberg and Rudnick (1991) have presented several applications and extensions of fitness variance. They first consider how schema variance changes as one moves from more general (larger) to more specific (smaller) schemata, where the more specific schemata is a subset of the more general schemata. The resulting change in schema variance derives from two sources. The first is the removing of diagonal, or squared, terms in the summation; the second results from the addition or deletion of off-diagonal, or cross-product terms.

They also show that refinement of a schema, or fixing one or more bits in the schema

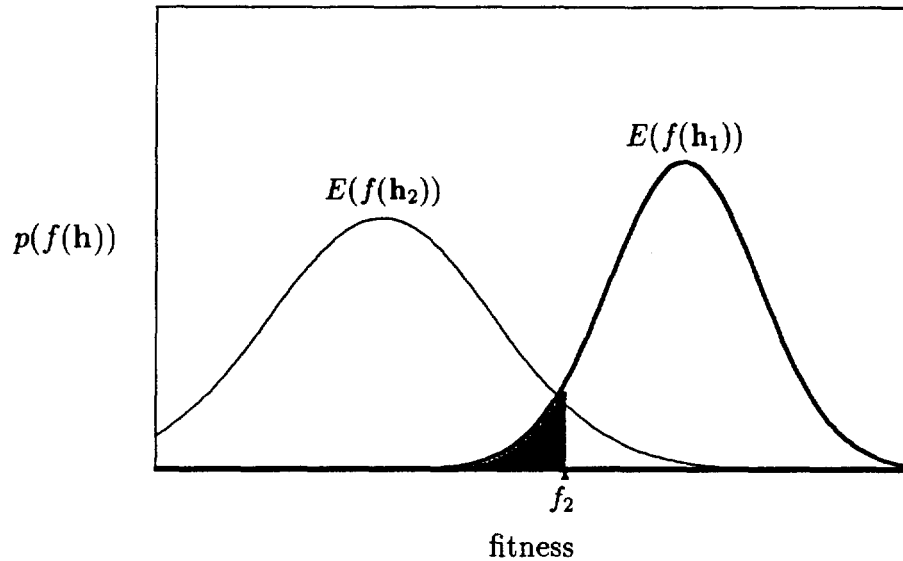


Figure 2.1: Fitness distributions for two schemata, h_1 and h_2 .

template, need not necessarily reduce the variance of the resulting schema, a possibly counter-intuitive result. Schema variance adjustments are also made to the schema theorem, eliminating the expectation operator on the left of the inequality by the use of confidence intervals based on schema fitness variance. Finally, they address the application of schema fitness variance to population sizing, which is presented in the next section and applied in Chapter 5.

2.3 Fitness Variance Based Population Sizing

Under the flat population assumption schema fitnesses do not vary. But when population size is small compared to the size of the search space, as is usually the case for GA runs, the actual schema fitnesses seen in randomly generated populations may vary considerably. In effect, generation of a random population is a statistical event, resulting in a sampling distribution of fitness for each schema.

Consider Figure 2.1. Hypothetical probability density functions for sampling distributions of fitness for two schemata are shown for a randomly generated population,

where the abscissa is schema fitness and the ordinate is probability. As can be seen, the expected fitness of \mathbf{h}_1 is greater than that of \mathbf{h}_2 , or $E(f(\mathbf{h}_1)) > E(f(\mathbf{h}_2))$. But because the schema fitness distributions overlap, for a particular population $f(\mathbf{h}_1)$ may be smaller than $f(\mathbf{h}_2)$. Say, for example, in a particular population the fitness of \mathbf{h}_2 is unusually high, as shown in Figure 2.1 by f_2 . There is then a certain probability that $f(\mathbf{h}_1) < f(\mathbf{h}_2)$, shown as the highlighted area under the $f(\mathbf{h}_1)$ distribution. Whenever $f(\mathbf{h}_1)$ falls within this range the schema theorem shows that in expectation the GA will enrich the succeeding population in \mathbf{h}_2 more than in \mathbf{h}_1 , despite the fact that \mathbf{h}_1 's expected (flat population) fitness is greater than \mathbf{h}_2 's. Thus due to stochastic variation or error, the GA will, in expectation, choose the inferior schemata.

The smaller the GA's population, the greater the fitness variance, and the greater the chance the GA will selectively enrich an inferior schema. Conversely, the larger the population, the smaller the fitness variance, and the smaller the chance of improper enrichment. Goldberg (1989a) has considered population sizing from the standpoint of schema turnover rate, knowingly ignoring variance and its effects, but explicitly identifying stochastic variation as a possibly important factor in determining appropriate population size. Because schema fitness variance, $\text{var}(f(\mathbf{h}))$, is a measure of stochastic variation, it may be used to select population size so as to minimize the probability that the sampled schema fitness ordering in a randomly generated population is wrong. Goldberg and Rudnick (1991) did just that, deriving a static formula for population size accounting for schema fitness variance. Their derivation is reviewed below in preparation for adding adjustments for objective function noise in Section 2.3.

Start by assuming that the fitness function for a maximization problem is linear or approximately linear and that all order-one terms in the Walsh expansion are equal to w'_1 . Consider all pairwise comparisons of competing order- k schemata. Choose a population size, n , so the probability that the sample mean fitness of the best schema is inferior to the sample mean fitness of the second-best schema is less than a specified value, α , or

$$P(\hat{f}(\mathbf{h}_{best}) < \hat{f}(\mathbf{h}_{2^{nd}best})) < \alpha \quad (2.19)$$

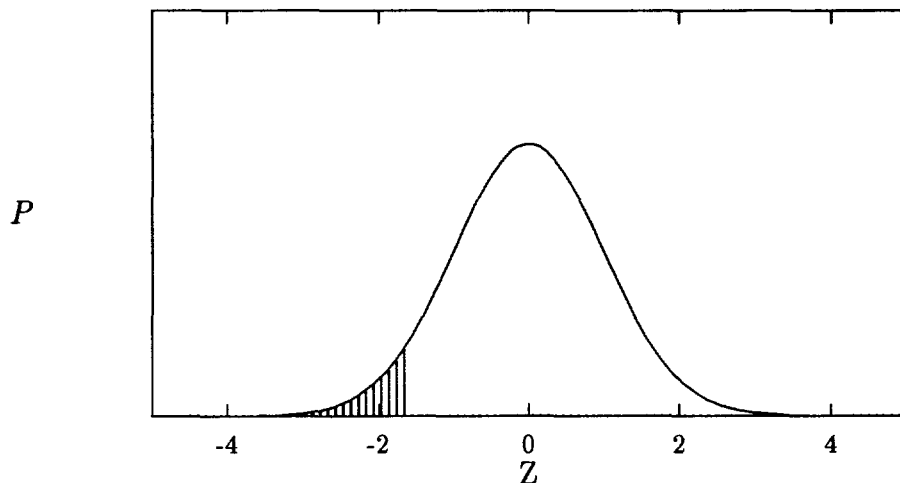


Figure 2.2: Sampling distribution probability, $P(\hat{f}(\mathbf{h}_{best}) - \hat{f}(\mathbf{h}_{2^{nd}best}))$, versus standardized z score.

where P denotes probability of an event and the hat is used to denote the sample mean.

Posed in this way, the problem becomes a statistical decision theory problem. The statistic of concern is the sampling distribution of the difference of the schema means

$$S = \hat{f}(\mathbf{h}_{best}) - \hat{f}(\mathbf{h}_{2^{nd}best}). \quad (2.20)$$

We want to know when S is non-positive, which will also be when the GA (in expectation, as per the schema theorem) mistakenly enriches the inferior schemata over the superior one; this is known as the *critical region* of a one-tailed test. The situation is as depicted in Figure 2.1, but we are now looking at the sampling distribution of the difference of the schema means, S , instead of the two schema fitness distributions \hat{f}_{best} and $\hat{f}_{s^{nd}best}$. Assuming each of the schema fitness distributions are normal, the sampling distribution of their difference will also be normal with mean μ_S and variance σ_S^2 . The sampling distribution may then be converted to a standardized normal distribution (mean of 0 and variance of 1) by

$$z = \frac{S - \mu_S}{\sigma_S}. \quad (2.21)$$

The situation is as depicted in Figure 2.2. The shaded area shows the critical region for a probability $\alpha = 0.05$ that the inferior schemata is seen by the GA as superior in any particular sample.

Assuming all variance is due to collateral noise, and assuming population sizes are large enough that the central limit theorem applies, the variance of the sample mean fitness of a single, order- k schema is

$$\text{var}(\hat{f}(\mathbf{h})) = \frac{(l-k)w_1'^2}{n/2^k}, \quad (2.22)$$

The numerator is the variance of \mathbf{h} for the flat population (which is why a static population sizing formula results), where $(l-k)$ is the number of $w_1'^2$ terms in the Walsh-variance computation; the denominator is the expected number of individuals from a population of size n belonging to an order- k schema, assuming a randomly generated population.

The sample mean fitness of the best and second-best schemata have the same variance, defined in Equation 2.22. Their sample fitness difference, the sampling distribution S , has variance

$$\sigma_S^2 = \sigma_{\hat{f}(\mathbf{h}_{best}) - \hat{f}(\mathbf{h}_{2ndbest})}^2 = \sigma_{\hat{f}(\mathbf{h}_{best})}^2 + \sigma_{\hat{f}(\mathbf{h}_{2ndbest})}^2, \quad (2.23)$$

where variance is denoted by σ^2 , since the variance of a difference of independent random variables is the sum of the individual variances. Taking the square root of each side, noting that both variances on the right are identical and defined by Equation 2.22, the standard deviation of the difference in sample mean fitness values is obtained as

$$\sigma_S = \sqrt{\frac{(l-k)2^{k+1}w_1'^2}{n}}. \quad (2.24)$$

Thus assuming independence between the schema fitness sampling distributions for \mathbf{h}_{best} and $\mathbf{h}_{2ndbest}$, the difference in sample mean fitness is

$$\mu_S = Mw_1', \quad (2.25)$$

where M is one of $2, 4, \dots, 2o(\mathbf{h})$ where \mathbf{h} is \mathbf{h}_{best} or $\mathbf{h}_{2ndbest}$ (they are of equal order since they belong to the same competition partition).

α	z	z^2
0.1	1.28	1.64
0.05	1.65	2.71
0.01	2.33	5.43
0.005	2.58	6.66
0.001	3.09	9.55

Table 2.4: Normal deviates and squared normal deviates for various levels of significance, α .

We want to find the critical region in the S distribution where $\hat{f}(\mathbf{h}_{best}) < \hat{f}(\mathbf{h}_{2^{nd}best})$. That occurs when $S \leq 0$, so

$$z \leq \frac{-\mu_S}{\sigma_S}. \quad (2.26)$$

Substituting in the worst-case difference in sample mean fitness values from Equation 2.25 and the standard deviation of S from Equation 2.24 yields

$$z \leq \frac{-Mw'_1}{\sigma_S}. \quad (2.27)$$

Squaring Equation 2.27, substituting for σ_S from Equation 2.24, and simplifying yields an expression for the population size,

$$n \geq \frac{z^2(l-k)2^{k+1}}{M^2}. \quad (2.28)$$

Note that the w'_1 factors dropped out of Equation 2.28 when Equation 2.24 was substituted into Equation 2.27; thus given the stated assumptions, population size is independent of the particular value for w'_1 . By examining Table 1.3, the worst-case sample fitness difference for Equation 2.28 is $2w'_1$, since mean sample fitness difference is inversely related to population size. Thus, a sufficient population size, irrespective of which schemata within the competition partition are considered, is

$$n = z^2(l-k)2^{k-1}. \quad (2.29)$$

Table 2.4 shows z and z^2 values for the one-sided test corresponding to selected levels of significance, α . Each z value corresponds to the probability that the difference

between the sample mean fitness of the best and second-best schema is negative is α . For example, considering $k = 1$ at a significance level of 0.1, the population sizing formula becomes $n = 1.64(l - 1)$. Many problems are run with strings of length 30 to 100, for which the formula would suggest population sizes in the range 49 to 164. This range is not inconsistent standard GA practice.² Similar reasoning may be used to derive population sizing formulas if the building blocks are scaled nonuniformly or if the function is nonlinear.

Note that the population sizing analysis presented applies only to the initial, randomly selected population; hence the descriptor *static* population sizing. To perform population sizing analysis after the initial generation, additional factors must be considered. Some factors lead to an increased population size, while others lead to a decreased size.

Since $\alpha \geq 0.5$ for even extremely small populations, the probability the correct choice will be made is never worse than 0.5, or no worse than a random walk. During GA operation, the least-fit individuals are quickly culled from the population. Thus the representation, or proportion, of more-fit schemata in a competition partition increases while the proportion of less-fit schemata decreases, a phenomena referred to here as *schema enrichment*. As a result, the fitness variances of the better schemata are reduced at the expense of increasing the fitness variance of the poorer schemata. Overall, the probability that the best schema will be selected over the second-best schema increases as schema enrichment progresses, but with the proviso that hitchhiking³ (Schaffer et al., 1991) and genetic drift⁴ (Goldberg & Segrest, 1987) don't destroy the exemplars of the more-fit schemata before they can be used. Thus, a population smaller than dictated

²For example, De Jong (1975) empirically tried population sizes of 50, 100, and 200 on his f1 problem (minimization of a three-dimensional paraboloid), finding that the larger populations gave small improvements in off-line performance while actually increasing on-line performance over the observed number of evaluations.

³Hitchhiking is a genotype linkage effect in which a passive allele (an allele not at the moment actively selected for) at a locus near another locus containing an actively selected allele 'hitchhikes' a ride into the offspring.

⁴Genetic drift is the tendency of a population not undergoing active selection pressure to nevertheless stray from its initial schemata distribution due to random walk effects.

by the static sizing formula may produce satisfactory results. Schema enrichment suggests the use of larger populations during the early portions of a GA run and smaller populations during the later stages.

Of course, there are factors that may serve to increase schemata fitness variance following the initial generation. These include stochastic effects resulting from the action of selection, genetic operators, linkage coupling and disruption effects, and variance in the fitness function (Section 5.5.2 and 5.6 each deal with the issue of population sizing in the presense of fitness function noise). Since all of these except fitness function variance are reduced as the GA population converges towards a single genotype, they too tend to suggest gradually decreasing population size during a GA run. These issues are left to future research.

Chapter 3

Signal Versus Noise

Most GAs function by sampling schema fitness. Because populations of modest size are generally used, schema fitness variance is a primary source of stochastic noise which can hamper correct evaluation of building blocks.

In Chapter 2 a Walsh basis expression for static schema fitness variance is derived. It is then used to control the probability of one source of incorrect evaluation of building blocks; for two schemata, the schema fitness difference (signal) and schema fitness variance (noise) are juxtaposed as a basis for static population sizing.

In the present chapter these notions of signal and noise are generalized from two schemata to competition partitions. Section 3.1 presents an overview of the roles of signal and noise. Rigorous definitions of signal, noise, and the signal-to-noise (SNR) ratio are given in Sections 3.2, 3.3, and 3.4. For each, a Walsh basis expression is derived under the flat population assumption. Finally, the relevance of the SNR to the quality of GA decision-making is then discussed in Section 3.5.

3.1 Overview of GA Signal and Noise

Each schema may be viewed as a partial solution defined by its fixed positions. Schemata are organized into competition partitions — competing partial solutions, or sets of schemata fixing the same bit positions. Thus, every complete solution, such as a member of a GA's population, belongs to exactly one schema within each competition partition. In effect, the partial solutions within a competition partition compete for representation

in the GA's population of complete solutions.

Each schema has as its *schema fitness*, $f(\mathbf{h})$, the average fitness of its elements. The partial solutions (schemata) within each partition will, in general, have a spread in their fitnesses, which can be thought of as the partition's selection pressure, or convergence signal. The greater the fitness spread among the partition's schemata (partial solutions), the greater the partition's signal strength, and the greater will be the convergence occurring within the partition. It is this spread in fitness which enables the GA to select better partial solutions in the partition.

The GA continually operates to enrich succeeding populations with respect to the fitness of the partial solutions represented within the population. It does this by preferentially selecting individual solutions having above-average fitness with respect to the current population. When some partitions have strong signals while other partitions have weak signals, the GA pays attention (through the mechanism of selection) to the strong signals at the expense of the weaker signals. In effect, the GA has only so much selection attention to distribute among the various competition partitions.

Thus from the point of view of a particular partition, signals from all other partitions contribute to noise competing with its signal. The net effect is that the 'signal' from partitions with weak signals is lost among the loud 'noise' from partitions with strong signals. This happens when the difference between the strong and weak partition signals is large relative to the size of the population and results in poor-quality GA decisions being made in the weak partition.

3.2 Signal

We define the measure of the force tending toward convergence within a competition partition as the square root of the variance of the schema fitnesses of the schemata within the partition, and call it the *partition signal strength*, $S(\mathbf{J})$, or in mathematical form,

$$S^2(\mathbf{J}) = \text{var}(f(\mathbf{J})). \quad (3.1)$$

The reason for the squaring is that variance is itself a squared measure. Note that signal may be computed directly from any GA population (Bridges & Goldberg, 1991) by using a proportion-based definition of variance,

$$\text{var}(f(\mathbf{J})) = \sum_{\mathbf{h} \in \mathbf{J}} P(\mathbf{h})(f(\mathbf{h}) - \bar{f})^2, \quad (3.2)$$

where $P(\mathbf{h})$ is the proportion of the population in schema \mathbf{h} , $f(\mathbf{h})$ is now the average fitness of that part of the population in \mathbf{h} , and \bar{f} is the average fitness of the population.

Next, an expression for a partition's static signal strength in the Walsh basis is derived. Restating Equation 3.1 in terms of the variance expression given in Equation 2.3,

$$S^2(\mathbf{J}) = \overline{f^2(\mathbf{h})} - \bar{f}(\mathbf{h})^2, \quad (3.3)$$

where \mathbf{h} varies over the schemata in \mathbf{J} . Tackling the first term in Equation 3.3,

$$\overline{f^2(\mathbf{h})} = \frac{1}{|\mathbf{J}|} \sum_{\mathbf{h} \in \mathbf{J}} f^2(\mathbf{h}), \quad (3.4)$$

from the definition of the mean, where $|\mathbf{J}|$ is the number of schemata in \mathbf{J} . Substituting the expression for $f(\mathbf{h})$ from the Walsh-schema transform, Equation 1.13, yields

$$\overline{f^2(\mathbf{h})} = \frac{1}{|\mathbf{J}|} \sum_{\mathbf{h} \in \mathbf{J}} \left(\sum_{j \in J_i(\mathbf{h})} w_j \psi_j(\mathbf{h}) \right)^2. \quad (3.5)$$

Expanding the quadratic yields

$$\overline{f^2(\mathbf{h})} = \frac{1}{|\mathbf{J}|} \sum_{\mathbf{h} \in \mathbf{J}} \sum_{j,k \in J_i(\mathbf{h})} w_j w_k \psi_{j,k}(\mathbf{h}). \quad (3.6)$$

Note that all schemata within a single partition share the same index set, i.e., $J_i(\mathbf{h}_j) = J_i(\mathbf{h}_k)$ for all $\mathbf{h}_j, \mathbf{h}_k \in \mathbf{J}$. Thus, the same Walsh coefficient products occur $|\mathbf{J}|$ times in the outer summation, but with possibly different signs due to the action of $\psi_{j,k}(\mathbf{h})$. In fact, because of the orthogonality of the Walsh basis and because each partition covers the entire search space, when $j \neq k$ an equal number of plus and minus terms occur for each Walsh product pair, $w_j w_k$, resulting in the elimination of all off-diagonal products. Thus, Equation 3.6 reduces to

$$\overline{f^2(\mathbf{h})} = \sum_{j \in J_i(\mathbf{J})} w_j^2, \quad (3.7)$$

where $J_i(\mathbf{J})$ has been substituted for $J_i(\mathbf{h})$ by noting that all schemata in a partition share the same index set.

The last term of Equation 3.3 can be expanded analogous to Equations 3.4, 3.5, and 3.6, resulting in

$$\overline{f(\mathbf{h})}^2 = \left[\frac{1}{|\mathbf{J}|} \sum_{\mathbf{h} \in \mathbf{J}} f(\mathbf{h}) \right]^2 = \left[\frac{1}{|\mathbf{J}|} \sum_{\mathbf{h} \in \mathbf{J}} \sum_{j \in J_i(\mathbf{h})} w_j \psi_j(\mathbf{h}) \right]^2. \quad (3.8)$$

Likewise, because \mathbf{J} covers the space of all possible genotypes and Walsh functions are orthogonal, Equation 3.8 simplifies to

$$\overline{f(\mathbf{h})}^2 = \left[\frac{|\mathbf{J}|}{|\mathbf{J}|} w_0 \right]^2 = w_0^2, \quad (3.9)$$

as it must since the average of the averages of equal-sized partition elements is simply the average of the underlying space, which for the entire search space is w_0 .

A Walsh expression for a partition's squared signal, $S^2(\mathbf{J})$, may now be formed by substituting the Walsh basis expressions for $\overline{f^2(\mathbf{h})}$ and $\overline{f(\mathbf{h})}^2$ from Equations 3.7 and 3.9, respectively, into Equation 3.3, producing

$$S^2(\mathbf{J}) = \sum_{j \in J_i(\mathbf{J})} w_j^2 - w_0^2. \quad (3.10)$$

Since w_0 is in every $J_i(\mathbf{J})$, the effect of subtracting w_0^2 is to remove it from the equation altogether, which is equivalent to removing zero from the index set, and Equation 3.10 may be restated as

$$S^2(\mathbf{J}) = \sum_{j \in J_i(\mathbf{J}) - \{0\}} w_j^2, \quad (3.11)$$

where the minus sign in the index expression denotes set difference. As an example, the squared signal of partition $\mathbf{J} = ff^*$ is $S^2(ff^*) = w_2^2 + w_4^2 + w_6^2$. Equation 3.11 is a remarkably simple expression — a partition's squared signal strength is just the sum of the squares of the Walsh coefficients of order-one or greater in the partition's index set. Further, the expression is general, in the sense that nothing has been assumed about the form of the fitness function. Its role in GA decision-making is discussed in Section 3.5.

3.3 Noise

Signals from other partitions contribute to noise competing with the signal of the partition under consideration for the control of the GA's selection process. We define *partition root-mean-squared noise*, $C(\mathbf{J})$, as the square root of the average of the collateral noise values for each schema in the competition partition under consideration, or

$$C^2(\mathbf{J}) = \overline{\text{var}(f(\mathbf{h}))} = \frac{1}{|\mathbf{J}|} \sum_{\mathbf{h} \in \mathbf{J}} \text{var}(f(\mathbf{h})), \quad (3.12)$$

where $\overline{\text{var}(f(\mathbf{h}))}$ is the average collateral noise among the schemata in the partition. Notice that, as with signal, noise may also be directly computed given a specific population by using the proportion-based definition of variance, Equation 3.2.

Next, an expression for a partition's static root-mean-squared noise in the Walsh basis is derived. Substituting the definition of $\text{var}(f(\mathbf{h}))$ from Equation 2.18 into Equation 3.12 yields

$$C^2(\mathbf{J}) = \frac{1}{|\mathbf{J}|} \sum_{\mathbf{h} \in \mathbf{J}} \sum_{(j,k) \in J_{\oplus}^2(\mathbf{h}) - J_i^2(\mathbf{h})} w_j w_k \psi_{j,k}(\mathbf{h}). \quad (3.13)$$

As with Equation 3.6, and for the same reason, the off-diagonal product terms are zero and can thus be eliminated from the index set. Note that the remaining (diagonal) entries in the inner summation's index set are exactly the elements which are not in $J_i(\mathbf{J})$. Thus, the index set is $\overline{J_i(\mathbf{J})}$, the complement of $J_i(\mathbf{J})$. As in Equation 3.6 the outer summation cancels against the $1/|\mathbf{J}|$ term because each term is added $|\mathbf{J}|$ times, and Equation 3.13 simplifies to

$$C^2(\mathbf{J}) = \sum_{j \in \overline{J_i(\mathbf{J})}} w_j^2. \quad (3.14)$$

Continuing the previous example, the squared noise of partition $\mathbf{J} = ff^*$ is $C^2(ff^*) = w_1^2 + w_3^2 + w_5^2 + w_7^2$.

Note that signal plus noise equals a constant determined by the particular fitness function used, or

$$S^2(\mathbf{J}) + C^2(\mathbf{J}) = \sum_{j \in \{1,2,\dots,l-1\}} w_j^2. \quad (3.15)$$

Thus signal and noise can each be expressed in terms of the other, as in

$$C^2(\mathbf{J}) = \sum_{j \in \{1,2,\dots,l-1\}} w_j^2 - S^2(\mathbf{J}) \quad (3.16)$$

and

$$S^2(\mathbf{J}) = \sum_{j \in \{1,2,\dots,l-1\}} w_j^2 - C^2(\mathbf{J}). \quad (3.17)$$

Next the equations for signal and noise are combined to give a measure of which has the upper hand in a particular situation.

3.4 Signal-to-Noise Ratio (SNR)

The *signal-to-noise ratio* (SNR), $R(\mathbf{J})$, is a measure reconciling the opposing effects of signal strength and noise. It is defined as

$$R(\mathbf{J}) = \frac{S(\mathbf{J})}{C(\mathbf{J})}. \quad (3.18)$$

Substituting in the expressions for squared signal and squared noise from Equations 3.11 and 3.14 gives

$$R(\mathbf{J}) = \sqrt{\frac{\sum_{j \in J_i(\mathbf{J}) - \{0\}} w_j^2}{\sum_{j \in J_i(\mathbf{J})} w_j^2}}. \quad (3.19)$$

Completing the ongoing example for partition $\mathbf{J} = ff*$,

$$R(ff*) = \sqrt{\frac{w_2^2 + w_4^2 + w_6^2}{w_1^2 + w_3^2 + w_5^2 + w_7^2}}. \quad (3.20)$$

Note the way in which each Walsh coefficient other than w_0 occurs exactly once in Equation 3.19, contributing to either squared signal (the numerator) or squared noise (the denominator). That w_0 , the average fitness over the entire search space, does not participate in S , C , or R , makes sense since both signal and noise are composed exclusively of combinations of variances, which plays no part in the search space's average fitness. That Equation 3.19 is so simple is yet another demonstration of how the Walsh basis respects competition partitions.

Finally, note that $R(\mathbf{J})$ is undefined for the competition partition containing order- l schemata, since its noise is zero. Likewise, $R(\mathbf{J})$ for the competition partition whose single element is the order-zero schema is zero, since $S(\mathbf{J}) = 0$.

3.5 Discussion

The definitions of a competition partition's signal, noise, and SNR are general, in the sense that they assume nothing about the form of the fitness function. Likewise, the definitions do not require the flat population assumption; signal, noise, and the SNR may each be computed for arbitrary populations by using the proportion-based definition of variance, Equation 3.2. However, since the Walsh expression for schema fitness variance, Equation 2.18, requires the uniform population assumption, the Walsh expressions for signal, noise, and SNR are static.

Of what relevance is the SNR? Holland (1973) used a statistical decision theory approach to develop his bandit theory analyses of the GA's exploration versus exploitation behavior. A partition's signal, noise, and SNR each induce a total order on competition partitions. That is to say, each may be used to rank competition partitions into a linear sequence in which ties are possible. From a statistical decision theory perspective, the SNR is a measure appropriate to reconciling the opposing effects of signal and noise with respect to GA decision-making. It effectively establishes which partition's convergence has first call on the control of selection events, and thus establishes a convergence priority queue among competition partitions.

$R(\mathbf{J})$ generalizes the statistical-decision-theory-based population size analysis of Section 2.3. There it was shown how stochastic variation can result in the schema with the best expected fitness having a sample fitness in a randomly generated population less than that of an inferior schema. The schema theorem then shows how such variation can result in the GA mistakenly enriching an inferior schema.

The static population sizing calculation used the sampling distribution of the fitness difference of two schemata to establish probabilistic bounds on making such a sampling

error as a function of population size. That calculation used a ratio of the mean of the sampling distribution of the fitness difference of the two schemata to its variance, computing the critical region of the standardized z distribution in Equation 2.27. Taking that equation, setting $M = 2$ (see Equation 2.25) for an order-one partition, using the flat population variance instead of the sampling distribution variance, replacing z by z' to distinguish the two distributions, and simplifying yields

$$z' \leq \frac{1}{\sqrt{l-1}}. \quad (3.21)$$

Equation 3.21 is also the value of the SNR under the assumptions made by Equation 2.27, or

$$R(J) = \sqrt{\frac{w_1'^2}{(l-1)w_1'^2}} = \frac{1}{\sqrt{l-1}}. \quad (3.22)$$

Thus, the SNR may be thought of as generalizing the statistical decision theory based population sizing computation of Section 2.3 from competition partitions of size two to arbitrary competition partitions and removing some of the simplifying assumptions. The numerator of Equation 2.26 is generalized to the root-mean-squared error of the partition's schema fitnesses, and the denominator is generalized to the root of the average schema variance in the partition. Note that although Equation 3.22 assumes the flat population, the definition of the SNR makes no such assumption (only the Walsh basis expression derivations require the uniform population assumption) and may therefore be computed for arbitrary populations by using the proportion-based definition of variance, Equation 3.2. Thus, the SNR functions to measure the search-space dependent likelihood that the fitness spread among the competing schemata of any particular partition is sufficient to control selection events amongst all the other partitions vying for control.

Chapter 4

Domino Convergence

It has long been known that the more significant bits of binary-coded GAs converge more rapidly than bits of lesser significance (Schraudolph & Belew, 1990, for example), a phenomena here called domino convergence. The intuitive explanation is that there is more convergence pressure on the more significant bits. A closely related phenomena sometimes occurs when convergence stops prematurely, a phenomenon dubbed convergence stall. Several analyses and models are presented exploring various aspects of these phenomena.

The chapter is organized as follows. Section 4.1 presents a simple problem demonstrating both domino convergence and convergence stall. Section 4.2 presents an analysis of expected initial convergence window width. Section 4.3 presents analysis of convergence stall point using first a simple model and then a more refined model. Section 4.4 presents a signal-to-noise ratio based analysis. Finally, Section 4.5 discusses the role of mutation in the light of convergence stall.

4.1 Simulation

The identity problem, $f(\mathbf{x}) = x$, is chosen to demonstrate domino convergence and convergence stall, where \mathbf{x} is interpreted as the binary fraction $.x_1x_2\dots x_l$. Since the problem is linear, it may be stated in the binary basis as

$$f(\mathbf{x}) = \sum_{i=1}^l a_i x_i \quad (4.1)$$

where $a_i = 1/2^i$. The problem is tackled with a simple GA using tournament selection, single-point crossover, and mutation. A 53-bit¹ genotype is used, with population of size 100 randomly initialized, and crossover rate of 0.5. A series of runs were made for mutation rates of 0.0025, 0.005, 0.01, 0.02, 0.04, and 0.08.² In all cases 300 generations were used. In order to wash out stochastic deviations, 1000 simulation runs³ were averaged for each mutation rate.

Figure 4.1 shows plots of the proportion of 1s in the population at each loci versus generation number for mutation rates of 0.0025, 0.005, 0.01, 0.02, 0.04, and 0.08. In each plot the abscissa is time in generations and runs from time zero, the randomized initial generation, through generation 240 in steps of three generations. Each ordinate encodes locus (bit position in the genotype) and runs from locus 53, the least significant bit, to locus one, the most significant binary digit. The Z, or vertical, axis shows convergence level for each locus as a proportion of 1s in the current population, and runs from 0.5 at the bottom to 1.0 at the top. The flat, upper plateaus for a mutation rates of 0.0025, 0.005, and 0.01 are areas of nearly full convergence. The seas, or lower plateaus, represent areas of no convergence, where the number of 0s and 1s in the population at each locus as averaged over the 1000 runs are approximately equal. The intermediate Z values on the slopes represent areas of good skiing.⁴

The relatively large amount of variation among the individual runs can be seen both from Figure 4.3, which shows a single GA run with a mutation rate of 0.01, and from the plots of Figure 4.2, which show the standard deviation of convergence proportion for each of the six runs shown in Figure 4.1. In Figure 4.2 the ordinate axis is reversed so that the slopes “behind the mountains” are not hidden.

Figure 4.4 shows loci converged versus time in generations for each of the different

¹This value was determined by the number of bits of precision available in double-precision floating-point numeric representation.

²These values were chosen to bracket the crossover point, falling between 0.01 and 0.02, between fully converged runs and stalled runs (mentioned later in this section).

³The number of simulation runs to be included in each average was determined by how many runs could be performed in a reasonable period of time, rather than on the basis of a statistical test of significance based on the amount of variance present in the distribution of runs.

⁴Actually, they represent areas of partial convergence.

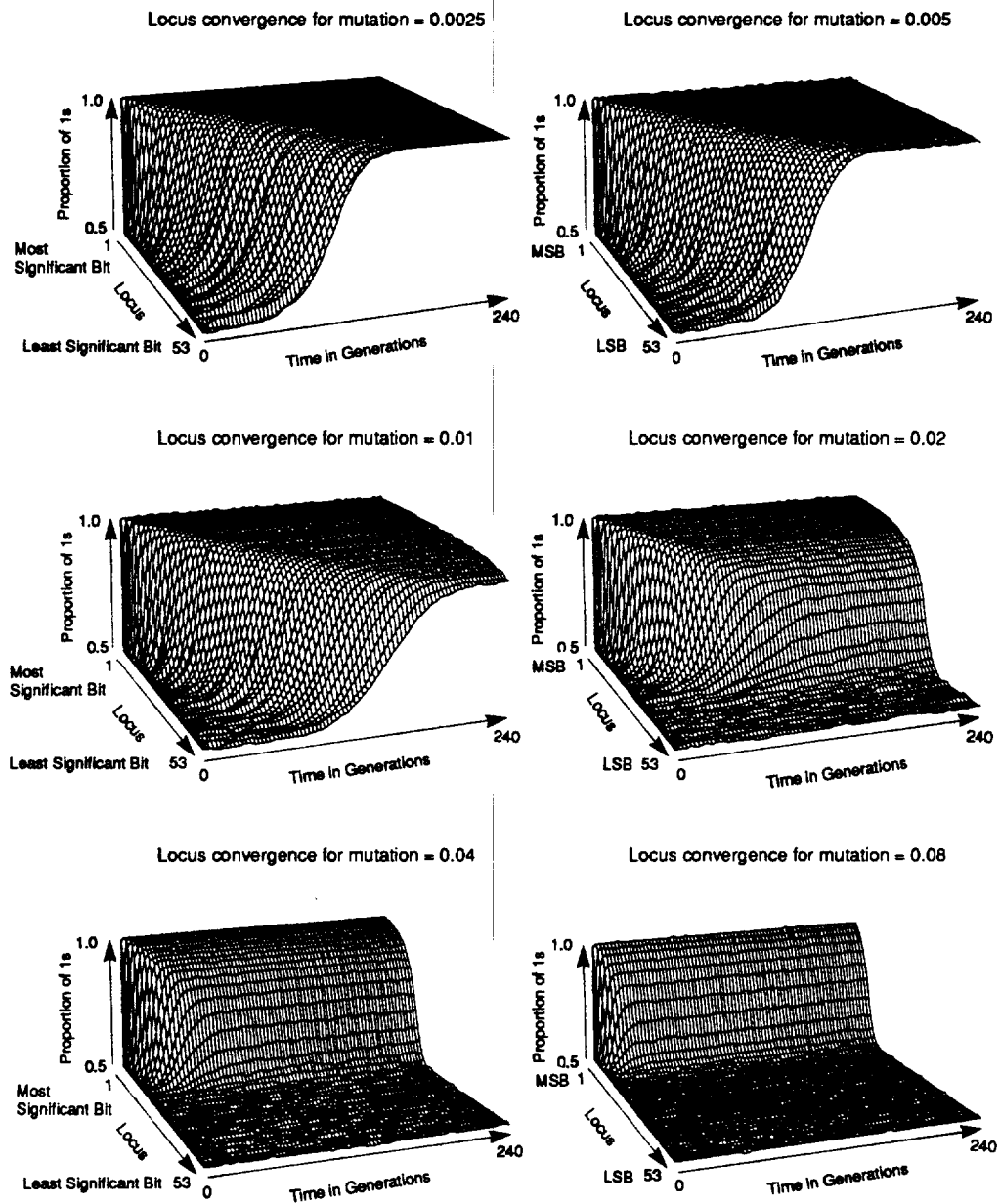


Figure 4.1: Convergence levels for mutation rates: first row, 0.0025 and 0.005; second row, 0.01 and 0.02; third row, 0.04 and 0.08.

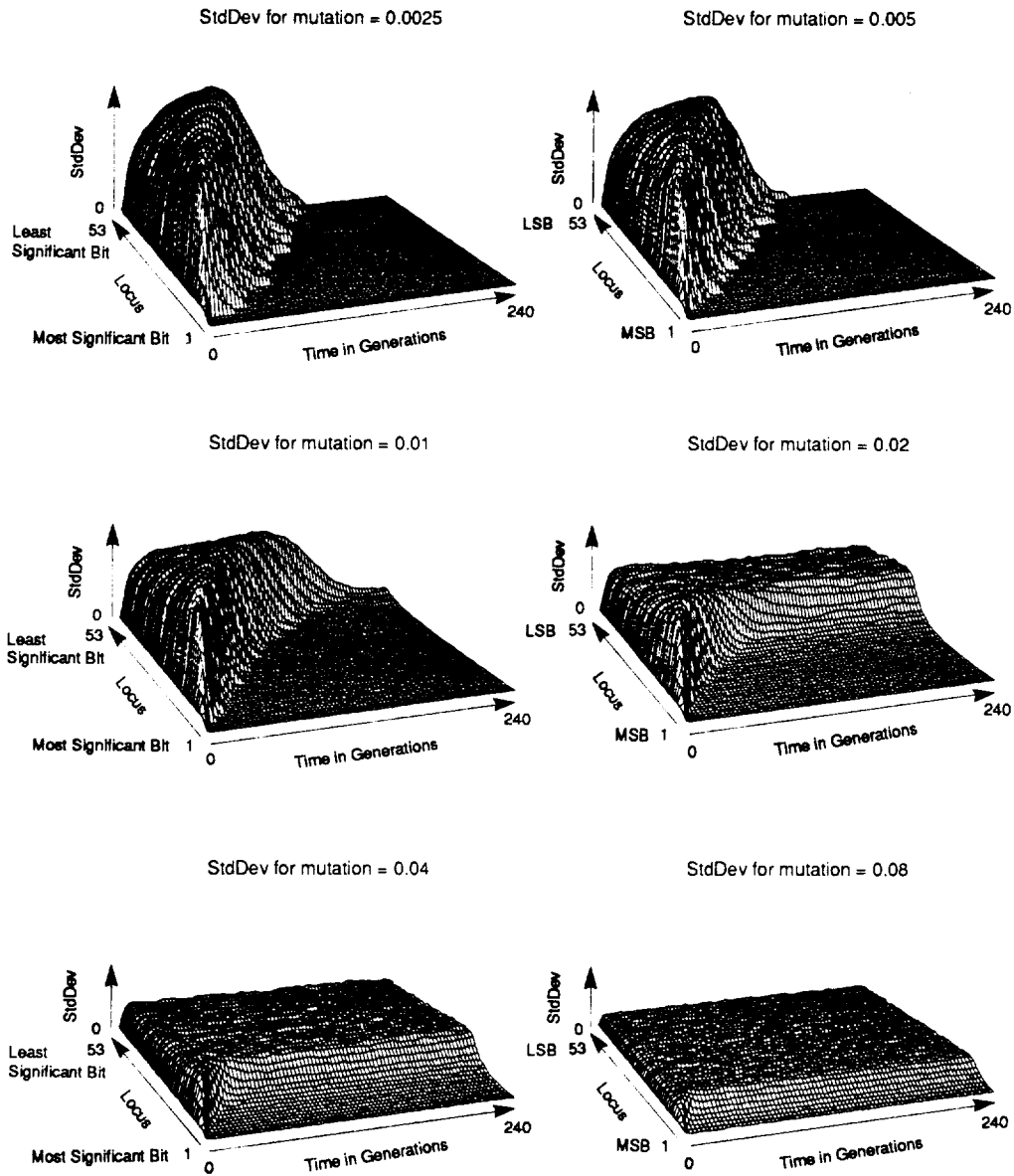


Figure 4.2: Standard deviations of convergence for mutation rates: first row, 0.0025 and 0.005; second row, 0.01 and 0.02; third row, 0.04 and 0.08.

mutation values, where convergence is defined as a proportion of at least 0.9 1s. Both Figure 4.4 and the perspective plots show how convergence quickly “stalls out” in runs with higher mutation rates.

Despite the fact that tournament selection exerts more convergence pressure than many other forms of selection (Goldberg & Deb, 1991), the runs sometimes took over 200 generations to reach a steady-state (all positions constant on the convergence plots of Figure 4.1) on this simple problem. The main factor causing such slow convergence is that no convergence is occurring in the less significant loci until the more significant loci have converged. In other words, the convergence of the more significant loci allow loci of lesser significance to converge, not unlike a row of dominos falling over in succession — hence the name, *domino convergence*. From a signal versus noise perspective,⁵ the large signal from non-converged, higher-significance bits is controlling selection events; this effectively precludes the relatively weak signals of the low-significance bits from controlling any selection events, and results in no convergence taking place among the low-significance bits until the higher-significance bits have converged. Domino convergence is a general phenomena in GA convergence, not limited to binary coded parameters.

The net result is that a *convergence window* exists in which convergence actively occurs; outside the window little or no convergence occurs. The situation is depicted in Figure 4.5, which shows locus position on the abscissa with the most significant loci to the left, versus convergence (proportion of 1s) at each locus along the ordinate. At the start of a simulation run, the convergence window is at the left edge of the abscissa and contains only the most significant loci. As the simulation proceeds, the window slides to the right. In effect, three distinct convergence regimes exist, each with its own invariant conditions. Loci to the left of the window are already fully converged; loci to the right of the window have undergone no convergence; only loci within the window are actively converging.

From the signal versus noise perspective, the signal level of each locus is ranked

⁵See Section 4.4 for a uniform-population derivation of expressions for signal, noise, and the signal-to-noise ratio for each bit position.

from highest to lowest. For the initial, random population in expectation, the most significant bit's locus has the highest signal, while the least significant bit's locus has the lowest signal. At any particular time the loci whose signals are above the current noise level are actively controlling selection events, while those below the current noise level are either already converged and are in the converged region (for example, a fully converged partition has a signal of zero), or have undergone no convergence and are in the unconverged region. As the loci in the convergence window become fully converged, they move into the converged region and their contribution to the noise level is eliminated. The resulting lowering of the noise level eventually allows the signal of the locus in the unconverged region next to the convergence window to be heard so that it begins to affect selection events; thus, its partition begins to converge. The net effect is that as convergence proceeds the convergence window slides to the right.

Note that with fixed nonzero mutation rates and sufficient bits in the genotype, the window's movement to the right will eventually slow and stop. The window stops when the increase in convergence due to selection is exactly offset by the reduction in convergence due to mutation among already converged loci, or from the signal versus noise perspective, when the reduction in noise due to selection is exactly offset by the increase in noise due to mutation among the already converged loci. Thus, a steady-state "stalling out" of convergence, or *convergence stall*, results, as can be clearly seen in the later generations of the plots for mutation rates 0.02, 0.04, and 0.08 in Figure 4.1.

4.2 Analysis of Convergence Window Width

In this section convergence window width is analyzed for the demonstration problem, resulting in a model of expected convergence window width at the start of a run.

For the $f(x) = x$ problem, what factors determine convergence window width? To answer this question first consider how binary tournament selection works. Since tournament selection is a rank-based selection method, only the ordering of fitness values of the genotypes in the population matter — other than this, the actual fitness values do not

Single Run Convergence, $P_m = 0.01$

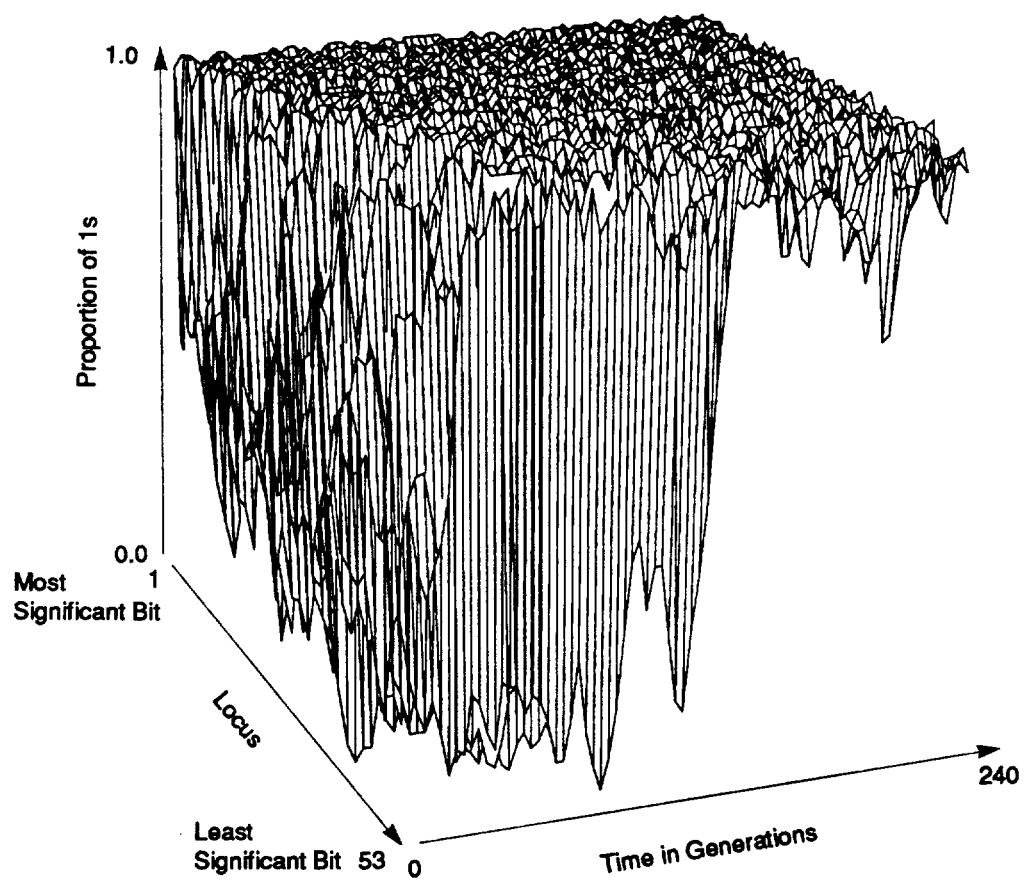


Figure 4.3: Results from a single run.

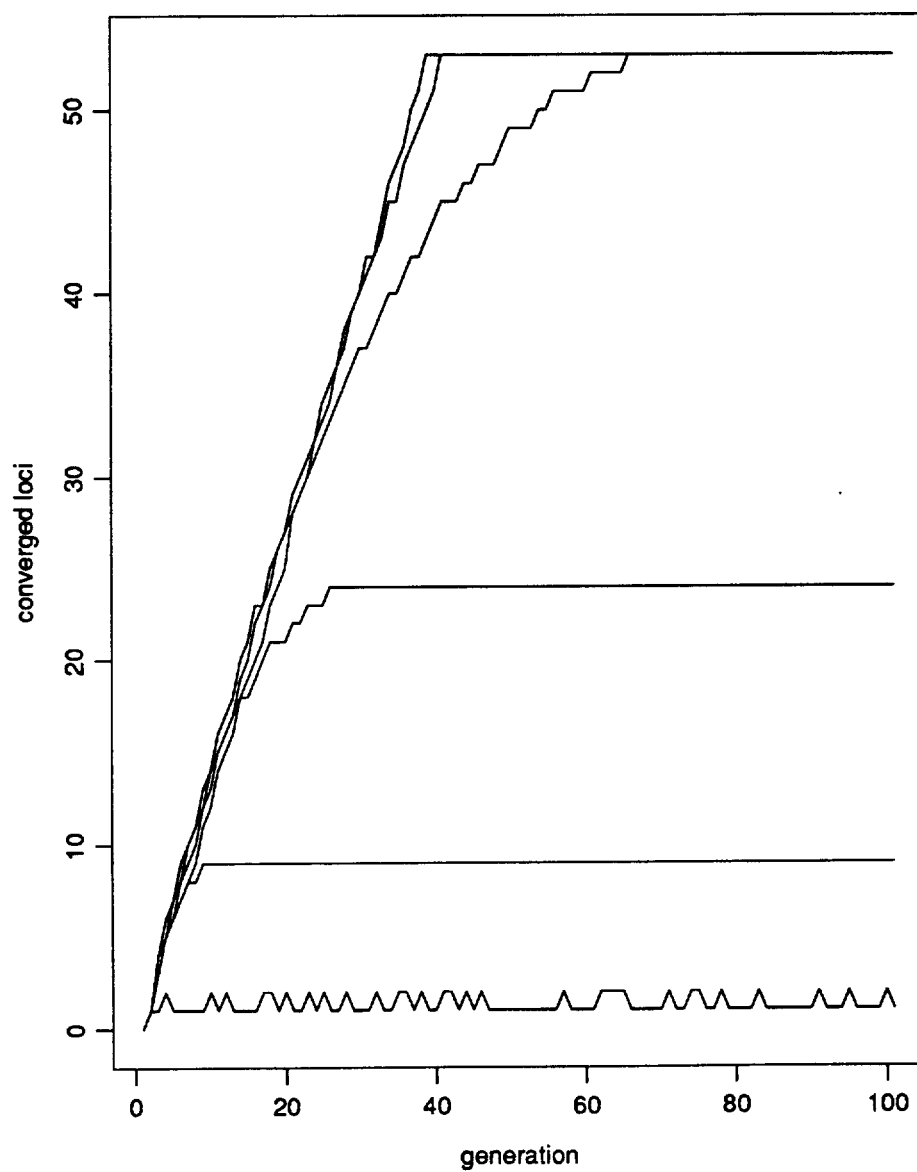


Figure 4.4: Loci converged versus time for 90% convergence for various mutation rates. Plot lines, from bottom to top, correspond to GA runs with mutation rates of 0.08, 0.04, 0.02, 0.01, 0.005, and 0.0025.

parent 1		parent 2		decided by locus 1	decided by locus 2	decided elsewhere
locus 1	locus 2	locus 1	locus 2			
0	0	0	0			•
0	0	0	1		•	
0	0	1	0	•		
0	0	1	1	•		
0	1	0	0		•	
0	1	0	1			•
0	1	1	0	•		
0	1	1	1	•		
1	0	0	0	•		
1	0	0	1	•		
1	0	1	0			•
1	0	1	1		•	
1	1	0	0	•		
1	1	0	1	•		
1	1	1	0		•	
1	1	1	1			•

Table 4.1: Enumeration of tournament selection events controlled by each of the 16 possible two-locus tournament pairings. The first four columns give the values of the four alleles at the two most significant genotype positions of the two parents. The last three columns use ‘•’ to show where the tournament is decided.

matter at all. Interpreting the genotype as a binary fraction as was done in Section 4.1, the rank of a genotype’s fitness is identical to the rank of the genotype itself. This is true not only for the demonstration problem, but for any monotone fitness function. Thus, the relative ranking of two individuals paired in a tournament depends entirely upon their allele values at the first locus at which they differ, and further, loci at which the parent individuals have identical alleles play no part in determining relative ranking.

Given a random population, an expression may be derived for the expected proportion of tournament pairings decided at each locus. The four possible pairings of locus one’s alleles among two parents are (0,0), (0,1), (1,0), and (1,1). In the (0,0) and (1,1) pairings, allele fitness contributions are irrelevant because the tournament will be decided at a less-significant locus. But in the (0,1) and (1,0) pairings, the relative allele fitness determines the outcome and the child will have a 1 at locus one. Thus in expectation, half the time

relative allele fitness at locus one determines the outcome of the tournament. Similarly in considering which tournaments are decided at locus two, only the loci of the first and second alleles are relevant. As shown in Table 4.1, there are 16 possible pairings covering all possible genotype combinations of two alleles in each parent. In 1/4 of these 16 possible pairings, the fitness contribution of the locus-two alleles determine the outcome of the tournament.

In general for locus i , $1/2^i$ proportion of the time the values of locus i 's alleles determine the outcome of the tournament. Thus for a population of size n , the expected number of individuals selected on the basis of the values of the alleles at locus i will be $n/2^i$. Notice that the number of individuals selected at each locus must be integral, since only whole individuals can be selected. Thus for the population of size 100 used in the domino convergence simulation suite, locus six should control $100/2^6 = 1.56$ selection events, or about one or two individuals, while locus seven should control $100/2^7 = 0.78$, or about 0 or 1 individuals. Therefore, the expected width of the initial convergence window should be six or seven loci. Checking against the simulation runs shown in Figure 4.1 reveals an increase in the average proportion of 1s in generation three (the next generation shown after the initial, random, generation zero) for only the first seven, eight, or nine loci, a good match to the width predicted by the initial convergence window width model.

In general and after the initial generation, either a fully or partially converged region will exist among the most significant loci, and the random population assumption of the initial convergence window width model will be violated. The analysis can be extended to times beyond the initial generation by deriving expressions for the proportions of each allele at each locus in the converged and partially converged regions. Approximations for these expressions are derived in the next section.

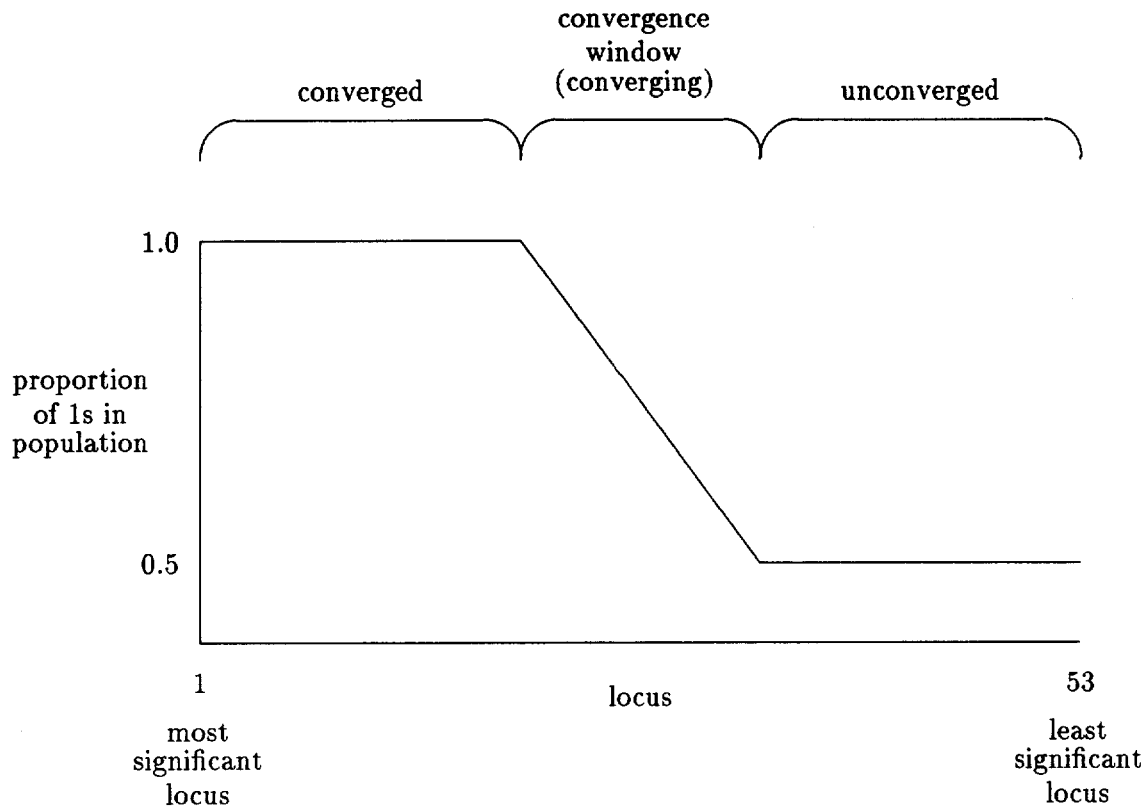


Figure 4.5: Context of locus i for simplified convergence stall analysis.

4.3 Analysis of Convergence Stall

In this section two analyses of convergence stall are given. First, a simplified model derived by Goldberg (1991b) is defined and compared to empirical results, then a more refined model is considered.

4.3.1 Simple Model

Consider again the context for the simplified steady-state analysis of convergence stall as shown in Figure 4.5. From left to right, the three convergence regions are shown. First, the most significant loci comprise the converged region; note that in the initial

competing alleles	outcome (child)	probability
1 vs 1	1	P_t^2
1 vs 0	1	$P_t(1 - P_t)$
0 vs 1	1	$P_t(1 - P_t)$
0 vs 0	0	$(1 - P_t)^2$

Table 4.2: Possible tournament pairings at locus i and their probabilities of occurrence.

window width analysis of the previous section, the size of the converged region was zero. Second, the next most significant loci comprise the partially converged region in which convergence is actively taking place, the convergence window. Third, loci of lesser significance comprise the unconverged region.

Consider locus i , the most significant locus in the convergence window. The outcome probability part of the initial window width model of Section 4.2 is generalized to allow an arbitrary proportion of 1 alleles at locus i . Locus one of the previous model now corresponds to locus i in Figure 4.5, the first position within the convergence window. As in the previous model, the 1 alleles are assumed to have higher fitness than the 0 alleles. If all of the more significant loci in the converged region were to remain fully converged (no mutations occurred among them during previous generations), then locus i would be the first to have a shot at affecting the selection between the genotypes engaged in a tournament. As shown in Table 4.2, either a 1 or a 0 allele may result at locus i from such a tournament. A 1 results in three ways having combined probability of $2P_t - P_t^2$, where P_t represents the proportion of 1s in the population at locus i . A 0 results with probability $(1 - P_t)^2$, only when both individuals have a 0 at position i .

The possibility that a mutation either occurred in the converged region during the previous generation (so that it now prevents the alleles at locus i from determining the outcome of the tournament) or occurs at locus i during this generation is accounted for as follows. The probability that a mutation occurs at a locus is p_m , so the probability that no mutation occurs is $1 - p_m$. The probability that no mutations occurred in the converged region or will occur at mutation i is $(1 - p_m)^{c+1}$, where c is the number of loci in the converged region. Thus, the probability that one or more mutations either

occurred in the converged region or occurs at locus i is

$$p'_m = 1 - (1 - p_m)^{c+1}. \quad (4.2)$$

Finally, the probability that no such mutation occurred is $1 - p'_m$. Incorporating this into the group probabilities derived from Table 4.2 yields the probability that a 1 results from a tournament pairing as

$$P_{t+1} = (2P_t - P_t^2)(1 - p'_m). \quad (4.3)$$

P_{t+1} is the proportion of 1s produced in the next generation. This will be called the *simple model*. That Equation 4.3 is an approximation may easily be seen by considering the decision tree shown in Figure 4.7, a complete model for the context presented in Figure 4.5. This full model will be explored in detail in Section 4.3.2. Note that there are four leaves which produce 1s in the next generation, while Equation 4.3 employs only the single term which corresponds roughly to the leftmost leaf of the decision tree. Thus the proportion of 1s at succeeding generations is underestimated, and a corresponding underestimate of the convergence stall point is to be expected.

To estimate the convergence stall point based on the model of Equation 4.3, we set $P_{t+1} = P_t$, which results in

$$P_{ss} = \frac{1 - 2p'_m}{1 - p'_m}, \quad (4.4)$$

where P_{ss} is the fixed point of the equation, the proportion of 1s at locus i at steady-state. We arbitrarily define 'converged' to mean 90% or more 1s, substituting $P_{ss} = 0.9$ into Equation 4.4, and solving for p'_m yields

$$p'_m \cong 0.091. \quad (4.5)$$

Substituting 0.091 for p'_m in Equation 4.2 and solving for c yields

$$c \cong \frac{-0.0414}{\log_{10}(1 - p_m)} - 1. \quad (4.6)$$

Figure 4.6 shows the 90% convergence stall points from the simulation runs presented in Section 4.1 (denoted by triangles), those predicted by the simple model (denoted by

octagons), and the streamlined model presented in Section 4.3.2 (denoted by squares). As expected, the simple model underestimates the stall points. To try to get a better analytical estimate, a more refined analytical model is next considered.

4.3.2 Refined and Streamlined Models

The simple model presented in the preceding section significantly underestimates the 90% convergence stall points because several of the sources of 1 alleles shown in the decision tree of Figure 4.7 are ignored. To improve the stall point estimate, a set of recurrence equations are defined modeling the proportion of tournaments available to be decided at locus i . The recurrence equations are then simplified to the convergence context depicted in Figure 4.5 and incorporated into a more refined model. The model is then compared to both the empirical simulation results and to the predictions of the simplified model.

A fairly general set of recurrence equations may be written defining D_i , the proportion of tournaments available to be decided at locus i (i.e., not already decided at more significant loci), and d_i , the expected proportion of tournaments decided at locus i . For locus $i = 1$,

$$D_1 = 1, \quad (4.7)$$

since all tournaments are available to be decided at locus 1, and with P_1 the proportion of 1s at locus i ,

$$d_1 = 2P_1(1 - P_1), \quad (4.8)$$

since the tournament will be decided at locus one when the allele pairs are either (0,1) or (1,0), each of which occurs with probability $P_i(1 - P_i)$, where P_i is the proportion of 1 alleles at position i . For locus $i = 2$ the respective equations are

$$D_2 = D_1 - d_1, \quad (4.9)$$

since all tournaments are available to be decided at locus two except those already decided at locus one, and

$$d_2 = D_2 2P_2(1 - P_2) \quad (4.10)$$

Convergence Stall, simple model vs runs

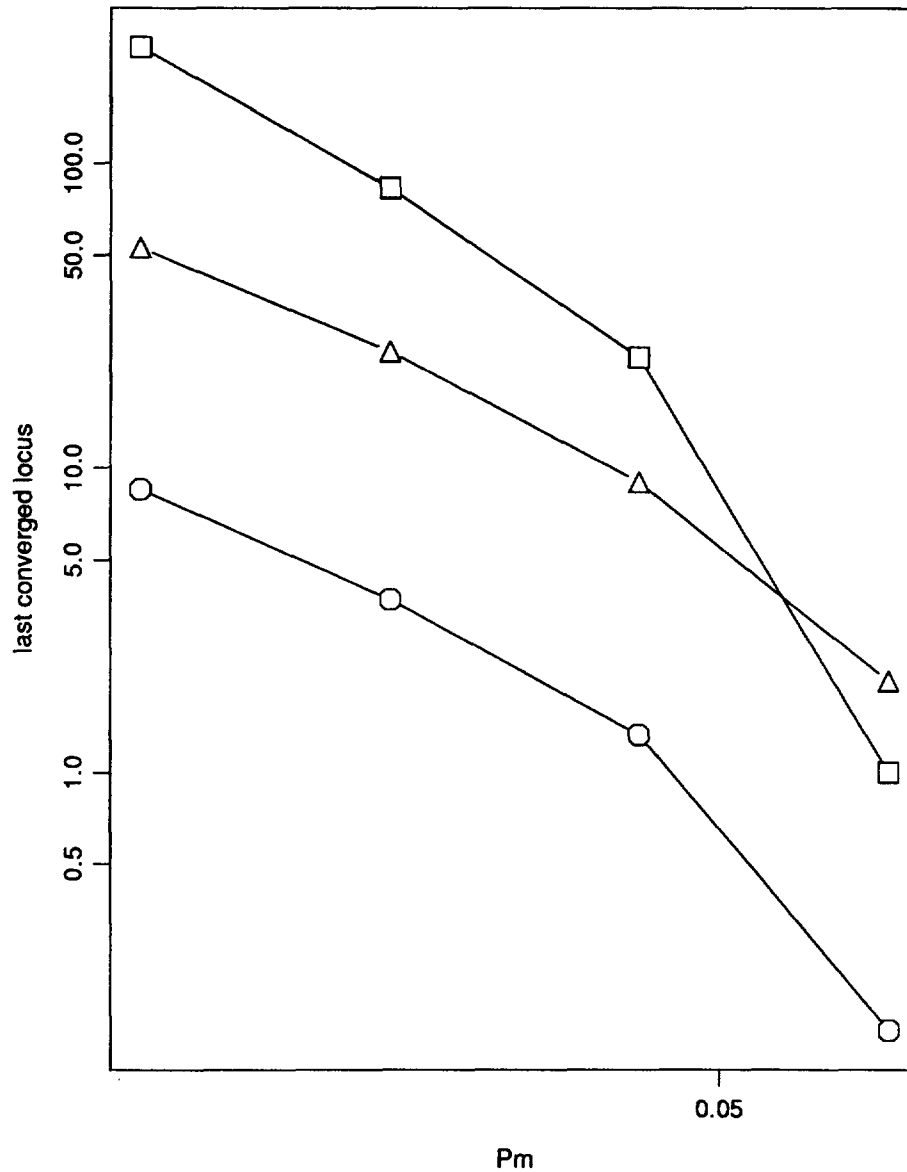


Figure 4.6: Steady-state convergence level versus last-converged locus for empirical simulations (triangles), simple model (octagon), and streamlined model (squares). The definition of convergence used requires 90% or more 1s at each locus from the most significant position to the last-converged position. Mutation rates (P_m 's) of 0.01, 0.02, 0.04, and 0.08 are shown on a linear scale. Note that the 0.0025 and 0.005 mutation rates are not shown, since the performance of the simulation runs is clipped due to the limited precision of double precision floating point representation.

since there are D_2 tournaments available to be decided at locus two, and $2P_2(1 - P_2)$ of those will be decided at locus two. Substituting for D_1 and d_1 from Equations 4.7 and 4.8 into Equations 4.9 and 4.10 yields

$$D_2 = 1 - 2P_1(1 - P_1) \quad (4.11)$$

$$d_2 = [1 - 2P_1(1 - P_1)]2P_2(1 - P_2) \quad (4.12)$$

In general,

$$D_i = D_{i-1} - d_{i-1} = \prod_{j=1}^{i-1} [1 - 2P_j(1 - P_j)] \quad (4.13)$$

and

$$d_i = 2P_i(1 - P_i)D_i = 2P_i(1 - P_i) \prod_{j=1}^{i-1} [1 - 2P_j(1 - P_j)]. \quad (4.14)$$

Equations 4.13 and 4.14 are a general model for the expected proportion of tournaments decided at locus i , and assume only that the fitness function is strictly monotone, monotonically increasing as opposed to simply non-decreasing, and the proportion of 1 alleles at each locus is known. The model applies to all three convergence regimes shown in Figure 4.5 — the converged area to the left of the convergence window, the area of active convergence within the convergence window, and the area of nonconvergence to the right of the window.

The model given in Equations 4.13 and 4.14 may be combined with a probabilistic model for the allelic value resulting at locus i from a tournament pairing. The combined model is shown as a decision tree in Figure 4.7. Where present, node labels indicate the value of the resulting child's locus i allele at that point in the process. Each branch is labeled with both the action the branch represents and the probability with which that action occurs. Levels in the tree proceed from left to right. The first level of the tree determines whether or not the tournament is decided above locus i . If not (lower branch at level one), the second level determines whether a 0 or 1 allele results from the tournament. If the tournament was decided above locus i (top branch at level one), the second level determines whether a 0 or 1 allele hitchhikes at locus i . Finally in all cases, level three determines if a mutation occurs at locus i to change the resulting allele. Each

leaf represents a possible outcome with the associated probability being the product of the probabilities along the path leading to the leaf. Four of the outcomes (leaves) result in 1 alleles at locus i in the resulting child and four result in 0 alleles. Together the leaves cover all possible outcomes.

Combining the model of Equations 4.13 and 4.14 with the decision tree probability model from Figure 4.7, the expected proportion of 1s at locus i in the next generation is

$$P_{i,t+1} = \left[(2P_{i,t} - P_{i,t}^2)(1 - p_m) + (1 - P_{i,t})^2 p_m \right] D_{i,t} + [P_{i,t}(1 - p_m) + (1 - P_{i,t})p_m](1 - D_{i,t}). \quad (4.15)$$

Making substitutions for $D_{i,t}$ from Equation 4.13 yields

$$P_{i,t+1} = \left[(2P_{i,t} - P_{i,t}^2)(1 - p_m) + (1 - P_{i,t})^2 p_m \right] \prod_{j < i} [1 - 2P_{j,t}(1 - P_{j,t})] + [P_{i,t}(1 - p_m) + (1 - P_{i,t})p_m] \left\{ 1 - \prod_{j < i} [1 - 2P_{j,t}(1 - P_{j,t})] \right\}. \quad (4.16)$$

Equation 4.16 is a fully general model applicable to all three convergence regimes and will be called the *refined model*. It assumes only that all loci are independent, as is the case with a bitwise linear fitness function or when uniform crossover is used by the GA, an assumption required by the hitchhiking branch leaving the upper level one node of the decision tree. The model could be programmed and run iteratively to produce a trajectory, in expectation, of the proportion of each allele at each location. It can then be used to investigate the steady-state behavior of the modeled GA, although this has not been done here. Explicit solution of Equation 4.16 for the steady-state stall points appears impractical.

To make further progress, the refined model of Equation 4.16 is simplified by restricting the locus under consideration to be the leftmost, or most converged, position in the active convergence window as depicted in Figure 4.5. Further, any mutations occurring in the converged region are assumed to be completely corrected by selection in each generation, so that no mutations accumulate from generation to generation. The new model will be called the *streamlined model*.

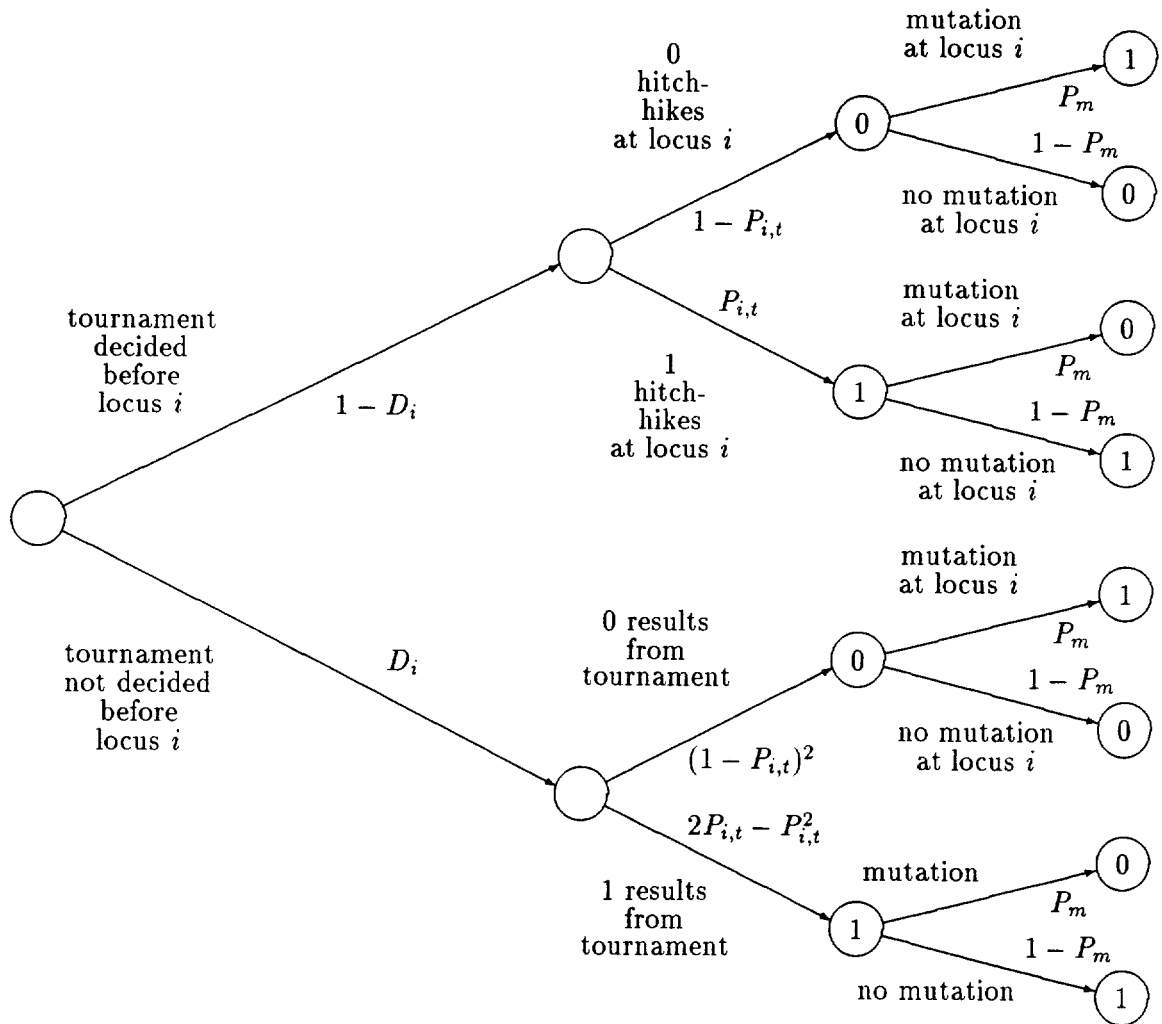


Figure 4.7: Locus decision tree — combined model.

Figure 4.8 shows the decision tree for the streamlined model. The main difference is that the probabilities $D_{i,t}$ and $1 - D_{i,t}$ at the first level of the tree have been simplified to $(1 - p_m)^x$ and $1 - (1 - p_m)^x$, where p_m is the bitwise mutation probability and x is the number of bits in the converged region. The resulting difference equation for proportion of 1s at locus i in the succeeding generation is obtained by adding the probabilities for the 1 leaves from Figure 4.8, producing

$$\begin{aligned} P_{t+1} = & (1 - p_m)^x (2P_t - P_t^2)(1 - p_m) + (1 - p_m)^x (1 - P_t)^2 p_m \\ & + [1 - (1 - p_m)^x] P_t (1 - p_m) + [1 - (1 - p_m)^x] (1 - P_t) p_m. \end{aligned} \quad (4.17)$$

Note that the i subscript has been dropped since that aspect is now fully captured by x in the streamlined model. By setting P_{t+1} to P_t and solving the resulting quadratic, the expected steady-state, or fixed point, proportion of 1s becomes parameterized by p_m and x . Figure 4.6 shows the steady-state convergence level versus mutation rate for the streamlined model by the line denoted by squares, as compared to the simulation runs of Section 4.1 denoted by triangles, and the simplified model of Section 4.3.1 denoted by octagons. It can be seen that the streamlined model generally predicts convergence stall will occur at higher positions than are observed empirically. For example, the $p_m = 0.02$ run shown in Figure 4.1 clearly shows convergence in the GA simulation runs stalling out around locus 30, while the streamlined model predicts stall should not occur until position 80 or so. A possible source for the discrepancy may be the assumption, made by both the simple and streamlined models, that no mutations accumulate in the converged region from generation to generation. Such an oversimplification would underestimate the number of mutations in the converged region and therefore overestimate the convergence stall position. It may be possible to adjust Equation 4.17 for such mutation accumulation, but that has been left as future research.

The GA convergence models presented are all limited to monotone fitness functions. In the next section, the more general signal versus noise model defined in Chapter 3 is applied to domino convergence.

4.4 Domino Convergence and the Signal-to-Noise Ratio

In tournament selection with strictly monotone fitness, a mutant allele in the converged region takes precedence over all alleles of lower significance. From the signal versus noise perspective, mutation acts to constantly inject collateral noise into the converged region, just as it does in the other regions. But the mutations occurring in the converged region are of special significance, since they generate a relatively higher level of noise. This noise serves to lower the signal-to-noise ratio (SNR) of the loci of lesser significance, effectively competing with the fitness discrimination signals of the loci in the convergence window. When the amount of mutation noise injected into the converged region becomes large enough, convergence stall results. As the mutation rate or the size of the converged region is increased, the noise eventually overwhelms the signal. In this section, the $f(\mathbf{x}) = x$ domino convergence demonstration problem of Section 4.1 is examined from the signal versus noise perspective.

As shown by Goldberg (1989a), any linear fitness function may be stated in terms of its Walsh basis as

$$\begin{aligned} f(\mathbf{x}) &= b + \sum_{i=0}^{l-1} a_i x_i, \quad x_i \in \{0, 1\} \\ &= w_0 + \sum_{i=0}^{l-1} w_{2^i} \psi_{2^i}(\mathbf{x}), \end{aligned} \quad (4.18)$$

where the Walsh coefficient w_{2^i} denotes the indices of the order-one Walsh coefficients, and the values of the Walsh coefficients are

$$\begin{aligned} w_0 &= b + \frac{1}{2} \sum_{i=0}^{l-1} a_i \\ w_{2^i} &= -\frac{a_i}{2}, \quad i \in \{0, 1, \dots, l-1\}. \end{aligned} \quad (4.19)$$

Because $f(\mathbf{x})$ is linear, only the order-zero and order-one Walsh coefficients are nonzero. Throughout this section the competition partitions considered will be limited to those of order-one or less, since all other Walsh coefficients are zero.

Thus for linear fitness functions, the Walsh basis expression for squared signal, Equation 3.11, becomes

$$S^2(\mathbf{J}) = \sum_{j \in J_S(\mathbf{J}) \cap W_1} w_j^2, \quad (4.20)$$

where $J_S(\mathbf{J}) = J_i(\mathbf{J}) - \{0\}$, the usual partition signal index set used in Equation 3.11, and W_1 is the set of indices for the order-one Walsh coefficients. In effect, the usual partition signal sum is used, but restricted to order-one Walsh coefficients. Each order-one competition partition has exactly one coefficient satisfying these constraints, namely, the order-one coefficient corresponding to the locus partition \mathbf{J} fixes, or makes constant.

In similar fashion, the Walsh basis expression for squared noise, Equation 3.14, simplifies to

$$C^2(\mathbf{J}) = \sum_{j \in J_C(\mathbf{J}) \cap W_1} w_j^2, \quad (4.21)$$

where $J_C(\mathbf{J}) = \overline{J_i(\mathbf{J})}$, the usual partition noise index set used in Equation 3.14. Again, the usual partition noise sum is used, but restricted to order-one Walsh coefficients because the higher-order coefficients are all zero.

Finally, the SNR Walsh basis expression, Equation 3.19, becomes

$$R(\mathbf{J}) = \sqrt{\frac{\sum_{j \in J_S(\mathbf{J}) \cap W_1} w_j^2}{\sum_{j \in J_C(\mathbf{J}) \cap W_1} w_j^2}}. \quad (4.22)$$

Note that Equations 4.20, 4.21, and 4.22 apply to any linear fitness function. Since for linear fitness functions the number of nonzero Walsh coefficients is linear in the length of the genotype, actually performing these computations becomes attractive as compared to the case of the general fitness function, where the number of potentially nonzero Walsh coefficients is exponential in the length of the genotype.⁶

Consider again the identity function, $f(\mathbf{x}) = x$, used in Chapter 4, Equation 4.1, as the domino convergence demonstration problem fitness function. Since $f(\mathbf{x})$ is a linear

⁶Of course, computing $R(\mathbf{J})$ for nonlinear fitness functions can also be done, but the number of potentially nonzero Walsh coefficients becomes large very fast since it is exponential in the length of the genotype. This, of course, is no different from most Walsh basis computations.

locus	$S(\mathbf{J})$	$C(\mathbf{J})$	$R(\mathbf{J})$
1	0.5000	0.2886	1.7320
2	0.2500	0.5204	0.4803
3	0.1250	0.5636	0.2217
4	0.0625	0.5739	0.1088
5	0.0312	0.5765	0.0542
6	0.0156	0.5771	0.0270
7	0.0078	0.5772	0.0135
8	0.0039	0.5773	0.0067
9	0.0019	0.5773	0.0033
10	0.0009	0.5773	0.0016

Table 4.3: Signal, noise, and SNR by locus for the identity problem.

function, it may be expressed in the Walsh basis as shown by Equations 4.18 and 4.19. Doing so results in the first-order Walsh coefficients taking the values

$$w_{2^j-1} = -\frac{1}{2^j}. \quad (4.23)$$

Substituting these values for the Walsh coefficients in Equations 4.20 and 4.21 produces

$$S^2(\mathbf{J}) = \frac{1}{2^{2j}} \quad (4.24)$$

and

$$C^2(\mathbf{J}) = \sum_{k \neq j} \frac{1}{2^{2k}} \quad (4.25)$$

for squared signal and squared noise, respectively, where j is the locus fixed (held constant) by partition \mathbf{J} and may range from 1 to l . Finally, the resulting expression for the SNR for the domino convergence demonstration problem is

$$R(\mathbf{J}) = \sqrt{\frac{\frac{1}{2^{2j}}}{\sum_{k \neq j} \frac{1}{2^{2k}}}}. \quad (4.26)$$

Table 4.3 shows the resulting values of signal, noise, and SNR for the 10 most significant loci, $j = 1, \dots, 10$. As can be seen, signal strength and SNR drop off exponentially with locus position, while the noise level rises asymptotically to a constant.

It is tempting to view $R(\mathbf{J})$ as the expected proportion of selection events decided by the value of the allele at the locus fixed by \mathbf{J} (as was done in the initial convergence

window width analysis of Section 4.2), so that the first seven loci are likely to show some convergence during generation 1. This would be a good match to observed domino convergence as shown in the 0.01 mutation rate run of Figure 4.1, and these results would be in reasonably close agreement with both the earlier analysis of initial convergence window width and the empirical data. However, the SNR is not a direct measure of expected net convergence, as can be seen by the fact that its value for locus one is greater than 1.

It will be argued in Chapter 6 that the SNR is, in fact, closely related to GA convergence, though establishing its exact connection to expected convergence is left to future research.

4.5 Mutation and Convergence Stall

GA simulation runs are generally performed using a mutation rate that does not change over time. However, both from the previous analyses and from the empirical runs it is clear that convergence stall cannot be avoided whenever a nonzero mutation rate is used on a monotone problem possessing a sufficient number of loci. Thus, the common practice of using fixed mutation rates generally leads to convergence stall. Further, any rigorous GA convergence proof must treat this fact.

As has been noted by others (Sirag & Weisser, 1987; Wilson, 1987; Schraudolph & Belew, 1990; Belew et al., 1990), a solution to this problem is to allow the mutation rate to decay in over time, as is often done with artificial neural network model parameters (for example, Kohonen's (1988) topological map model) and simulated annealing (Kirkpatrick et al., 1983). Mutation decay would gradually reduce the disturbance within the converged region, either as a function of time or some measure of population diversity, thereby extending the stall point. In the limit as the mutation rate approaches zero, the convergence stall point approaches infinity and ceases to be a problem.

Chapter 5

Evolutionary Network Design (END)

Faced with a particular problem to be solved using an artificial neural network (ANN), the practitioner seeks to choose an optimal or near-optimal ANN architecture. Finding such an architecture will be called the *network design problem*. Although a number of approaches providing partial solutions to the network design problem are available, especially for the more popular ANN models, which architectures are optimal and how to find them is generally an open question. Since nature used evolution to design brains, using evolutionary search to design network architectures seems an obvious approach to explore — *evolutionary network design* (END).

Experience gained using GAs to evolve networks solving the contiguity problem is presented, both to illustrate how the fitness variance population sizing calculation of Section 2.3 may be adapted to handle the fitness function noise resulting from non-deterministic fitness functions and as an interesting GA application in its own right. The fundamental thesis underlying the work presented is that GAs can search the space of network architectures effectively.

Section 5.1 presents a prototypical ANN model and then reviews the aspects of model architectures of relevance to END. Section 5.2 considers some aspects of the network design problem ANN practitioners face. Section 5.3 reviews previous END work. Section 5.4.1 introduces a program called GAND, genetic algorithms for network design, giving an overview of GAND's design, data structures, and algorithms. Section 5.4.2 defines the contiguity problem and why it was chosen as an END test problem. Section 5.4.3 describes the training and testing data sets used, while Section 5.4.4 presents

the coding used by GAND. Section 5.5 presents a series of GAND runs performed in attempting to solve the test problem. Section 5.6 extends the population sizing work of Section 2.3 to include fitness function noise. Finally, Section 5.7 discusses the END work, with Section 5.7.1 considering what normal form representations can be useful and Section 5.7.2 discussing opportunities for additional END research.

5.1 Artificial Neural Networks

Artificial neural networks (ANNs) may be viewed as distributed computational systems based on highly abstracted models of biological neural networks. Most ANNs have nodes and connections corresponding to the neurons and synapses of biological neural nets. Usually some of the nodes are designated as inputs and others as outputs. ANNs may be thought of as performing a transformation by taking an input, or series of inputs, and producing an output, or series of outputs. They may also be viewed as a class of biologically inspired artificial intelligence capable of classifying and manipulating patterns.

The field of artificial neural networks, also known by the terms connectionism and parallel distributed processing, has a history stretching back to the middle of this century (McCulloch & Pitts, 1943; von Neumann, 1987). During the last decade, however, an ANN renaissance has occurred, resulting in dozens of distinct ANN models (Williams, 1987; Torras i Genis, 1986; Tesauro, 1986; Sun et al., 1988; Rumelhart et al., 1985; Rumelhart & Zipser, 1985; Carpenter & Grossberg, 1986; Hestenes, 1986; Barto et al., 1983; Fukushima, 1981; Fukushima et al., 1983; Hecht-Nielsen, 1987; Hinton et al., 1984; Hopfield, 1982; Kosko, 1987b; Kosko, 1987a; Linsker, 1988; Granger et al., 1989; Pearlmutter & Hinton, 1986; Pellionisz & Llinas, 1982; Pineda, 1987; Daugman, 1988; Kohonen, 1988; Moody & Darken, 1988; Foldiak, 1989; Plumbley & Fallside, 1988; Klopff, 1987; Reeke & Edelman, 1987; Specht, 1988; Scofield, 1988; Peterson & Anderson, 1987; Chua & Yang, 1988; Marks II et al., 1987; Lansner & Ekeberg, 1989). Some of these models closely follow the biology of nervous systems while others do not. However, all

of the models share a few unifying characteristics. For example, all of the models share some notion of node and connection. A *node* may be thought of as either an agent performing a (usually simple) computation or the place where the computation is performed. Generally, nodes produce an output, usually a single scalar value. Nodes are linked by *connections*, channels over which information may pass between nodes. Connections are unidirectional, with bidirectional connections implemented as a pair of unidirectional connections. In addition to the basic ideas of node and connection, most models share some notion of a scalar model parameter associated with each connection. Such *connection weights* usually serve to scale, or weight, values passing along the connection. A collection of all connection weights for connection to a particular node may be viewed as a vector, \mathbf{w} . Figure 5.1 shows a node from the resulting generic model. Often the computation performed by the node is the inner product of the node's weight vector, \mathbf{w} , and the node's input values, \mathbf{x} , followed by a non-increasing, bounded, nonlinear transformation, \mathbf{f} , so that the node's output is

$$O = f\left(\sum_{i=1}^n w_i x_i\right), \quad (5.1)$$

where n is the number of input connections to the node.

A *network architecture* is both a description of how many nodes there are, what computational model each node employs, and how the nodes are interconnected, including which nodes are network inputs and which are outputs, and the adjective denoting such a network. A commonly used network architecture is the *layered feedforward network*. It consists of layers of nodes ordered from input to output, with adjacent layers completely connected from the layer nearer the input to the layer nearer the output. Architectures which possess feedback connections are *recurrent networks*. Another dimension along which network architectures may be characterized is their degree of modularity. For example, a feedforward network may be viewed as a connected feedforward sequence of single-layer network modules. Alternately, smaller networks may be connected to form larger networks, as described by Waibel (Waibel et al., 1988; Waibel, 1989) and Leen (Leen et al., 1990). Yet another classification dimension might be degree of regularity.

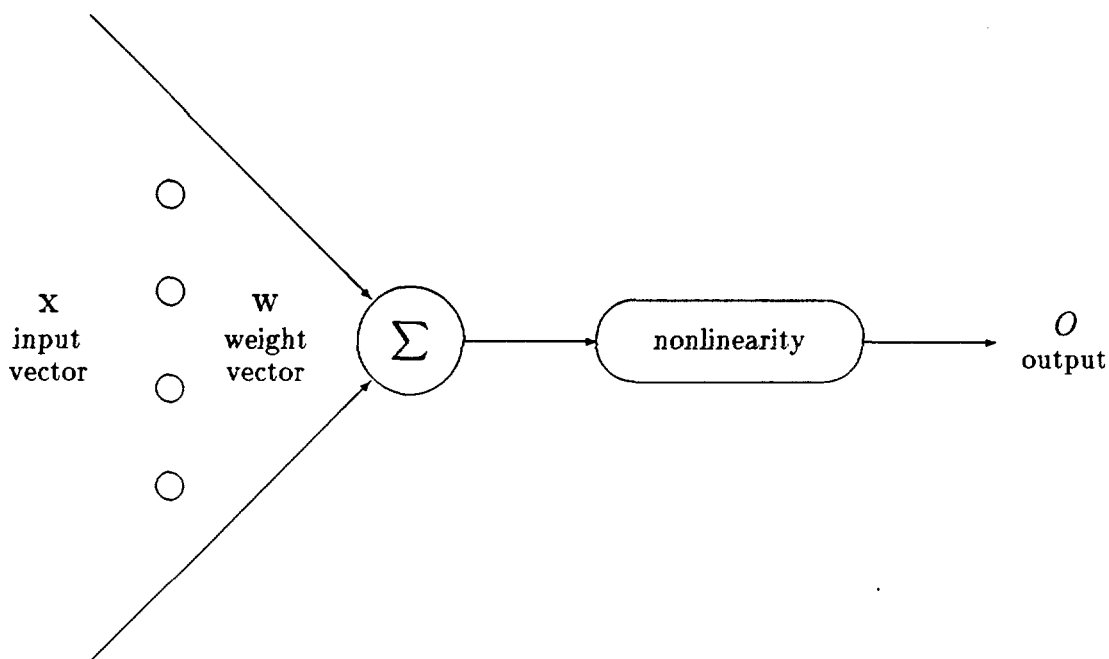


Figure 5.1: Generic artificial neural network node.

For example, although it has a large amount of randomness in the details of its structure, the Lynch-Granger pyriform cortex model (Granger et al., 1989; Lynch et al., year unknown; Granger et al., 1987) also has a high degree of regularity.

The degree of homogeneity of the model is still another network characterization dimension. Networks containing only a single kind of node will be called *homonets*, while networks containing more than one kind of node will be called *heteronets*. For example, Hopfield networks (Hopfield, 1982) and back-propagation networks (Rumelhart et al., 1985) are both homonets, but a back-propagation network whose outputs are connected to portions of a Hopfield network is a heteronet. Heteronets may be viewed as a collection of interconnected homonet modules. Only a very few combinations of network models have been tried. Examples include principal component networks used to perform data compression as a preprocessing step to using a standard back-propagation classifier network (Sanger, 1989a; Sanger, 1989b; Sanger, 1990; Leen et al., 1990), among others

(Ritter, 1989; Holdaway, 1989, for example). Waibel's time-delayed neural network modules (Waibel et al., 1988; Waibel et al., 1989) might be considered a heteronet example, since although only feedforward back-propagation modules are used, the output nodes are manually configured to compute an average (they do not learn) and thus constitute a second kind of node. Of course, not all heteronet combinations make sense, since some models are fundamentally incompatible with most other models. For example, models using spike trains to pass information between nodes (Dress, 1987b) or models using hysteresis to code information (Hoffmann & Benson, 1986) are not directly compatible with models passing scalar outputs.¹ However, since most kinds of model neurons output a single scalar value, incompatible models are rather the exception.

In addition to their architecture, ANN models may vary in a number of other respects, including their treatment of time (for example, discrete versus continuous), degree of precision used or required, locality of information used by each node, and learning mechanism. With respect to time, each node may function synchronously (for example, traditional feedforward nets (Rumelhart & McClelland, 1986)) or asynchronously (for example, Hopfield nets (Hopfield, 1982)). With respect to precision, many researchers use full 64-bit floating point arithmetic for network calculations, although use of integer arithmetic to increase simulation speed has become more common. Baker and others have empirically demonstrated that different models have different precision requirements. For example, the Lynch-Granger pyriform cortex model requires only 4 to 6 bits of precision (Means, 1989; Hammerstrom, 1989) while back-propagation typically requires 12 to 16 bits (Baker, 1990; Baker & Hammerstrom, 1989a; Baker & Hammerstrom, 1989b; Hammerstrom, 1989; Personnaz & Dreyfus, 1988). Finally, some models are Boolean, effectively using one-bit precision.

Most ANN models require that the computations performed use only information locally available to the nodes. This is an especially important design constraint for

¹Of course, ANN models converting from one modality to another may be possible, effectively interfacing otherwise incompatible models.

VLSI implementations because VLSI has limited interconnect capability (Bailey & Hammerstrom, 1986; Hammerstrom et al., 1989; Bailey & Hammerstrom, 1988; Rudnick & Hammerstrom, 1988a; Rudnick & Hammerstrom, 1988b; Hammerstrom, 1986; Hammerstrom, 1988; Rudnick et al., 1987; Hammerstrom et al., 1987) and is usually considered a requirement for biologically plausible learning rules. Nevertheless, some models do make use of global information. For example, performing a winner-take-all computation among a collection of nodes is global to that collection,² although local implementations are possible.

ANN models can use a variety of learning mechanisms. Learning rules may be grouped into two broad classes: unsupervised and supervised. In unsupervised learning, the network is presented input exemplars without any feedback or guidance as to the merit of its resulting output (Rumelhart & Zipser, 1985; Kohonen, 1988; Sanger, 1989b; Pearlmutter & Hinton, 1986; Plumbley & Fallside, 1988). In supervised learning, some form of feedback is provided. Two kinds of supervised learning may be distinguished: reinforcement and target. In reinforcement learning, a scalar feedback signal is provided indicating the degree to which the automaton's output is 'good' or 'bad' (Williams, 1987; Barto, 1985). In target learning, the automaton's desired output is provided as feedback. The classic example here is the error back-propagation learning rule (Rumelhart et al., 1985; Rumelhart et al., 1986).

Other learning models abound. In the original Hopfield spin-glass model (Hopfield, 1982) an outer product was used to pre-compute static weights which were then loaded into the network. This is an example of a static, non-learning network with fixed, pre-computed weights. The weights can also be learned using a Hebbian learning rule (Linsker, 1988; Leen et al., 1990) in which each weight is incrementally moved toward the product of the pre-synaptic and post-synaptic activities. Hebbian learning is biologically

²Even with global computations, relatively efficient mechanisms may still be available to perform the computation. For example, Lazzaro has an $O(n)$ technique for performing the winner-take-all computation in VLSI (Lazzaro et al., year unknown). Further, each implementation technology has its own individual set of constraints; for example, many of the interconnect problems of planar VLSI may not be problems for optical implementations.

plausible and is an example of unsupervised learning.

5.2 Network Design Problem

Because artificial neural networks are both diverse and complex, a practitioner wishing to solve a particular problem using ANN technology is immediately faced with a bewildering choice of possible architectures, especially when considering heteronets. Back-propagation will serve as a case study. Back-propagation has as advantages that it tends to be both simple and robust. Even so, many architectural and algorithmic questions must be answered: How many hidden layers should be used?³ How many nodes should be in each of the hidden layers? Should interconnections be limited and if so how? Should feed-through connections jumping over intermediate layers be used? Should learning algorithm enhancements be used, and if so what kinds — for example, momentum; various forms of learning rate adaptation; fast weights; gradient estimation by full training set batch weight updates or by partial training set weight updates, and if partial, how many exemplars should be used per weight update; conjugate gradient descent (Barnard & Cole, 1989); etc.? How should the various non-weight parameters be set or varied? Hidden among all these choices and parameters are likely to be problem-specific networks of superior performance.⁴ Finding which are the superior networks will be called the *network design problem*; it includes all aspects of determining a completed network except setting the weights.

Each network design decision has consequences in terms of learning speed, performance (meaning how good an answer is provided), generalization ability, feedforward execution speed, and ease of implementation. The choices made can interact in complex and poorly understood ways. As a consequence many ANN practitioners simply use a standard back-propagation model — one hidden layer, full inter-layer interconnect, no

³Of course, it is known that a single hidden layer in a non-linear, feedforward network can approximate virtually all functions arbitrarily well (Hornik et al., 1990). Nevertheless, engineering practice has shown multiple layers may be desirable (Waibel et al., 1988; Waibel et al., 1989; Waibel, 1989, for example,).

⁴What constitutes superior performance is generally user defined and will vary both from user to user and from problem to problem.

feed-through connections, static momentum, static learning rate, no fast weights, single exemplar gradient estimation, and no conjugate gradient techniques. The momentum and learning rate terms are either set to nominal values (guessed at) or set by trial and error during a few simulation runs. The latter often amounts to doing a manual line search. Again, the number of hidden nodes is usually determined by similar trial and error methods. All told there is no assurance the resulting solution is particularly good in terms of the space of all possible back-propagation networks. Essentially, a trial-and-error search is performed over the space of back-propagation architectures.

In addition, of course, there are now many dozens of different ANN models in the literature, often having significant followings and useful applications. Most of these models, like back-propagation, have their own version of the network design problem. And since each model's design space is usually multidimensional, the space of binary heteronets (heteronets consisting of two homonets) is the Cartesian product of all these models. Likewise, heteronets containing many different, possibly fine-grained homonets are also possible. All in all, this is a large space of network architectures to be searched when seeking an ANN solution for a problem. It's no wonder only a small portion of heteronet space has been explored.

The network design question then, is: "Given a specific problem to be solved and criteria for measuring success, what ANN models and architectures provide superior solutions?" One general approach is to search the space of network architectures, preferably in a systematic way that can handle all dimensions of network design space, both continuous and discrete, both homonets and heteronets. The approach taken here is to use genetic algorithms to search the space of ANN architectures — *evolutionary network design* (END). In the next section previous END work is surveyed.

5.3 Previous Work

A number of techniques have been proposed to tackle various aspects of the network design problem.⁵ Most of the techniques address optimizing homogeneous back-propagation networks. These include additive techniques where nodes or other components are added to an existing back-propagation network (Honavar & Uhr, 1989; Waibel, 1989; Fahlman & Lebiere, 1990; Gutierrez et al., 1989; Ash, 1989) and subtractive techniques where nodes or other components are removed from an existing back-propagation network (Rumelhart et al., 1986; Hinton, 1987; Fahlman, 1989a; Wieland and Leighton, 1988; Rumelhart, 1988; Kruschke, 1989; Baum et al., 1988; Keeler, 1986; Mozer & Smolensky, 1989; Le Cun et al., 1990a; Le Cun et al., 1990b). However, none of these techniques provide a mechanism able to select the better networks from among the broad range of possible networks. Each places considerable limitations upon the range of networks considered, usually because the potential solutions considered have been limited in advance to a narrow class of models or architectures.

Evolutionary search techniques, and genetic algorithms in particular, need not be limited in advance to a particular class of models or architectures. They are capable, in principle, of handling homonets, heteronets, and the various model parameters associated with many ANN architectures. Any network architecture which can be specified in the genetic representation can be included in the search domain. Thus, GAs are a network architecture optimization technique of general applicability.

Only recently has work begun exploring evolutionary network design (END) (Rudnick, 1990). As will become apparent, a comprehensive, unifying treatment providing a foundation for END and illuminating fundamental issues is needed. The primary works motivating and using END are surveyed below.

⁵For example, at the 1989 Neural Information Processing Conference, three workshops touched on various aspects of the problem — Scott Fahlman's "Neural Network Learning: Moving from Black Art to a Practical Technology," Peterson and Snyder's "Neural Networks and Optimization Problems," and Davis and Rudnick's "Neural Networks and Genetic Algorithms".

5.3.1 Dress's Artificial Insect

Most of the work reported in the literature has appeared during the last two years. A notable exception is the work of Bill Dress (Dress & Knisley, 1987; Dress, 1987a). Dress used a 50-dimensional genetic code to specify an 'insect,' including rudimentary sensory channels and nervous system. Using a standard genetic algorithm, he searched the 50-dimensional search space to find phenotypes performing well in the artificial environment. For example, touching a 'predator' with its feelers was penalized by reducing the fitness of the artificial animal. An indication of the potential power of the GA design technique occurred when insects evolved which moved backwards, thereby avoiding the "predator touched with feelers" penalty. That automata might evolve which backed up to predators was a complete surprise to Dress and represents a fitness function design oversight. The anecdote provides, however, an effective demonstration of how END might overcome biases implicit in human ANN design.

Dress appears to be the first to use evolutionary search for ANN design. But since sensory apparatus, morphological attributes, and an interactive environment were included in addition to the ANN, Dress's work may be more properly classified as an early example of artificial life.

5.3.2 Mjolsness's Recursive Network Definition

A much different approach to network architecture specification has been taken by Eric Mjolsness et al. (Mjolsness et al., 1988a; Mjolsness et al., 1988b; Mjolsness et al., 1988c; Mjolsness et al., 1987). A set of growth rules are first defined using a recursion relation having a fixed number of coefficients. Each possible recursion relation defines a family of successively larger neural networks. Each weight of an ANN so generated is specified through the recurrence relations and is ultimately derived from a combination of the weights in the original progenitor net and the parameters of the recurrence relations. Training a family of such networks is accomplished by using simulated annealing to optimize (over the space of recursion relation parameters) the performance of the family's

smaller members.

The continuous code problem was used as a test problem. The input is a unary number. The desired output is a compressed coding, or representation, of the input arranged so that similar inputs map to similar outputs where Hamming distance is used as the similarity metric. Although perhaps a rather simple problem, Mjolsness showed that recursively defined networks more than two orders of magnitude larger than the small, trained networks still performed well on the task.

Mjolsness's approach may have the potential to be more generally applicable. First, it must be shown that most potential ANN solutions can be cast using a recursive definition approach. Second, it must be shown that Mjolsness's genetic neural networks will work for most other problems. Although Mjolsness continues this work, its present applicability appears limited.

5.3.3 Hinton & Nowlan's Work

Hinton and Nowlan (1986; 1987) performed a simulation supporting the notion that learning by an individual can synergistically aid evolutionary learning (Baldwin, 1896; Smith, 1987; Waddington, 1942). They defined a 20-bit binary string problem in which each bit must be correctly specified to solve the problem. Their genetic code consisted of strings of 20 ternary digits in the alphabet $\{0, 1, ?\}$. Digits 0 and 1 are hard-coded in the phenotype and digit '?' is settable. Learning during the lifetime of an individual phenotype consists of randomly setting the '?' bits of the genotype with each individual having a fixed number of such trials. If evolutionary search alone was used, the fitness function would be zero everywhere except at the exact solution point — the needle in the haystack problem. But when learning by the individual is added, the fitness function is modified; it still peaks at the solution point, but has Gaussian-like tails trailing away as Hamming distance from the solution increases. As pointed out by Belew (1989), who has carried this work further, the effect is to add a basin of attraction around the solution point.

Although the particular example problem used is simple and somewhat contrived,

and although END is not directly touched upon, this work illustrates an important interaction between evolutionary search and individual learning. It looks through the END telescope the other way around, viewing learning by the individual as aiding evolution instead of the usual END view of evolution aiding learning in the individual. It suggests the potential for rich interactions between evolutionary population learning and neural network learning in individuals.

5.3.4 Miller's Connection Matrix

Miller, Todd, and Hegde (Miller et al., 1989; Todd, 1988) combined Grefenstette's GENESIS (Grefenstette, year unknown) genetic algorithm C program with Rumelhart and McClelland's (1988) back-propagation program. Their genetic representation consists of a binary string encoding the network's connection matrix, where the network contains a fixed number of nodes. Each connection is represented by a 1-bit; absent connections are represented by a 0-bit. Thus for n nodes, an n^2 -bit binary string specifies a network. Although recurrent connections are possible in such a representation, all recurrent connection specifications are ignored so that a simple back-propagation learning rule may be used. Also, note that nodes having no connections are in effect not present.

Their END system was tested on three problems: exclusive-or, four-quadrant, and pattern copying. For exclusive-or, a five-by-five interconnection matrix representation was used. Thus, all feedforward networks having five or fewer nodes were included in the search space. Two nodes were designated as network inputs and one as output.

The four-quadrant problem consists of dividing the region of \mathbb{R}^2 bounded by (0,0) and (1,1) into four equal-sized quadrants. Given a real-valued two-dimensional input vector, the task is to classify the vector as to whether it falls into an even quadrant or an odd quadrant. Thus, the four-quadrant problem may be viewed as a real-valued generalization of the exclusive-or problem. For the four-quadrant problem, a seven-by-seven interconnection matrix representation was used, with two nodes dedicated as inputs and one node as output. Thus the search space consisted of all feedforward networks having two inputs and one output.

The pattern-copying problem is simply the identity mapping over binary strings. Although not explicitly stated in their paper, 20 units were apparently provided for in the interconnection matrix genetic code, with 10 designated as inputs and 10 as outputs.

For all three problems starting with randomly generated networks, architectures solving each problem were evolved within a few generations, while the entire population performed well within 5 or 10 generations.

Any work using END must deal with the issue of *developmental specificity*, a term coined by Miller et al. to characterize whether the genotype maps directly to the phenotype or a significant developmental mechanism is present. They used the strongest possible developmental specificity for their genetic codes — a connection matrix where the presense or absence of each connection is explicitly expressed in the genotype. In their words (Miller et al., 1989, pages 380-381):

Weak specification representation schemes use relatively abstract genetic ‘blueprints’ that must be translated through some ‘developmental machinery’ to yield a network phenotype, e.g., (Harp et al., 1989a). Such schemes may be good at capturing the architectural regularities of large networks rather efficiently. However, they necessarily involve either severe constraints on the network search space, or stochastic specification of individual connections. For example, a weak specification scheme could represent whole network layers in single genes, facilitating the recombination and evaluation of large, highly regular networks, but precluding detailed connection design.

Strong specification schemes, which interpret genes more directly as individual connections, are good at capturing the connectivity patterns within smaller networks very precisely and deterministically. Such a scheme could facilitate the rapid evolution of finely optimized, compact architectures. A variety of moderate specification schemes are possible.

We chose a strong specification scheme to gain greater resistance to human design biases for crisply articulated network layers, localist representations,

and easily interpretable processing strategies, all of which can creep into weak specification schemes. A strong specification scheme may facilitate the rapid generation and optimization of tightly pruned, interesting designs that no one has hit upon before. We hope that the inspection of such streamlined designs will hone our intuitions about what weak specification schemes might work well for larger network designs.

In Section 5.7.1 we will return to consideration of representational issues.

5.3.5 Harp's Area Blueprint

Nearer the weak end of Miller's developmental specificity spectrum is the GENESYS program used by Harp, Samad, and Guha (1989a; 1989b; 1990). Here again, only feedforward networks trained by the error back-propagation learning rule were allowed.

Their search space is restricted to networks expressible in terms of generalized layers called *areas*. Each area appears in the genetic representation as a fixed length area parameter specification (APS) along with associated projection specification fields (PSFs). APSs contain parameters specifying number of nodes in the area, dimensionality of the area, and relative size of each dimension. Each APS may have one or more PSFs defining projections to other areas. Each PSF contains parameters for connection density, initial learning rate, rate of exponential decay of the learning rate, target address, addressing mode (absolute or relative), and X, Y, and Z radius parameters allowing for local receptive fields in up to three dimensions. Many of the numerical parameters are coded \log_2 , allowing a small number of values to span a large region of the search space at a cost in the graininess of the representation. Gray coding is generally used so that adjacent values are represented by adjacent codes, presumably to avoid Hamming cliffs in the search space (Caruana & Schaffer, 1988).

In their original work, back-propagation network architectures were designed for two problems, exclusive-or and a four-by-eight-pixel font digit recognition problem. Starting with random networks, in each case GENESYS produced solution networks.

For very small networks, the size of Harp's weakly specified, coarse-grained genetic representation will be larger than Miller's strongly specified, fine-grained representation; thus Harp's search space will be larger than Miller's. However, at some point as the number of nodes in the network grows, and provided the desired networks can be expressed in Harp's representation, Harp's search space becomes smaller than Miller's, and thus may lead to a more efficient search.

5.3.6 Kitano's Graph L-System

The only work other than Harp's area blueprint to use a developmental genotype-to-phenotype map of significance is Kitano's (1990) graph L-system work. He coded a graph grammar in the genotype and then used its production rules to grow a network architecture from the grammar's start symbol. An L-system (Lindenmayer, 1968; Lindenmayer, 1971) was used as the graph grammar. Kitano used the encoder-decoder problem as his test problem, evolving a variety of both four-by-four and eight-by-eight networks. He compared the direct, or connectivity matrix, representation and graph L-system representation, concluding that graph L-system encoding scaled better. However, it is hard to interpret the significance of these results since only 10 training epochs were used, the number of hidden nodes was always at least twice the number of input or output nodes, and generalization was never tested.

5.3.7 Other Related Work

Several others have done work either directly or indirectly related to END. Schaffer and Caruana (Schaffer et al., 1990) used a fine-grained, strong developmental specificity genetic code similar to that of Miller, et al. Wilson (1990) has successfully used evolutionary search for design of the sparsely connected conjunctive detector nodes in Rosenblatt's (1962) classic perceptron. Also, Peter Hancock (1990) has successfully used END on a face recognition task.

Dodd (1989; 1991) used END to code for spread nets on a Dolphin vocalization recognition task similar to those used by LeCun (Le Cun et al., 1990a; Le Cun et al.,

1990b) for handwritten character recognition. Dodd's networks were relatively large compared to all the other END work cited. Thus it provides an important example suggesting that END can scale to larger problems. Of course, a large network size alone is not sufficient to base scaling conclusions upon, since the problem could potentially be solved using a much smaller network. Dodd used a relatively coarse-grained, weak developmental specificity in his genetic code, which presumably allowed his END to search a space of large networks efficiently by using a compact genetic code having a smaller search space.

5.4 GAND Overview

In this section GAND (genetic algorithms for network design), a C program for doing END, is introduced. GAND has been used to evolve networks solving the contiguity problem. Section 5.4.1 presents the design of the GAND program. Section 5.4.2 presents the contiguity problem and discusses why it was used as a GAND test problem. Section 5.4.3 presents a description of the training and testing data sets used during GAND evaluation of genotype fitness. Finally, Section 5.4.4 presents the genetic representation used by GAND.

5.4.1 GAND Design

The GAND program which has been implemented is really several independent programs fused together. Conceptually GAND includes a simple GA program and one or more ANN simulation programs. The logical design is shown in Figure 5.2. The top level GAND module implements the user and system interfaces, initialization, and overall control functions. It calls the GA module to perform each GA generation. The GA module, in its turn, calls the evaluation module to evaluate each genotype (the genotype specifies the ANN architecture). The evaluation routine calls three modules: a network construction module, a training and test data set generation module, and then the appropriate ANN model module to train and test the network.

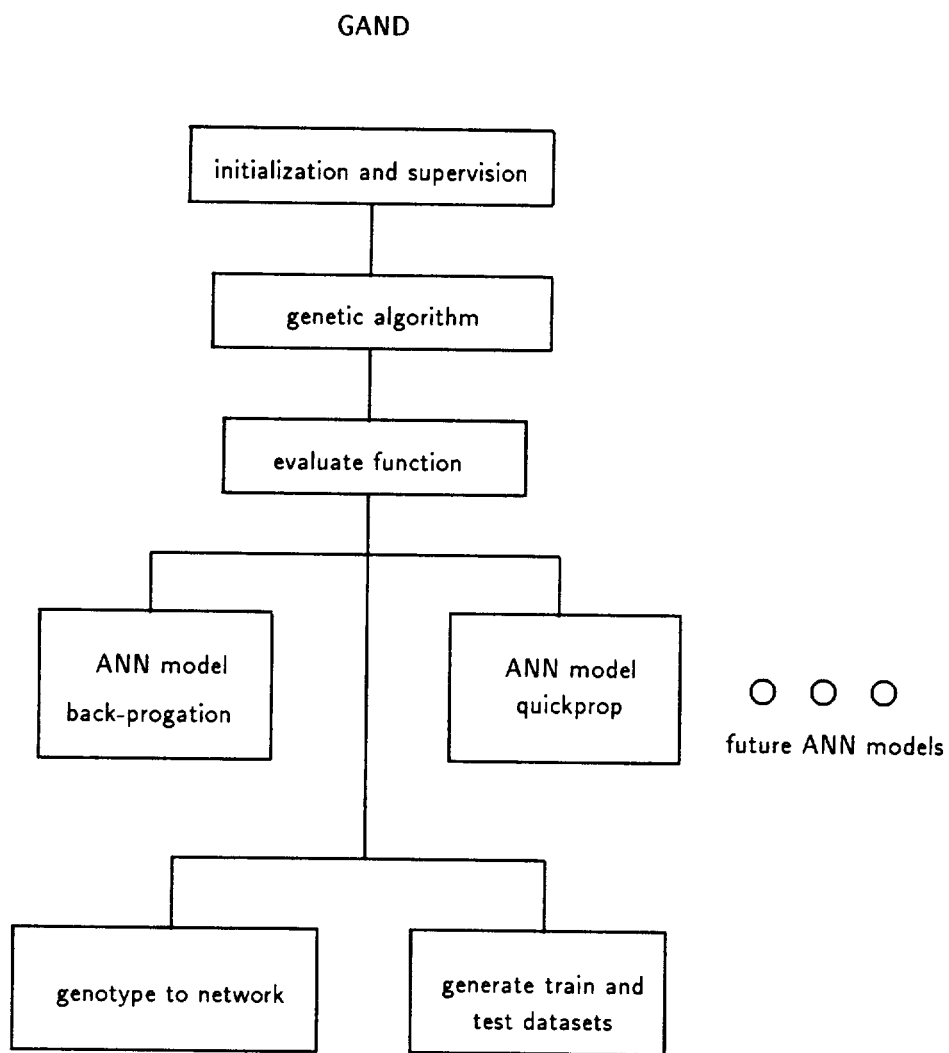


Figure 5.2: GAND design.

In keeping with sound software engineering practices, a modular design philosophy was employed to allow easy maintenance, modification, and addition of independent modules. Readily modifiable components include the network construction mechanism, the functional details of each ANN model, which ANN models are available, and the particulars of the GA, such as the kinds of selection, crossover, and mutation employed.

The GA module implements the simple GA described in Section 1.1. A single generation consists of the GA processing the old population into a new population (the next generation) by repeatedly selecting a pair of parents, performing crossover and mutation operations, and then evaluating the resulting genotype by a call to the evaluation module. The GA module currently employs tournament selection, a two-point crossover operator, a bitwise mutation operator, and a fixed-length, haploid⁶ binary string genotype.

The genotype-to-network (GTN) module converts a genotypic code into an initialized ANN data structure suitable to be trained by the module implementing the selected network model. In the current GAND implementation, the GTN module has a separate code segment, and uses different data structures, for each network model supported. This is largely a result of the fact that pre-existing ANN network simulation programs employing incompatible data structures were cannibalized for the two ANN models currently supported.

Currently back-propagation (Rumelhart et al., 1985) and Scott Fahlman's Quickprop (1989b) ANN algorithms are implemented in GAND. The back-propagation module was adapted from McClelland and Rumelhart (1988). The Quickprop module was adapted from Terry Regier's (1990) C language version of Scott Fahlman's Quickprop lisp program. Each of the two network modules currently in place have their own set of data structures.

An input file specifies the more frequently modified GA parameters, the ANN models, and overall GAND control. Other parameters exist, such as the GA population size and

⁶Most living things have their chromosomes organized as pairs. During the meiosis stage of sexual reproduction, the diploid pairs of chromosomes present in somatic cells are separated into the haploid sets of nonpaired chromosomes present in mature germ cells. In a genetic algorithm context, haploid simply means unpaired chromosome.

which pseudo-random number generator is to be used, that are not included in the GAND input file and thus require recompilation to modify. Finally, because repeated creation, training, and testing of ANNs is computationally time consuming,⁷ a checkpoint and restart facility has been added, allowing the GA to be continued from any previous checkpoint.

5.4.2 The Test Problem

The contiguity problem was selected as the GAND test problem. It consists of learning the binary function $f : (0, 1)^{|I|} \mapsto \{0, 1\}$,

$$f(I) = \begin{cases} 0 & \text{if } k \leq k_0, \\ 1 & \text{if } k > k_0 \end{cases}, \quad (5.2)$$

where k is the number of clumps (independent, continuous runs) of 1s in the binary vector I . The constant, k_0 , functions to threshold f on the number of clumps of 1s in I ; that is, it determines whether the value of f is 0 or 1, based on whether the number of clumps of 1s in I is above or below the threshold. For example if $k_0 = 2$, vector $I = 1100011001$ has three clumps of 1s and would result in $f = 1$, while vector $I = 0011101000$ has only two clumps and would produce $f = 0$.

The contiguity problem has several advantages suggesting its use as a GAND test problem. First, it is simple enough to be computationally manageable. Of equal importance, Solla (1988) has characterized an aspect of single hidden layer back-propagation network architectures for the contiguity problem which strongly relates to the network's generalization ability. Thus, network generalization performance makes an excellent criteria for network architecture optimization by END.

Figure 5.3 shows the input to hidden node connections of a network designed by hand by Solla to solve the contiguity problem. Both connection and associated weight is set manually. All hidden nodes connect to a single output node with +1 weights. Hidden

⁷The GAND program takes about two days to do a typical run, such as those of Section 5.5.1, on an IBM RS-6000/320.

Hidden Node	Hidden Node Receptive Field									
	Input Nodes									
	1	2	3	4	5	6	7	8	9	10
1	+
2	+	-
3	.	+	-
4	.	.	+	-
5	.	.	.	+	-
6	+	-
7	+	-	.	.	.
8	+	-	.	.
9	+	-	.
10	+	-

Figure 5.3: An inter-layer connection matrix representation of the first connection layer of a feedforward network solving the contiguity problem. First row codes connections to first hidden node from 10 inputs; second row codes inputs to second hidden node from the same 10 inputs; and so forth. A + means the connection is present and the associated weight is +1. A - means the connection is present and the associated weight is -1. A '.' means connection is absent. The hidden layer to output connections (not shown) consists of each hidden node connecting to a single output node with a +1 connection weight. The bias weights (from a unit with constant activation of +1) are -0.5 to each hidden node and $-(k_0 + 0.5)$ to the output node.

nodes have -0.5 biases, and the output node has a bias of $-(k_0 + 0.5)$. The network works by counting the number of clumps of 1s in the input field by decoding each clump's left-hand edge.

Using a problem size of $|I| = 10$, Solla showed that narrower hidden node receptive fields yield better network generalization performance up until the network can no longer learn the training set. Table 5.1 shows test data set performance, %G (generalization), as hidden node receptive field size, s , varies from 10 to 3 for successful training runs (%L = 100, meaning all training exemplars were successfully learned). The greater the receptive field size, the greater the receptive field overlap for each hidden node, until in the extreme (receptive field size of 10) all hidden nodes share the entire input string as their common receptive field — the fully connected layer's structure commonly used in back-propagation networks.

As a preliminary step toward using the contiguity problem as a GAND test problem,

s	$\%G$
10	55
9	59
8	61
7	63
6	68
5	73
4	90
3	95

Table 5.1: Solla's test data set results, $\%G$, with varying receptive field sizes, s , averaged over successful training runs ($\%L = 100$).

Solla's empirical results were replicated. Appendix A details that work, showing similar results but with minor variations.

For computational efficiency, Scott Fahlman's (1989b) Quickprop was used for the initial GAND runs instead of back-propagation. But upon investigation, Quickprop was discovered to provide the GA with a weaker fitness function receptive field width discrimination signal on the contiguity problem, an interesting observation but possibly problematical for the GA. Thus after the initial runs, back-propagation was used exclusively. Appendix B details the Quickprop versus back-propagation investigations.

5.4.3 Training and Testing Data Sets

There are 1024 possible 10-bit binary input vectors. To make these results comparable to Solla's results, only input vectors containing either two or three clumps of 1s were used, a total of 792 vectors. For each network created, trained, and tested by GAND, 100 of these 792 vectors were randomly selected for use as training exemplars and the remaining 692 were used as testing exemplars. Each input vector was paired with an output target value (a vector in general, but since there was only one output node a single value results). Output target values of 0.9 and 0.1 were used as the above and below threshold output values, respectively.

5.4.4 Genotype Representation

The choice of genetic code is one of the more important facing a GA practitioner. As pointed out by Liepins and Vose (1990), the choice of genotype representation can make the difference between a problem being fully deceptive, partially deceptive, or nondeceptive.⁸ Goldberg (1989c, pages 80-82) relates GA coding choices to underlying theoretical analytical motivations, presenting two principles.

1. **Principle of meaningful building blocks:** The user should select a coding so that short, low-order schemata are relevant to the underlying problem and relatively unrelated to schemata over other fixed positions.
2. **Principle of minimal alphabets:** The user should select the smallest alphabet that permits a natural expression of the problem.

Initially, a general feedforward representation using the feedforward portion of a full interconnect matrix representation was employed. A binary coding was used, with a 1 indicating the corresponding link was present and a 0 showing its absence. Such a code admirably adheres to Goldberg's second coding principle, but it violates the "relatively unrelated to schemata over other fixed positions" portion of his first coding principle because of the normal forms problem discussed in Section 5.7.1. This problem, intrinsic to all connection matrix based representations of back-propagation networks, is left to future research.

The genetic code was capable of representing any feedforward architecture possessing 10 input nodes, 10 hidden nodes, and a single output node. All hidden nodes had links to the output node. Thus, up to 10 hidden layers were logically possible, although each would consist of only one node. Figure 5.4 shows the coding for a network having a uniform hidden node receptive field size of two and no connections between hidden

⁸A problem and representation are deceptive if they violate the building block hypothesis. Building blocks are short, low-order schemata of above-average fitness. The schema theorem shows that the representation of building blocks in the population tends to grow over time. When building blocks do not contain the global optima, deception is present — the GA is misled. If sufficient deception is present the GA is prevented from finding the global optima.

Hidden Node	Hidden Node Receptive Field																			
	Input Nodes										Hidden Nodes									
	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6	7	8	9	10
1	1	1										
2	.	1	1									
3	.	.	1	1									
4	.	.	.	1	1									
5	1	1									
6	1	1									
7	1	1	.	.	.									
8	1	1	.	.									
9	1	1	.									
10	1	1	.									

Figure 5.4: Representation for a general feedforward architecture showing uniform hidden node receptive field size of two, where feedforward inter-hidden-node connections are allowed (but not present in the network specification shown). 1 codes for the presence of a connection and '.' codes for its absence. First row codes for presence of connections to first hidden node from 10 inputs; second row codes for the presence of connections to second hidden node from 10 inputs and first hidden node; and the last row codes for the presence of connections to the last hidden node from the 10 input nodes and the previous nine hidden nodes. The binary genotype string would consist of the concatenation of all rows, with '.'s replaced by 0s, and would therefore have a length of 145 bits.

Hidden Node	Hidden Node Receptive Field									
	Input Nodes									
	1	2	3	4	5	6	7	8	9	10
1	1	1	1
2	1	1	1
3	.	1	1	1
4	.	.	1	1	1
5	.	.	.	1	1	1
6	1	1	1	.	.	.
7	1	1	1	.	.
8	1	1	1	.
9	1	1	1
10	1	1	1

Figure 5.5: The same network as shown in Figure 5.4, but using a representation disallowing hidden nodes connecting to other hidden nodes, so that only input nodes may connect to hidden nodes. Thus, the representation consists of only the “Input Nodes” portion of Figure 5.4. This representation is only capable of coding the conventional, layer-oriented network architecture. As in Figure 5.4, each row codes for the connections to a single hidden node, with 1 denoting the presence of a connection and ‘.’ denoting the absence of a connection. The binary genotype string would, again, consist of the concatenation of all rows with ‘.’s replaced by 0s and would have a length of 100 bits.

nodes. The actual genotype depicted in Figure 5.4 may be formed by replacing ‘.’s with 0s and concatenating the rows.

A few trial runs were performed using the general feedforward representation, but they showed an increase over randomly generated networks of no more than 4 percentage points during the first 20 generations in the generalization performance measure (%G) used by Solla. Although any improvement is helpful, 4% is not enough to make GAND a useful network design tool. To further simplify GAND’s task, the general feedforward genotype representation was replaced by an inter-layer connection matrix representation coding for all possible links between the 10-input-node layer and a 10-hidden-node layer; as before, only the feedforward portion was actually used and all hidden nodes connected to the single output node. Figure 5.5 shows how the network of Figure 5.4 looks when represented using the inter-layer connection matrix representation. Note how the inter-layer connection matrix reduces the length of the coding from $l = 145$ bits to only 100 bits. Such a reduction in search space size can have a significant effect on GA performance (for example, see Section 5.5.5). All GAND results reported in the following section employed the inter-layer connection matrix representation.

The inter-layer connection matrix representation is not a normal form, which simply means that a single network may have multiple codings when using the representation. For example, any permutation of hidden node order of a particular genotype will yield identical networks with respect to the back-propagation training algorithm. Back-propagation has no notion of node order within a layer, so whether or not two hidden nodes are adjacent does not change back-propagation’s behavior. However, permuting hidden node order in a genotype string results in a very different genotype from the GA’s perspective. Multiple genotype codes for the same network results in symmetries in the search space. The question of normal forms is left to Section 5.7.1.

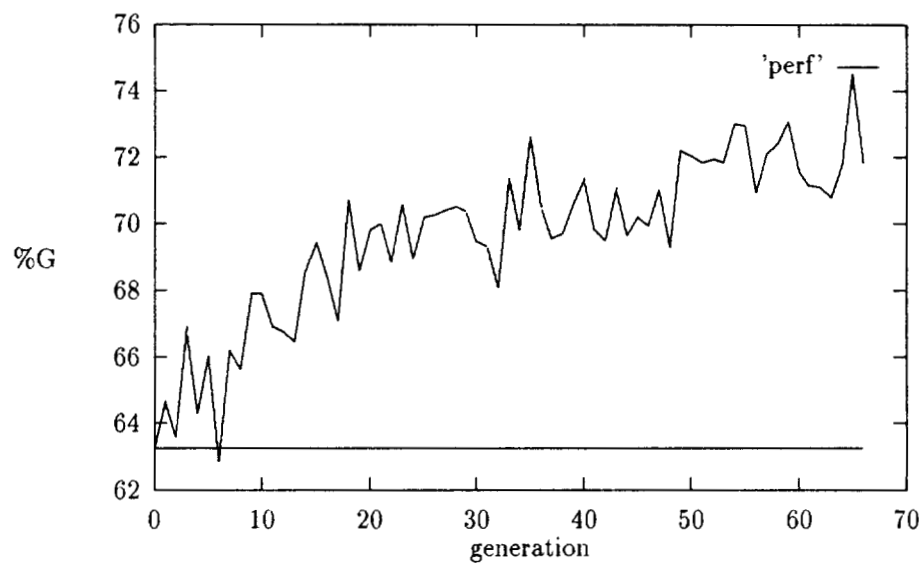


Figure 5.6: Generation number versus percent generalization performance on best initial GAND run. The horizontal line just above 63% is the performance of the randomly generated networks of generation zero.

5.5 GAND Results

Having now set the context for GAND on the contiguity problem, this section presents empirical performance results. A number of GAND experiments have been performed. In each case time histograms of average performance across the population are used to compare different experiments. It is important to realize that, in most cases, some individuals in the population performed considerably above or below the mean performance. Thus, the best individuals found by GAND during a simulation run often performed at a considerably higher level than the mean population performance.

The initial GAND runs were made on IBM RTs using Quickprop and a population size of 20. Figure 5.6 shows one of these runs relative to the performance of the randomly generated networks of generation zero (the horizontal line at the bottom of the graph). The maximum increase of 10 percentage points in the population's average generalization performance measure, a 17% improvement over randomly generated networks, is noteworthy, since it was obtained using a simple GA, a blind, reinforcement search technique given no problem-specific information whatsoever other than the scalar fitness signal.

Notice that randomly generated networks using Quickprop performed at the 63% level, a level noticeably higher than the fully connected networks, which performed at only a 55% level (see Appendix B for additional details). Upon reflection this is not particularly surprising. Decoding edges plays an important role in solving the contiguity problem, and random connectivity makes for many randomly placed, relatively narrow hidden node receptive fields which are good at detecting edges.

Despite the improvements in performance, these results are still marginal in terms of GAND being an effective tool for network design. In trying to understand why GAND did not find near-optimal networks, five hypotheses were considered. Each is a candidate problem that, when fixed, could result in GAND finding the high-performing networks. First, Quickprop's shallow discrimination slope as compared to back-propagation's steeper slope could be a problem (see Appendix B). Second, nondeterminism, or noise, in the objective function could seriously degrade the ability of the GA to shepherd its

population into near-optimal regions of the search space. Third, the population size selected may not have been large enough. Fourth, the GA may need a better objective function, such as might be obtained by incorporating a “link tax” to conserve allocation of links. And fifth, the poor performance may simply be due to too high a disruption by crossover — in effect, the schema theorem may not be satisfied. In addition each candidate problem may be acting simultaneously, so that several of them may together result in the mediocre performance observed. Of course, it is also possible that the END approach simply does not work for network designs more difficult than the relatively simple problems tackled by researchers thus far. However, such a possibility seems unlikely since nature offers existence proof after proof that evolutionary search solves tough design problems. Even so, there is no guarantee that the solution of any particular problem will be computationally tractable.

The remaining sections present a series of simulations designed to empirically characterize each of the candidate problems that may be affecting GAND performance. Section 5.5.1 presents GAND runs using the back-propagation training algorithm instead of the Quickprop algorithm. It serves to characterize GAND baseline performance. In each of the following sections, a single factor is varied from the baseline run’s performance, allowing a direct test of each candidate problem’s effect upon GAND performance. In Section 5.5.2 the effect of fitness function nondeterminism, or noise, on GAND performance is explored by replacing the default fitness function with fitness averaged over three independent training trials, effectually reducing the fitness function’s noise level. In Section 5.5.4 an alternate fitness function is tested by adding a connection cost, or “link tax,” component to the fitness function. In Section 5.5.3 population size is trebled to check for too small a population size. In Section 5.5.5 elitist selection is used to check for excessive disruption.

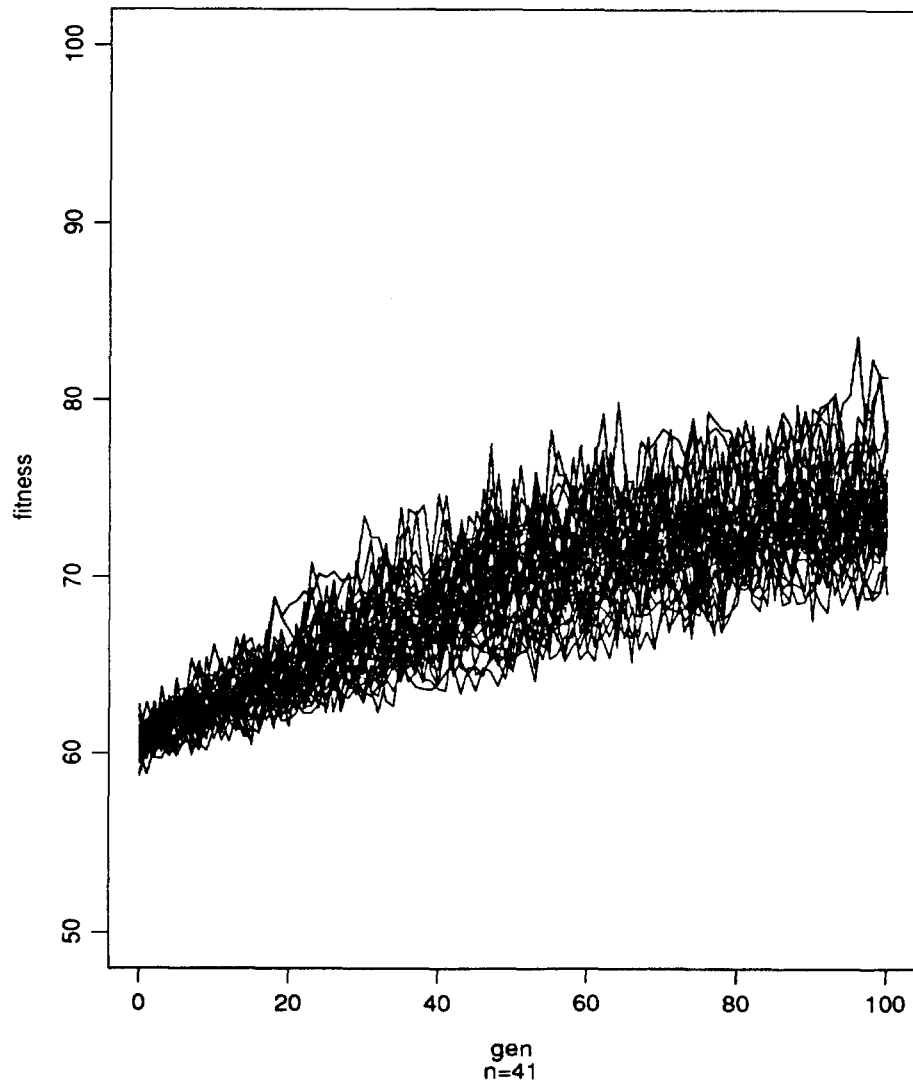


Figure 5.7: Baseline GAND back-propagation performance.

5.5.1 Back-Propagation

The first thing done was to switch from using Quickprop back to using back-propagation. As noted earlier, Quickprop provides a smaller discrimination slope than does back-propagation. Because of this, the succeeding runs were all performed using back-propagation. Also, these and all following runs were executed on IBM RS6000/320s because they are faster than IBM RTs.

Figure 5.7 shows the time trajectory of average generalization performance for 41 independent GAND runs, each started from a different pseudo-random number generator seed. For these runs and all succeeding runs (unless otherwise noted), a population size of 50 was used. Maximum GAND performance averaged across all runs is about 73% or 74%, 12 percentage points higher than randomly generated networks in the population's average generalization performance measure and a 20% improvement. Thus, using back-propagation instead of Quickprop does not provide significant improvement.

Each of the following sections present a series of GAND runs. In each case only a single factor is varied from the baseline back-propagation runs presented here, allowing a direct and meaningful empirical comparison of that factor's effect on GAND performance. Identical scales are used in the following graphs so that the performance plots can be compared easily.

5.5.2 Objective Function Noise

Perhaps the most obvious potential problem is noise in the objective function. Call the initial weight vector used during a back-propagation training run the *weight seed*. Consider the performance landscape over the space of all possible weight configurations, and call it the *weight space*. When a feedforward network is trained using the back-propagation learning algorithm, the network starts at the initial weight seed and follows a generally downhill trajectory in weight space until it reaches a performance minimum. Provided the learning rate and momentum parameters are appropriate to the objective function landscape being traversed (so that 'ridges' are not jumped), the

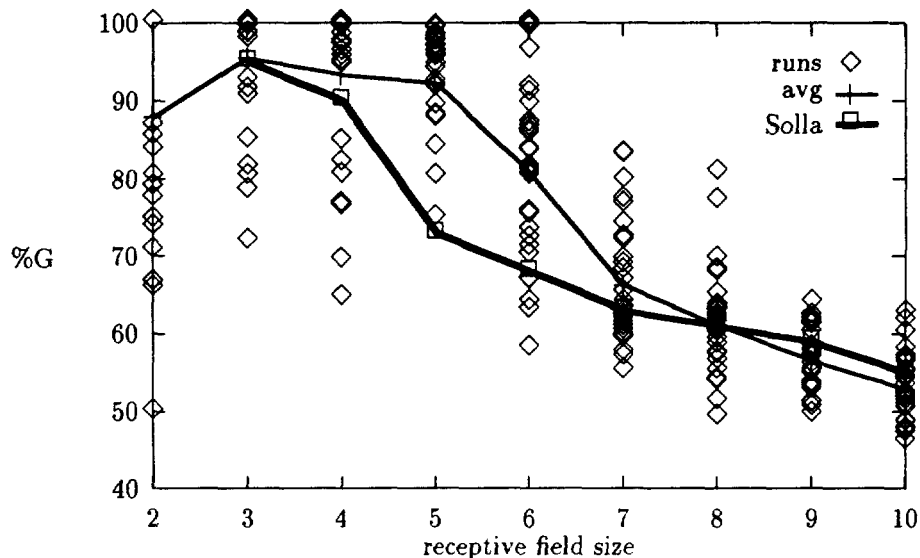


Figure 5.8: Solla's back-propagation versus the back-propagation GAND used: Hidden node receptive field size versus percent generalization performance (%G) for back-propagation learning, 5,000 training epochs. The lighter line shows the average of the individual runs, each shown by a diamond. Also plotted using the heavy line are Solla's results.

network's weights will converge to the minimum within the basin containing the weight seed. When the same network is repeatedly retrained starting from a different randomly selected weight seed, a different basin may be encountered, and a different local minimum may be obtained from each basin.

Thus from GAND's point of view, when the same network is repeatedly retrained starting from different randomly selected weight seeds, a variance in the fitness of a single network results. Figure 5.8 shows hidden layer node receptive field size versus percent generalization performance (%G). Each receptive field size (each column in Figure 5.8) corresponds to one of nine unique network interconnect architectures. Each of these nine networks was run 30 times starting from different random weight seeds. The performance on the test data set for each run is shown by a diamond. Thus, each column shows the performance spread resulting when a single network of specified receptive field size is

repeatedly trained. The vertical spread may be viewed as variance, or noise, in the objective function value of the network architecture. The variance of each receptive field network's fitness was computed, and then the average of these variances was computed, yielding an overall average variance of 95.7, and a corresponding standard deviation of 9.8. Obviously, considerable noise is present.

A number of empirical studies have suggested that GAs are good at optimization in the presence of noise (Jong, 1975; Schaffer, 1984; Fitzpatrick et al., 1984; Grefenstette & Fitzpatrick, 1985; Fitzpatrick & Grefenstette, 1988). However, no comprehensive analytical treatment of how much noise GAs can tolerate has been performed, nor has a rigorous definition of what constitutes noise from the GA's perspective been proposed. The definition of noise developed in Chapter 3 will be applied to GAND population sizing in Section 5.6. An empirical test of how fitness noise affects GAND performance is presented in Figure 5.9. The GAND runs depicted are identical to those in Figure 5.7, except the fitness used to evaluate a genotype was the average performance of three independent back-propagation training sessions on the same network architecture, each started from a different random weight seed. Averaging independently obtained objective function performance figures for a single network architecture reduces the variance of the sampling distribution of means to

$$\sigma_{\bar{X}}^2 = \frac{\sigma_X^2}{n}, \quad (5.3)$$

where $\sigma_{\bar{X}}^2$ is the variance of the sampling distribution of genotype fitness means, \bar{X} ; σ_X^2 is the variance of the genotype fitness population; and n is the number of independent trials averaged, 3 in this case. Thus, the fitness variance seen by the GA should be $\frac{1}{3}$ of the variance of the objective function, or reduced from the 95.7 value obtained from Figure 5.8 to 31.9. The reduction can be seen visually as the narrower vertical cross-section at generation zero in Figure 5.9 as compared to the baseline runs in Figure 5.7. More importantly, although both runs started out with randomly generated genotypes performing at about the 61% fitness level, the reduced variance runs quickly rose to a 78% performance level while the baseline runs reached only 74%, an improvement of

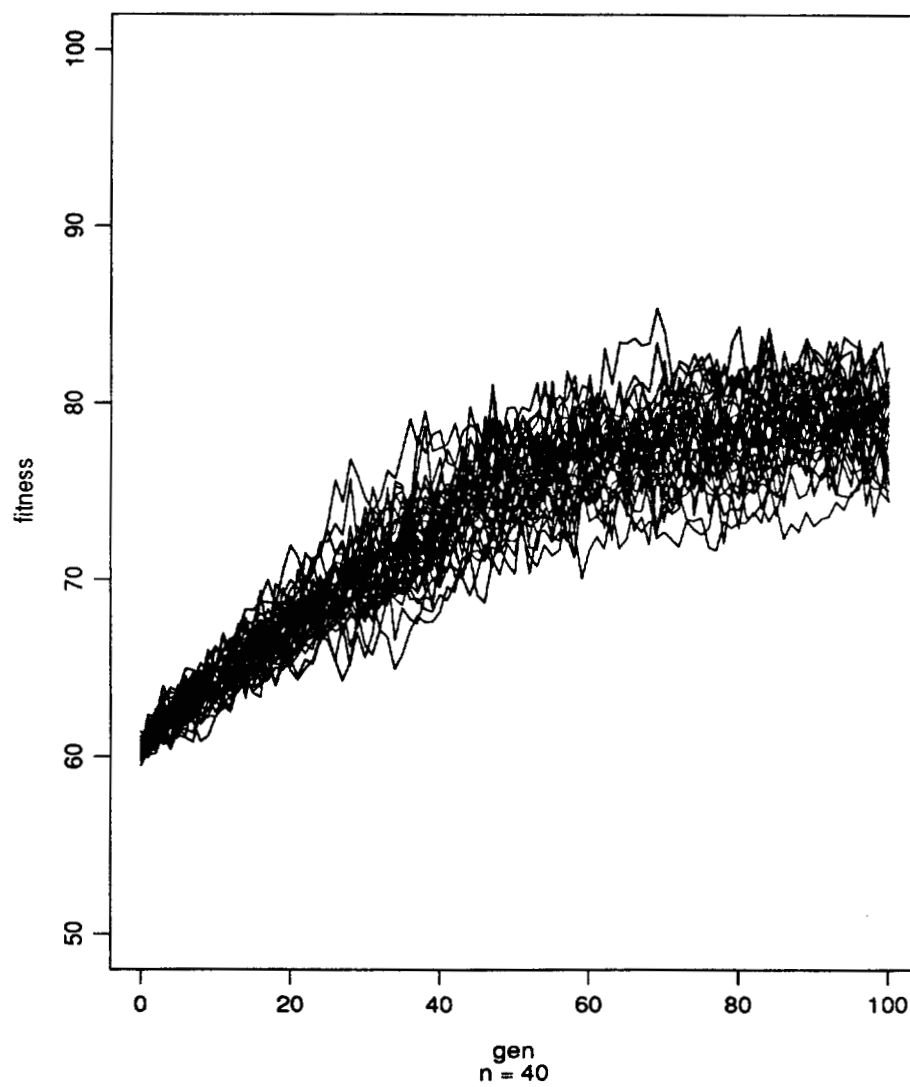


Figure 5.9: GAND performance using average of three independent evaluations of each network architecture.

4 percentage points. Thus, averaging three independent evaluations to reduce fitness function noise resulted in a performance 17 percentage points higher than the randomly generated networks as measured in the population's average generalization performance measure, a 28% improvement over the random nets. Reducing fitness noise improves GAND performance, but it's still not optimal.

5.5.3 Initial Investigation of Population Size

It is important to realize that the reduction in fitness noise from the previous section is obtained at considerable computational cost — three times each genotype is translated into a network, trained, and tested.⁹ Since training the network dominates computational time, this is computationally equivalent to using a GA population three times as large, but training each network only once. Further, increasing population size is one of the factors identified that may improve GAND performance.

Does increasing population size improve GAND performance as much as reducing fitness noise by averaging multiple training sessions? To find out, a series of GAND runs were performed using a population size of 150, three times the population size used in the baseline runs, and resulting in the same number of network evaluations as performed in the noise reduction runs of Figure 5.9. The resulting performance levels are shown in Figure 5.10. As can be seen, the additional computational resources produce GAND performance gains virtually identical to those using noise reduction, and for the same increase in computational cost.

Section 5.6 presents an analysis of GA population sizing when noise is present in the fitness function. A statistical decision theoretic bound on useful population size for GAND on the contiguity problem is derived in Equation 5.17. Given a population size of 150 and working backwards from Equation 5.17 gives a z value of 0.77, corresponding to a 78% confidence level. In similar fashion, the baseline GAND population size of 50

⁹Rebuilding the network from scratch for each independent evaluation is not strictly necessary; it is an idiosyncrasy of the GAND implementation. However, since building the network is computationally trivial compared to training the network, so little harm is done.

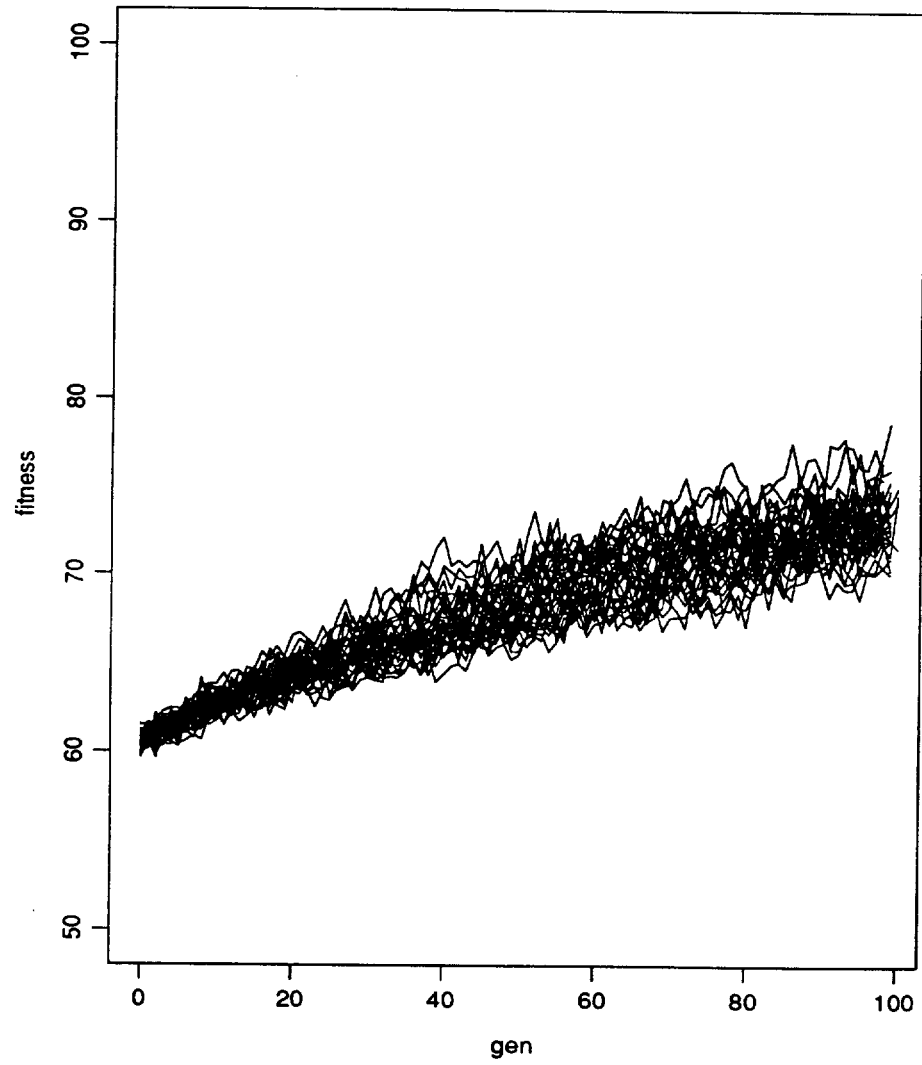


Figure 5.10: GAND performance — population $N = 150$.

corresponds to a z value of 0.44 and a confidence level of 67%. Thus, correct decisions are likely to be made 78% and 67% of the time, respectively, with population sizes of 150 and 50. Because of the various simplifying assumptions made at various stages of the analysis, both confidence level figures are probably higher than they are in reality. Nevertheless, both are considerably better than chance — 50%. As is empirically demonstrated in Section 5.5.5, the baseline population size is adequate, provided the schema theorem is satisfied, for good GAND performance on the contiguity problem.

5.5.4 Link Tax

In artificial neural network models, a price must be paid for each link, either in hardware for a parallel processing implementation, or in computation time for a serial implementation. Each link also has an associated weight, or model parameter. Generally speaking, the fewer model parameters needed to adequately learn the data, the better the generalization performance. For both these reasons, it is natural to consider including a link tax as a component of genotype fitness. For the contiguity problem, however, including a link tax may be cheating, since it is known that a particular kind of sparse interconnect between the input and hidden layers yields superior generalization performance, and a link tax favors sparse interconnect. Despite this, there are compelling reasons for including a link tax independent of any *a priori* knowledge of the superior solutions.

Does including a link tax improve GAND performance? To answer this question a series of GAND runs was performed using the fitness function

$$f_L(\mathbf{x}) = f(\mathbf{x}) \frac{100 - n_L}{100}, \quad (5.4)$$

where \mathbf{x} is the genotype, $f(\mathbf{x})$ is the unmodified fitness function from Section 5.5.1, n_L is the total number of links between the input and the hidden layers, and $f_L(\mathbf{x})$ is the link-adjusted performance. Since the maximum possible number of links is 100, the effect is that fitness is scaled by the proportion of unused links, varying between 0% for full connectivity, to 100% when no connections are present.¹⁰

¹⁰Of course, when no connections are present the input is not connected to the output, and $f(\mathbf{x}) = 0$.

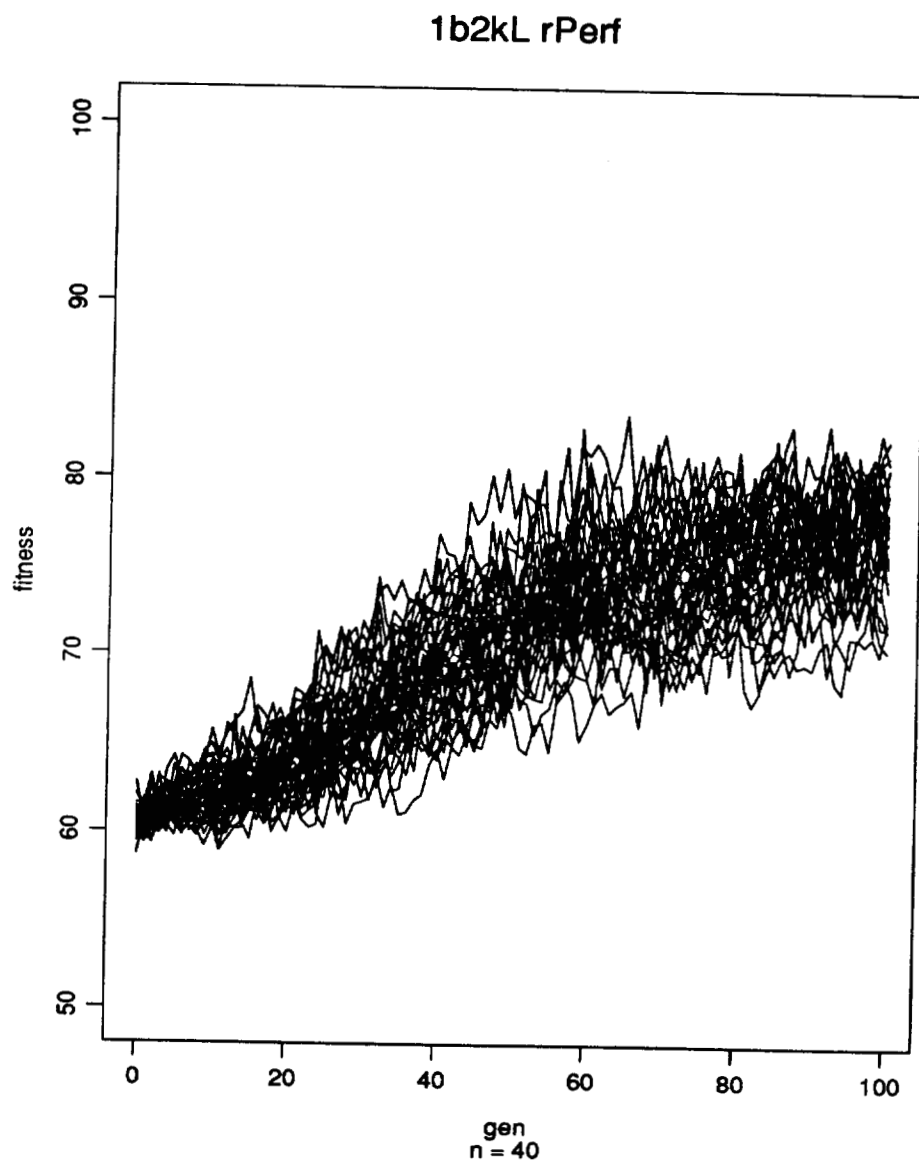


Figure 5.11: GAND performance — link tax.

The resulting performance is shown in Figure 5.11. Note that although link-adjusted fitness, $f_L(\mathbf{x})$, was used in these runs, the ordinate is the usual fitness, $f(\mathbf{x})$, so as to allow direct comparison of these results with other GAND runs. The link-adjusted fitness runs begin with the usual 61% performance level obtained by randomly generated networks at generation zero. However, as compared to the GAND baseline runs of Figure 5.7, there are several differences noticeable as the runs progress. First, the overall shape of the link tax performance curve is that of an 'S', while the baseline runs are an exponential increasing asymptotically to a constant. Second, the spread, or variance, of the link tax curve is noticeably larger than the baseline's. Finally, the maximum link tax population fitness is about 77 or 78%, as compared to only 73 or 74% for the baseline runs, an improvement of 4 percentage points — similar to the improvement obtained by reducing fitness noise. Thus another candidate solution to GAND's mediocre performance, that of employing a link tax, bites the dust.

5.5.5 Elitism

Given that GAND is being asked to not only evolve good solutions to the contiguity problem, but to also settle upon a single, population-wide coding (as discussed in Section 5.4.4), perhaps the solution to GAND's mediocre performance is to use a selection scheme such as elitism (Grefenstette, 1986; Whitley, 1989; Eshelman, 1991; Schaffer et al., 1991). *Elitism* permits outstanding individuals within the GA population to increase their effect upon the gene pool by letting parents, without modification, to compete with children for positions in the succeeding generation. Thus, an exceptional individual may bypass crossover and mutation disruption while still contributing children to successive generations.

Each GAND successor population was produced using Eshelman's (1991) population-elitist selection algorithm, which works as follows:

1. A temporary population is generated by the usual binary tournament method used in the baseline GAND runs.

2. The temporary population and the current population (the parents) are combined into a $2n$ intermediate population.
3. The n genotypes having highest fitness in the intermediate population are included in the successor population.

Thus the parents included in the successor population undergo no further mutation or crossover.

Because the fitness function used by GAND on the contiguity problem is noisy, a 'lucky,' or outlier, fitness evaluation could result in an average, or even below average, parent living indefinitely. To prevent this, each parent to be selected during step three undergoes an additional fitness evaluation which is averaged with its previous evaluations. If the parent, based on this new averaged fitness, is among the n best genotypes in the intermediate population, then it is included in the successor generation. Over a number of generations, this procedure reduces the fitness noise of the parent's evaluations.

Figure 5.12 shows a series of GAND runs using elitist selection. The only thing changed between this series of runs and baseline runs of Figure 5.7 is that elitism has been added. Although the improvement in GAND performance is striking, the results of Figure 5.12 are not quite as good as they seem because elitism and the nondeterministic fitness function interact to produce a "better half" effect. Consider an entire population consisting of a single genotype where the fitness function is noisy and no mutation is used. Under elitism the $2n$ intermediate population of step two will consist of identical genotypes, but since the fitness function is noise they will have a range of fitnesses, as shown in Figure 5.13. Only the "best half" of the $2n$ population, the shaded area of Figure 5.13, will be included in the successor population. Thus, the average fitness of the new population, f_{avg} in the figure, is higher than the average performance of the genotype, f_g in the figure.

Elitism brings GAND performance up to the best levels obtained in any of the explicit human designs shown in Table 5.1. This is particularly good news for the evolutionary approach to ANN design, because elitism is a general purpose GA technique that is in

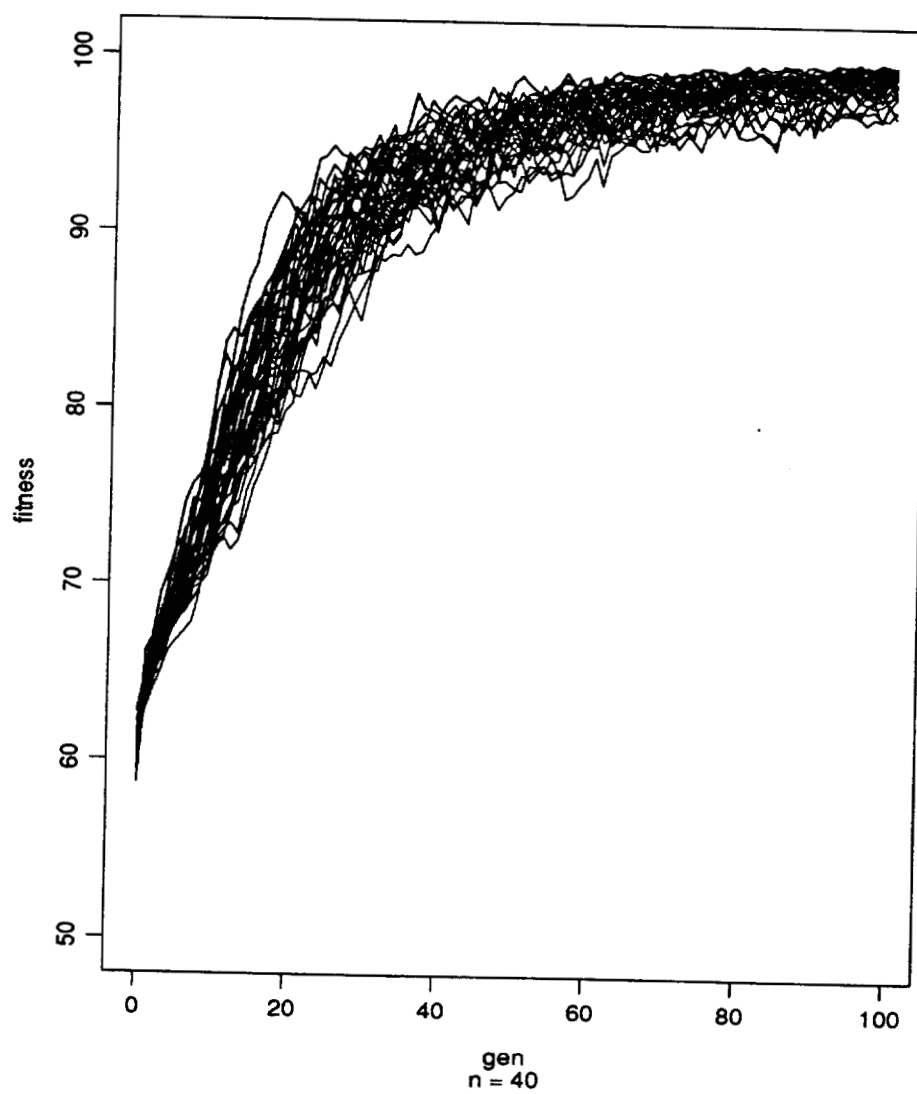


Figure 5.12: GAND performance — elitism.

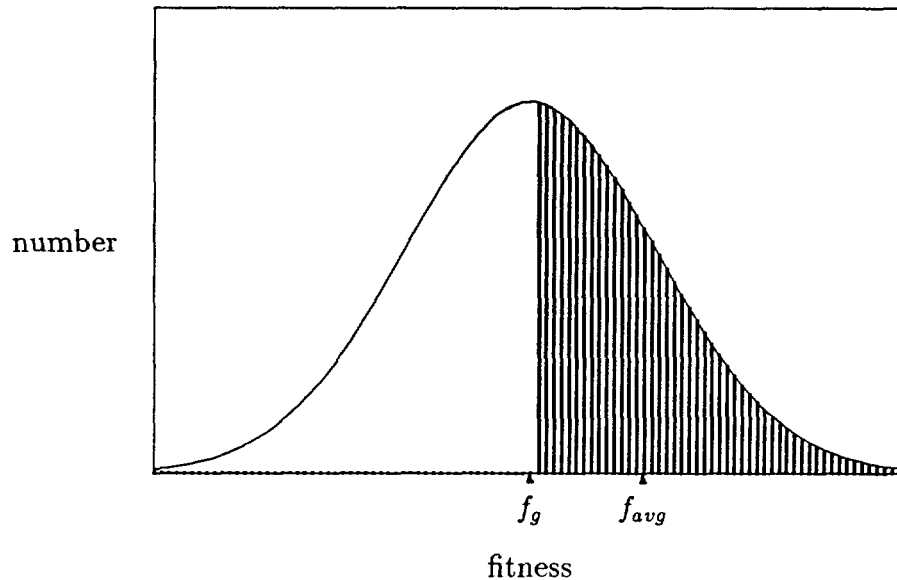


Figure 5.13: Intermediate $2n$ population. Shaded area becomes successor population.

no way specialized for the contiguity problem. Thus, there is reason to hope elitism will be useful in other END tasks.

Figure 5.14 shows two typical networks evolved to solve the contiguity problem from two different GAND elitist runs. Relatively narrow independent hidden node receptive fields (mostly of the near-optimal widths of 3, 4, or 5 as shown in Appendix A) span the input vector, allowing the network to easily learn to count the number of clump edges in the input. The network also has the ‘dirty’ appearance of extraneous connections common to networks evolved using END.

Why is GAND’s elitist selection performance so good? There appear to be two relevant factors, each related to the schema theorem. First, elitist selection has a higher selection pressure than binary tournament selection. Second, parents carried forward to successive generations undergo no disruption from mutation and crossover. An analysis of these two factors is now performed.

Goldberg and Deb (1991) developed a context from which the relative strengths of various selection algorithms can be evaluated. They define the *growth ratio* at generation

Solution Network 1											
Hidden Node	Hidden Node Receptive Field										
	Input Nodes										
	1	2	3	4	5	6	7	8	9	10	
1	.	.	1	1	1
2	1	1	1
3	1	.	1	1	1
4	1	1	.
5	.	1	.	.	.	1	1	1	.	.	.
6	.	1	.	1
7	1	.	.	.	1	1	1	.	.	1	.
8	1	1	1	1	.	.
9	1	1	1	1	.	.
10	1	1	1	.	.

Solution Network 2											
Hidden Node	Hidden Node Receptive Field										
	Input Nodes										
	1	2	3	4	5	6	7	8	9	10	
1	.	.	1	.	.	.	1	1	1	1	.
2	.	1	1	1	.	.	1	.	.	1	.
3	1	1	1	.	.	.
4	1	.	1	1	1	.
5	.	.	1	1	1	.	.	1	.	.	.
6	1	1
7	.	1	1	1	1	.	.
8	.	.	1	.	1	1	1
9	.	.	1	1	1	1
10	.	1	1	1	.	1

Figure 5.14: Typical networks produced by GAND elitist runs. Each solves the contiguity problem.

t , ϕ_t , as

$$\phi_t = \frac{P_{t+1}^*}{P_t^*}, \quad (5.5)$$

where P_t^* is the proportion of the best genotype in the population at time t . The proportion of genotypes with fitness greater than f_i at time t for binary tournament selection is

$$P_{i,t+1} = 2P_{i,t} - P_{i,t}^2. \quad (5.6)$$

Thus, the growth factor for binary tournament selection is $\phi_t = 1 - P_{i,t}$.

For GAND's elitist selection Equation 5.6 becomes

$$P_{i,t+1} = 3P_{i,t} - P_{i,t}^2 \quad (5.7)$$

with the proviso that $P_{i,t+1}$ can grow no larger than 1, resulting in an aggregate growth ratio of

$$\phi_t = 3 - P_{i,t} \quad (5.8)$$

during the early phase of selection when P_i is not yet near 1.

Consider the growth of the best individual \mathbf{x} in the population at time t under GAND elitist selection. When considering disruption of \mathbf{x} due to genetic operators like mutation and crossover in elitism, it is useful to think of the growth ratio ϕ as the sum of two components, a part due to the action of the elitist inclusion of parents in successor populations, ϕ_E , and a part due to the action of binary tournament selection, ϕ_T , or

$$\phi_t = \phi_{E,t} + \phi_{T,t}. \quad (5.9)$$

Since the parents included in successor populations do not undergo mutation or crossover, only the tournament selection growth factor component undergoes disruption. Thus define *net growth factor* at time t as

$$\gamma_t = \phi_{E,t} + \phi_{T,t}(1 - \epsilon), \quad (5.10)$$

where ϵ may be thought of as the disruption probabilities at each generation from the schema theorem. For \mathbf{x} to grow requires $\gamma_t > 1$, or

$$\phi_{E,t} + \phi_{T,t}(1 - \epsilon) > 1. \quad (5.11)$$

Noting $\phi_{E,t} = 1$ whenever $P_{i,t} < \frac{1}{2}$ for the best individual in the population, solving for disruption yields

$$\epsilon < 1. \quad (5.12)$$

Equation 5.12 says that for the best genotype,¹¹ when the probability of disruption is less than total, growth will occur. This makes sense for elitist selection, since at each generation both superior parents and any newly created superior children are included in the successor generation. GAND's elitism is conservative in the sense that whenever superior genotypes are created by crossover and mutation, they don't die until the population average fitness rises, which effectively makes the genotype inferior.

Thus, elitism as implemented in GAND works well both because it has higher selection pressure than binary tournaments and because the best genotypes, once created, are able to avoid the disruptive effects of crossover and mutation.

5.6 Population Sizing with Fitness Noise

The static population sizing equation derived in Section 2.3 assumes the fitness function, $f(\mathbf{x})$, is deterministic and so makes no contribution to schema fitness variance. But for GAND's contiguity problem the fitness function is nondeterministic, or noisy, because of the local minima encountered by back-propagation. In this section, the static population sizing equation, Equation 2.29, is adjusted for fitness function noise.

Fitness function noise may be modeled as ergodic, zero mean, additive, Gaussian noise, denoted by σ_f^2 . All the other assumptions from Section 2.3 are used here. In particular, the fitness function is still assumed to be linear, so it may now be modeled as

$$f'(\mathbf{x}) = \sum_{i=1}^l a_i x_i + b + g(\sigma_f^2), \quad (5.13)$$

where $g(\sigma_f^2)$ is zero mean Gaussian noise having variance σ^2 . Adding the sample variance

¹¹The analysis presented applies to only the best genotype, but may be extended to any genotype of above-average fitness provided the population does not saturate.

of the Gaussian noise, $\frac{\sigma_f^2}{n/2^k}$, to Equation 2.22 yields a noise-augmented equation for the variance of the sample mean fitness of a single, order- k schema,

$$\text{var}(\hat{f}'(\mathbf{h})) = \frac{(l-k)w_1'^2 + \sigma_f^2}{n/2^k}, \quad (5.14)$$

since the variance of a sampling distribution of a sum of random variables is the sum of the variances of the independent random variables. Working through to the population sizing formula of Equation 2.29 as was done in Section 2.3 yields

$$n = z^2 2^{k-1} (l - k + \rho), \quad (5.15)$$

where

$$\rho = \frac{\sigma_f^2}{w_1'^2}. \quad (5.16)$$

Note that the only difference between the function-noise augmented Equation 5.15 and Equation 2.29 is the ρ term in the $l - k + \rho$ factor. Thus, function noise serves to increase, normalized by Walsh coefficient magnitude, the effective code length of the representation used, or $l' = l + \rho$.

For GAND, the code length is $l = 100$, and the schema order is set to $k = 2$, since this is the minimum possible relevant schema size for the contiguity problem. To make further progress the two components of ρ must be estimated. To estimate σ_f^2 , 100 genotypes were randomly generated. Each was then independently evaluated 50 times, with each evaluation starting from a different pseudo-random number generator seed, to generate a reasonably accurate estimate of each network's mean performance and variance. The fitness variance of each of these 100 pseudo-random genotypes was then averaged to give an estimated value of $\sigma_f^2 = 21.8$. Note that this estimate should be reasonably accurate for the random starting population since it is based on randomly generated genotypes, but the value of σ_f^2 may change as the population evolves.

To estimate the value of $w_1'^2$, one bit in each of the 100 pseudo-random genotypes used in the σ_f^2 estimate was flipped so that each genotype had a different bit flipped. The modified genotypes were then each evaluated 50 times, again each starting from

a different pseudo-random number generator seed, to generate an estimate of the average performance of each modified individual. The magnitude of the difference between the average performance of each pair of (modified and unmodified) genotypes was then averaged across all the individuals to obtain an estimated value of $w'_1 = 0.88$.

Using the estimated values of σ_f^2 and $w'_1{}^2$ produced estimated value of $\rho = 28.4$. Substituting for l , k , and ρ in Equation 5.15 yields

$$n = z^2 253. \quad (5.17)$$

Equation 5.17 is a decision-theoretic population size estimate for GAND on the contiguity problem. Choosing a confidence interval of 90% yields a value of $z^2 = 1.64$ from Table 2.4 and a population size of $n = 415$. The comparable population size for a deterministic fitness function is $n = 321$. Thus for the contiguity problem and GAND coding used, fitness function noise should have a noticeable but minor effect, a result consistent with the empirical results of Section 5.5.2.

As with Equation 2.29, Equation 5.17 is based on the assumption that it makes a linear approximation of the fitness function. Likewise, Equation 5.17 applies only to the initial, randomly generated population, both because the population sizing equation of Section 2.3 only applies to the initial population, and because the estimates of σ_f^2 and $w'_1{}^2$ were based on randomly generated genotypes. Thus as mentioned in Section 2.3, the large population sizes derived for GAND on the contiguity problem should be viewed as an approximation. Further, as the GA runs and the number of genotypes in the superior schemata increases, their associated fitness variance will decrease and a smaller population size may be sufficient to choose among competing schemata, as was empirically observed in Section 5.5.2.

5.7 END Discussion

The GAND work presented here tackled an architectural space containing systematic, non-trivial, connectivity features having a significant impact on network generalization performance. Of the variety of GAND improvements examined, elitist selection worked

best, producing near-optimal architectures having performance similar to the best hand-designed architectures.

But as pointed out in Section 5.4.4, the inter-layer connection matrix representation used by GAND on the contiguity problem suffers on two counts. First, it is not a compact representation. Second, it is not a normal form with respect to back-propagation — a single network may have multiple codings. Both of these reasons contribute to the possibility that GAND using a connection matrix representation may scale up poorly. In Section 5.7.1, possible normal form representations are considered. In Section 5.7.2, other GAND future research directions are discussed. As concluding remarks, Section 5.8 presents an overview of the relevance of END in the endeavor to use massively parallel computational resources to solve difficult problems.

5.7.1 Normal Forms

When using a connection matrix style representation, any ANN model, such as back-propagation, which does not respect node order within a layer will have multiple genotype codes for the same network. This can be a problem for GAND both because it increases the size of the search space without increasing the expressiveness of the representation, and because two identical networks having different representations can produce non-viable offspring. In a normal form representation each network architecture is coded by only a single genotype, eliminating both difficulties.

For a specific example of how multiple codes can pose difficulty for GAND, consider genotypes A and B in Figure 5.15, two network specifications coded using the inter-layer connection matrix representation. These two genotypes represent one and the same network from back-propagation's perspective, a network which happens to perform well on the contiguity problem. But from the GA's perspective they are entirely different networks. In fact when they are bred by the GA, their offspring are likely to perform very poorly. For example, if a single crossover point¹² fell exactly in the middle of the

¹²Note that the same kind of problem also arises with multi-point crossover operators.

[illegible]

Figure 5.15: How two identical networks can produce non-viable offspring. Four genotype network specifications are shown. Each is a connection matrix showing the input-to-hidden-layer connections as in Figures 5.5 and 5.14. Genotypes A and B are identical networks from back-propagation's perspective, but have very different codings. Child 1 and 2 result from a midpoint crossover of genotypes A and B. Note that child 1 has no connections from the right side of the input field and child 2 has none from the left; thus, each child has poor performance on the contiguity problem.

genotype (between the fifth and sixth rows), child 1 and 2 in Figure 5.15 would result. Each of these networks will perform poorly, since each has no connection to nearly half the input layer, while having duplicate connections to the other half of the input layer. Thus, GAND is required to not only optimize nondeterministic network performance, but must also settle upon a single genotypic sub-code across the GA's population from among the many possible, semantically equivalent representations allowed by the connection matrix form of representation.

For the contiguity problem a 100-bit genotype was used, so the space being searched is of size 2^{100} . However, this space is larger than required because of search space symmetries; the space of unique network architectures for the contiguity problem is considerably smaller. The situation is much as though the GA were required to search the space formed by the Cartesian product of the network space intrinsic to the contiguity problem crossed against a large number of different representations. This will be called the *multiple codes problem*.

Thus a normal form representation is desirable, provided it does not introduce a nonlinearity worse than the multiple codes problem. One possible normal form representation is to order each hidden node's position in the 100-bit, 10-hidden-node genomic code by first left-most connection, with ties being recursively broken by next left-most connection. However, for the contiguity problem, a first left-most normal form representation doesn't seem to fit the semantics of what is important in the network architecture to solving the problem, namely, the location of size two (or larger) receptive fields. To see this clearly, consider genotypes C and D shown in Figure 5.16; each is in left-most normal form. When these two individuals are bred with a crossover occurring in the middle of the genotype, the same problem occurs as was shown in Figure 5.15 — child 3 has no functional receptive fields (having size two or greater) on the right side of the input field, while child 4 has no functional receptive fields on the left side of the input. So it is not simply that a normal form is needed, but rather, a normal form is needed which respects relevant network structure.

Since what is relevant in the network structure is likely to be problem dependent,

Genotype C'										Genotype D'									
1	1	1	1
.	1	1	1	.	.	1	1
.	.	1	1	1	1
.	.	.	1	1	1	1
.	.	.	.	1	1	1	1
.	.	.	.	1	1	1	1	.	.	.
1	1	1	1	1	.	.
1	.	.	.	1	.	1	1	1	1	.
1	.	.	1	.	.	.	1	1	1	1
1	.	1	1	1	1	1

Child 3'										Child 4'									
1	1	1	1
.	1	1	1	.	.	1	1
.	.	1	1	1	1
.	.	.	1	1	1	1
.	.	.	.	1	1	1	1
.	1	1	.	.	.	1	.	.	.	1	1
.	1	1	.	.	1	.	.	1	.	1	1	.	.	.
.	1	1	.	1	.	.	1	.	.	1	1	.	.
.	1	1	1	.	1	1	1	.

Figure 5.17: Left-most largest normal form representation. Genotypes C' and D' are the same two networks as genotypes C and D shown in Figure 5.16, but are here shown in left-most largest receptive field normal form. Now, when they undergo a midpoint crossover, the children are each viable.

Genotype A'		Genotype B'	
receptive field		receptive field	
start	size	start	size
0	3	0	3
0	3	0	3
1	3	1	3
2	3	2	3
3	3	3	3
4	3	4	3
5	3	5	3
6	3	6	3
7	3	7	3
7	3	7	3

Figure 5.18: Left-most single receptive field normal form. Network genotypes A and B, from Figure 5.15, are shown here in left-most single receptive field normal form. Note that the two identical networks now have identical genotypes, as there is only one way to represent a network.

what constitutes a good normal form is also likely to be problem dependent. For the contiguity problem, a left-most largest receptive field normal form seems sensible. Figure 5.17 shows genotypes C and D from Figure 5.16, but converted to left-most largest receptive field normal form. Note that when C' and D' are bred, and no matter where the crossover point occurs, relatively viable offspring will be produced. This is a big improvement over the kind of result shown in Figure 5.16. Now in some sense, use of a normal form tailored to respect receptive field position on the contiguity problem is cheating, since it is known beforehand that narrow receptive fields are advantageous. However, to the extent that receptive fields prove to be generally useful on a variety of problems, using left-most largest receptive field as a generic normal form is a sensible thing to try. But, of course, when multiple layers are coded, problems arise.

A yet more restrictive normal form may be obtained by using the first left-most normal form, but allowing only a single receptive field per hidden node instead of arbitrary inter-layer interconnect — first left-most single receptive field normal form. Each hidden node's interconnect could then be coded using only two parameters, starting position and

width of the receptive field. Figure 5.18 shows the networks A and B from Figure 5.15, but now in first left-most single receptive field normal form, in which identical networks have identical genotypes.

5.7.2 Future GAND Research

END is still very much an infant discipline. The GAND work presented here demonstrates END can find superior architectures for a challenging problem. However, much work remains and many areas still need to be addressed. In this section, some of the options for further research are outlined.

Elitist selection performance was far superior to the other GAND runs. The analysis of Section 5.5.5 showed two possible reasons for the improved performance — higher selection pressure and protection of genotypes from disruption (the elitist part of the selection scheme). But exactly how much of the improvement is due to each factor? As a first step towards an answer, a non-elitist, ternary tournament selection GAND run should be performed since it has a selection pressure similar to the elitist selection used, especially in the early stages of the run. The difference in performance between the ternary tournament selection runs and the elitist runs will then be partly due to the elitist protection of existing genotypes from disruption.

How much elitism is helpful? For example, the 100% elitism used by the GAND elitist runs (all genotypes compete for positions in the successor population) may be counter-productive by excessively reducing exploration when a majority of the new population are elitist parents. For example, some elitist selection schemes include only the single best genotype in the successor generation. To answer this question, a series of GAND runs can be performed varying the proportion of genotypes competing for positions in the successor generation.

GAND has used back-propagation training algorithms in which weights were updated after each epoch (complete presentation of the training data set). The number of training exemplars needed to adequately characterize the gradient should be explored. An easy, adaptive way to do this is to include number of training exemplars per weight update

in the genotype as a parameter to be optimized by the GA. At the same time GAND's stopping criteria must be changed from a fixed number of epochs to a fixed number of training exemplar presentations.

An empirical study would be useful which contrasts the performance of the three normal forms of Section 5.7.1 — left-most first connection receptive field, left-most largest receptive field, and left-most single receptive field — with the connection matrix representation used in the GAND results presented in Section 5.5.

The use of a multidimensional genotype and crossover operator (McMahon & Fox, 1991) should be tested in place of the customary one-dimensional genotype and crossover. Multidimensionality allows the receptive field work being done by the GA on one hidden node to be local, or adjacent, to multiple other hidden units with respect to crossover. It may well be the non-locality (with respect to crossover) between hidden node representations that makes the contiguity problem particularly difficult for the non-elitist, simple GA. Note that the mutation operator need not change when using a multidimensional genotype.

As discussed in Appendix B, Quickprop provides a reduced discrimination signal for narrow receptive fields on the contiguity problem as compared to back-propagation. Because of this, all but the first exploratory GAND runs were performed using back-propagation. Now that elitism has been used to minimize disruption (essentially, getting the schema theorem right), an additional set of runs using Quickprop should be used to explore the performance of GAND on the contiguity problem using Quickprop's reduced narrow-receptive-field discrimination signal.

GAND was only run on the contiguity problem. Although the contiguity problem is a particularly good END test problem, other problems should also be tried. For example, the X-OR problem could be run on GAND to verify that the results are comparable to those obtained by others. It is especially important for END to tackle more difficult problems and ANN models, especially where good architectures are currently not known. Hebbian network architectures would seem particularly promising in this respect because of their biological plausibility.

Related to the symmetry and normal form issues is developmental specification — how much and what kind of developmental translation should be performed in converting a genotype into a phenotype. Motivation for developmental specification is twofold. First, a more compact network specification usually results, reducing the size of the search space. But of course, at least a portion of the superior network architectures must be able to be expressed. Second, biology does this. For example, there is considerable evidence suggesting the number of genes coding for the human brain is substantially smaller than the number of neuron connections in the mature brain (Gierer, 1988). The present GAND work used a connection matrix representation with no developmental mechanism. A developmental mechanism should be incorporated into GAND, perhaps a graph L-system grammar as was done by Kitano (see Section 5.3.6).

A related issue is the interplay occurring in natural nervous systems between development and learning. For example, in humans at around eight years of age, a massive die-off of synapses occurs, effectively pruning neural connectivity. Although various architecture modifying mechanisms have been developed for ANNs (as mentioned in Section 5.3), they have yet to be combined with END. Such work should be undertaken.

Finally, the various parameters common to ANN models are also candidates for optimization by inclusion in the genotype, including the connection weights themselves — for example, see Whitley et al. (1989). Belew, McInerney, and Schraudolph (1990) have empirically shown that solving the symmetry problem by optimizing learning rate, momentum, and a coarsely coded initial weight vector using a GA is similar in total computational time to running a single back-propagation network on the same problem using conventional settings for these parameters. This counter-intuitive result is apparently attainable because for certain initial weight vector regions, very high learning rate and momentum values become feasible.

5.8 GAND Conclusion

Because of fundamental physical limitations of computational implementation technologies (especially the speed of light) the future speedup of serial, uniprocessor computer architectures is strictly limited. Thus, massively parallel computer architectures are being actively investigated as an alternative means of bringing additional computational resources to bear upon difficult problems. However, writing large, complex programs on uniprocessors taxes programming technology. Explicitly dealing with parallelism is known to dramatically increase programming difficulty. A number of approaches to programming parallel processors are actively under investigation and may eventually become economically attractive.

Of course some algorithms, such as artificial neural networks,¹³ are intrinsically parallel, and thus may be more easily implemented on parallel hardware. They effectively up-level the difficult parallel programming problem to one of choosing an ANN model, architecture, and training regime. As relatively cheap, massively parallel neurocomputers become available, the issues surrounding network design will continue to gain in economic and intellectual importance.

Given a specific problem to be solved, the task of choosing superior ANN architectures has been characterized here as the network design problem. Evolutionary network design has been demonstrated as a candidate solution for the network design problem. It can provide a systematic way to handle all aspects of the network design problem. It can also make good use of massively parallel computational hardware, both at the ANN architecture and in the fitness evaluation portion of END. Thus, evolutionary network design has a potentially important role to play in bringing massively parallel computational hardware to bear on difficult problems susceptible to solution by artificial neural networks.

¹³For that matter, genetic algorithms are also highly parallel in the fitness evaluation of each genotype, the time-consuming part of END.

Chapter 6

Discussion

In this chapter the present work is discussed and opportunities for further research are assessed. The present work is related to the schema theorem in Section 6.1. The relevance of the signal-to-noise ratio (SNR) to GA decision-making is discussed in Section 6.2. Opportunities for extending the current work are outlined in Section 6.3. Finally, the conclusions of this work are summarized in Section 6.4.

6.1 SNR, Schema Theorem, and GA Convergence

Despite its simplicity, the genetic algorithm is a highly complex dynamical system capable of solving difficult problems through use of a scalar reinforcement signal. Although considerable theoretical progress has been made in understanding GA function, and the fact that GAs do converge has been well established empirically, exactly how they converge continues to resist a complete analytic treatment. An analytic treatment of GA convergence is important. A GA convergence proof would provide a more detailed understanding of how GAs work, which should, in turn, aid the application of GAs to difficult problems. It may also provide additional insight into when the GA will and will not converge to a globally optimal solution.

A key early insight was the notion that schemata are central to the function of GAs. Based on that insight, the schema theorem provides an analytical statement about how schema average fitness relates to expected GA convergence. Two convergence regimes

for schemata result — an early, exponential growth regime (or decay regime, depending upon whether schema fitness is above or below population average fitness), and a later, asymptotic regime in which schema population proportions approach saturation (or extinction).

Another key insight is the notion that competition partitions are central to GA function. The competition partition is the locus of GA convergence — it's where GA convergence occurs.

The present work explores the role of fitness variance in GA function. Fitness variance relates to GA convergence through GA decision-making. As will be discussed in the next section, schema fitness variance relates to the quality of the GA decision-making taking place between competing schemata in each partition. More specifically, the SNR provides a measure of the quality, or correctness, of the decisions the GA makes in each partition. To reach this conclusion, a number of steps were taken.

First, a Walsh basis expression for schema fitness variance is derived. It is useful because the Walsh basis respects the way competition partitions structure the GA's search space. Signal, noise, and the SNR are defined, providing a relative measure of the quality of GA decision-making in each partition. Walsh basis expressions are derived for each, yielding a particularly lucid, fitness-function-independent expression (the effects of the fitness function are captured by the Walsh coefficients) for the SNR in the flat population.

A simple GA demonstration problem is empirically examined and then analyzed, showing how the SNR relates to domino convergence and providing new insights into GA convergence and convergence stall. The SNR provides a rank ordering, or queue, of competition partitions with respect to each partition's ability to control selection events. In effect, the competition partition SNR is a step towards generalizing the schema fitness ratio used in the schema theorem from schemata to competition partitions. Just as the schema fitness ratio provides a measure of how selection drives individual schema growth and decay, so too the SNR provides a measure of how the GA allocates its limited selection events among competition partitions, which, as discussed in the next section,

directly relates to partition convergence. Thus, a new analytical tool is available to explore and analyze GA convergence.

A clear graphical demonstration of domino convergence and convergence stall is given by the simulation runs of Chapter 4. However, domino convergence is a general phenomenon not limited to monotone fitness functions or binary coded parameters. In fact, it may well be the rare fitness function, such as functions of unitation¹ (Goldberg, 1990a), which do not undergo domino convergence. But even that case can be viewed as a degenerate case of domino convergence, where all order-one partitions belong to the same equivalence class in the total order induced by the SNR.

6.2 GA Decision-Making

The signal versus noise perspective provides new insight, based on a statistical-decision-theory-motivated analysis, to the quality of decisions² the GA makes in each competition partition. The simple version is obtained by assuming, as was done by Goldberg and Rudnick (1990), a bitwise linear problem (or approximation to the problem) and considering only order-one partitions. It results in a population sizing Equation, 2.29, providing a probabilistic bound on the GA choosing the second-best schemata over the best in the first generation due to stochastic sampling error in the population — a worst-case estimate in the sense that choosing any other schema (for example, the third best) over the best will be no more likely. Although this is an explicit analysis, both the restriction to order-one partitions and consideration of only the best two schemata in a partition are limitations.

The SNR of Chapter 3 generalizes the population sizing equation to arbitrary partitions and times. It induces a total order over partitions with respect to the correctness,

¹The unitation function is the number of 1s in a binary string, and functions of unitation are functions of the number of 1s in the string. For example, the number of 0s in a fixed-length binary string is a function of unitation.

²Quality of decisions, here, refers only to the aspect of decision-making concerning the adequacy of the GA population's sampling of the search space. The related but separate issue of deception leading the GA astray is not addressed.

or quality, of the decisions the GA makes in arbitrary partitions. Each partition can be thought of as a hog trying to feed at the selection trough. A partition feeding corresponds to controlling selection events. The SNR ranks each partition as to its ability to get to the trough — the higher a partition's SNR, the more control it has over the GA's selection events and the better are the resulting GA decisions, or choices, between that partition's competing schemata. Partitions that have a relatively low SNR don't get to feed at the trough — they don't control selection events. How much a partition feeds once it gets to the selection trough depends on how hungry it is, which isn't directly measured by the SNR; the SNR only measures a partition's ability to get to the selection trough. Thus, the SNR is directly related to how much additional convergence occurs in the partition, but is not a measure of the partition's expected convergence. Partitions with high SNRs but little diversity aren't very hungry — they're already mostly converged; they can get to the trough easily, but don't feed much once they are there.

A good overall picture of how this works is provided by Chapter 4's domino convergence simulations of the $f(\mathbf{x}) = x$ problem, Figure 4.1. Initially, only the high-significance bit positions get to feed at the selection trough (control selection events), receive good GA decision-making, and converge. The low-significance bit positions can't get to the trough, resulting in essentially random GA decision-making in those partitions. The poor decision-making, combined with crossover to mix the linkage between high-significance and low-significance positions, results in essentially³ no convergence at the low-significance positions. However, as convergence proceeds, the high-significance bit positions converge, eventually resulting in a SNR of 0. This results, for the unconverged positions of lesser significance, in a lowering of external noise and an increase in their SNR, enabling them to feed at the selection trough, improve their GA decision-making, and move towards correct convergence.

When the more-significant positions are mostly but not fully converged, they will

³Of course, this depends on population size. For population sizes of the order of the size of the search space and larger, some convergence would occur at all positions. However, for population sizes that are small relative to the size of the search space, essentially no convergence occurs at the low-significance positions.

have high SNRs; they get first crack at the selection trough. But since they are already nearly fully converged, they aren't very hungry — they seldom differ at the partition's fixed positions, and thus seldom compete to control selection events. Because they aren't hungry, they feed little at the trough and seldom control selection events. This allows partitions with lower SNRs to feed.

When the mutation rate is large with respect to genotype length, convergence stall occurs. In effect, the higher mutation rate makes each higher-significance position hungrier, with the result that each feeds more at the selection trough. By the time the low-significance positions get to the trough no food is left — they are starved out, control few selection events, do not get good decision-making, and thus fail to converge.

We argue that for any fitness function the SNR induces a total order of partitions with respect to the quality of the GA decision-making for the partition. For example, if the bit positions in the $f(\mathbf{x}) = x$ problem are randomly permuted, but each bit's contribution to fitness remains unchanged, the GA's function is essentially unchanged.⁴ Although the lucid plots of Figure 4.1 become scrambled, the SNR still orders partitions by quality of GA decision-making, and convergence order by pre-permutation bit position remains unchanged.

6.3 Future Research

This section explores possible extensions and additional research based on the current work. In Section 6.3.1 the relevance of schema fitness variance and the signal-to-noise ratio (SNR) to GA convergence is discussed. Suggestions are made to extend the present work toward a general analytic model of GA convergence. Section 6.3.2 discusses other extensions of the SNR work. Section 6.3.3 suggests an extension to the fitness variance based population sizing work. Section 6.3.4 discusses possible extensions and enhancements to the domino convergence work of Chapter 4. Finally, note that the GAND

⁴Of course, the linkage association, crossover mixing time, or how much hitchhiking occurs changes, but this can be eliminated by using uniform crossover.

evolutionary network design work was discussed in Section 5.7, including areas for further work, and especially focusing on representational issues.

6.3.1 Fitness Variance and GA Convergence

Much additional work is needed to make the connection between the SNR and GA convergence analytically explicit. First, a proper analytic definition of GA convergence is essential. Second, a proper analytic definition of selection pressure is also needed, since selection is the engine driving GA convergence. Finally, the convergence definition should be capable of being analytically related to both the SNR and selection pressure.

A GA convergence measure should meet five criteria.

1. It should be analytic.
2. There are two different meanings of convergence. One is static, measuring how much convergence has occurred; it might be termed *convergence level*, C_l . The other is dynamic, measuring how much convergence is currently taking place; it might be termed *instantaneous convergence rate*, C_i . Both are functions of time. Convergence level should simply be cumulative instantaneous convergence since generation zero, or

$$C_l(t) = \int_0^T C_i(t)dt, \quad (6.1)$$

where T is the time of interest.

3. In order to allow a structural aspect to the definition, convergence should include some notion of distance in the search space, or solution similarity.
4. Since both domino convergence and the SNR work indicate the structural unit in which convergence occurs is the competition partition, a definition of convergence should apply equally to populations and competition partitions.
5. A convergence measure should allow the fitness function's effect on convergence to be ascertained.

An information-theoretic definition of GA convergence (Wilson, 1987) seems promising based on exploratory empirical GA simulations. It would be based on entropy, with the usual probabilities replaced by either genotype or partition schema population proportions, depending upon whether population or partition convergence is being measured. Information redundancy would then serve as the measure of convergence. First, it is analytic. Second, it should be usable for both convergence level and instantaneous convergence. Third, it can be applied to the similarity subsets of partitions, and thus incorporates a notion of distance, or similarity, in the solution space. Fourth, it can also be applied to populations. And fifth, since the SNR can be stated in terms of the structure of the fitness function by using the Walsh basis, an information theoretic definition of convergence has the potential to relate the structure of the fitness function to the GA's convergence behavior through the SNR.

Goldberg and Deb's (1991) instantaneous growth ratio⁵, ϕ , might be used as a rigorous, analytic definition of selection pressure. Whether that definition is appropriate for use in the SNR and convergence context must be determined.

Given rigorous analytic definitions for selection pressure, convergence, and the SNR, the goal is to combine them to derive an analytic expression for instantaneous convergence as a function of selection pressure and competition partition. Doing this will bring a rigorous GA convergence proof a step closer.

One specific issue in need of clarification is the role played by partition order. For example, preliminary GA simulations suggest the SNR is monotone in partition order. This makes sense, since the higher the partition's order the more squared Walsh coefficients are in the numerator of $R(J)$ and the fewer are in the denominator. This might be interpreted as suggesting convergence rate increases with partition order, and thus the higher-order partitions should converge first, a patently counter-intuitive result. But entropy also appears to be monotone in partition order for both randomly generated populations and partially converged populations, and thus may balance out the SNR

⁵See Section 5.5.5

increase.

Finally, the addition of noise to the fitness function should be explored, both as a means of controlling premature convergence and as a possible means of improving the quality of GA solutions.

6.3.2 Other Signal-to-Noise Ratio Extensions

In addition to the GA convergence work outlined in the previous section, several other extensions to the signal versus noise work of Chapter 3 are possible. First, a Walsh basis schema fitness variance expression for non-uniform populations can be derived along the lines Bridges and Goldberg (1991) have taken with the Walsh schema transform and as suggested by Goldberg and Rudnick (1990). Using it, a non-uniform version of the Walsh basis SNR expressions can be derived. Second, an order-approximation for $R(\mathbf{J})$ can be derived similar to that done by Goldberg (1989a) with $\hat{f}^{(i)}$. And finally, an operator-adjusted version of the SNR might be derived along the lines taken by Goldberg with the operator-adjusted version of the schema theorem (Goldberg, 1989b).

6.3.3 Population Sizing

The static population sizing analysis presented in Section 2.3 might be improved by eliminating the assumptions about fitness function linearity and Walsh coefficient equality. The full Walsh fitness variance formula, Equation 2.18, can be used, resulting in a more accurate specification of the population size needed to achieve a specified level of confidence on the initial decisions the GA makes between two competing schemata.

Once a non-uniform version of the Walsh basis schema fitness variance expression is derived (see 6.3.2), it should be used to derive a dynamic version of the population sizing formula. Such a formula could be used to place rigorous, statistical-decision-theory-based bounds on the probability of GA decision errors between two competing schemata, given a specific population.

6.3.4 Domino Convergence

There are several ways in which the various models of domino convergence for the $f(\mathbf{x}) = x$ problem may be enhanced or extended. First, the initial window width model of Section 4.2 may be generalized to times beyond the initial generation by replacing the random population assumption with the expressions, derived in Section 4.3, for the proportions of each allele in the fully and partially converged regions.

Second, the refined model for the expected GA trajectory of Equation 4.16 (proportion of 1s in succeeding generations, $P_{i,t+1}$) can be coded and run on a computer to compare its predictions with actual simulation results. This would produce 2-d perspective plots like those in Figure 4.1. It would also yield predictions of the steady-state behavior and stall points of the GA for various levels of mutation.

Finally, attempts should be made to adjust the streamlined model of Equation 4.17 for the accumulation of mutations across generations, allowing it to more accurately predict the stall point behavior observed in the empirical runs.

6.4 Conclusion

Domino convergence and convergence stall are fundamental to GA function. Although they are GA convergence phenomena, they result from the ability, or inability, of the GA to make correct decisions within competition partitions. Partition's whose schemata are adequately sampled converge; those not adequately sampled stall.

Schema fitness variance relates to GA decision-making through statistical decision theory, leading to a signal versus noise perspective relating schema fitness variance to GA decision-making. The resulting signal-to-noise ratio is a measure ordering competition partitions with respect to the quality of GA decisions they may be expected to experience.

Appendix A

Characterizing Contiguity Problem

As a preliminary step toward using the contiguity problem as a test problem for GAND, Solla's reported empirical results were duplicated. In the process, a number of parameter values were set. Unless otherwise noted, all descriptions and specifications apply both to the duplication of Solla's work and to the various GAND runs reported in Chapter 5.

Solla used a data set containing all 792 possible input vectors of size 10, containing either two or three clumps of 1s. This data set was partitioned into a training data set of 100 exemplars and a testing data set consisting of the remaining 692 exemplars. In the present work for each independent network to be trained, the 792 input exemplars were randomly partitioned into training and test data sets so as to avoid the possibility of a systematic bias being introduced due to a single fortuitous or adverse partitioning. Target vector values of 0.1 for negative output (occurring when the number of clumps of 1s in the input field where less than or equal to 2) and 0.9 for positive output were used.

The receptive fields of size s at the beginning and end of the input vector are allowed to bump up against the edge of the input field, remaining width s rather than being truncated. This way the $p = 10$ width receptive fields correspond to fully connected layers.

A uniform distribution in the interval $(-0.5, 0.5)$ was used for setting initial random weights. This may be of relevance, since several people have shown that back-propagation can be particularly sensitive to initial weight vectors (Belew et al., 1990).

The values of $\lambda = 1$ (the exponential scaling factor in the logistic function), a momentum of 0.9, and a learning rate of 0.05 were used. A stopping epoch number for

GAND runs of 2000 was used except as otherwise noted. Per-epoch weight updates were used throughout the present study.

In order to verify whether the GAND results are comparable to those of Solla reproduced in Table 5.1, a set of runs intended to duplicate Solla's results were performed. The results are shown in Figure 5.8, which contrasts hidden node receptive field size versus percent generalization performance, showing both the comparison runs and Solla's reported results. Each of 30 randomly initialized runs per receptive field size is shown by a diamond, the average for each receptive field size is shown by the lighter line, and the corresponding averages obtained by Solla (listed in Table 5.1) are shown by the heavy line.

Several things can be seen here. First, there is considerable spread in performance (variance) for each network. Second, the comparison run's results and Solla's results are quite similar except for the mid-range receptive field sizes of $p = 5$ and 6, where the difference looks significant based upon the spread of the comparison runs (the diamonds). Solla reported no variance data to go with the reported averages; thus, level of confidence comparisons are not directly possible. However, these results strongly suggest there is some difference between the network model used here and Solla's. Third, Solla reported all $p = 3$ simulations achieved 100 percent performance on the training set, while several of the 30 comparison back-propagation runs did not. All in all, however, these differences are rather minor since both curves representing average performance monotonically increase as receptive field size decreases from 10 to 3.

Appendix B

Quickprop versus Back-propagation

Fahlman's Quickprop algorithm was implemented in GAND because it is much faster than back-propagation (nearly an order of magnitude on the contiguity problem). Such a speed improvement is fairly common (Regier, 1990; Fahlman, 1990; Fahlman, 1989b).

The generalization performance of Quickprop is compared to that obtained from GAND's back-propagation in Figure B.1. Each of 30 randomly initialized, 1000 epoch Quickprop runs done for each receptive field size is shown by a diamond, the average performance is shown by the lighter line, and the corresponding averages obtained from back-propagation are shown by the heavier line.

A considerable difference can be seen between Quickprop and back-propagation. The Quickprop solutions show much less difference in generalization performance as hidden node receptive field size varies: Back-propagation shows a 43% maximum difference in average generalization performance, while Quickprop shows only 21% — less than half the difference. Because of the relatively large variance in the performance of each network and because the slope of the Quickprop line is about half the slope of the back-propagation line, Quickprop provides GAND a much reduced discrimination signal relative to receptive field size. Thus, the likelihood of selecting the wider receptive field genotype (making the wrong choice) in a tournament between, say, a $p = 5$ and a $p = 6$ genotype is much larger for Quickprop than for back-propagation. In effect, the relatively small signal provided by Quickprop gets lost more easily in the noise of the fitness function. Thus, Quickprop may be a poor choice for guiding GAND to the

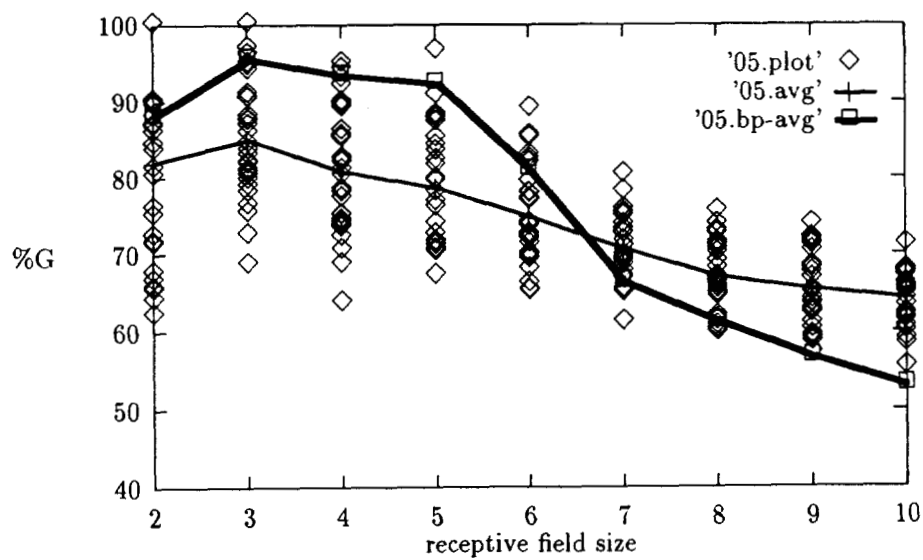


Figure B.1: Quickprop versus back-propagation: Receptive field size versus percent generalization performance (%G) for each of 30 Quickprop, 1000 training epoch runs (diamonds). The Quickprop average is shown by the lighter line. The back-propagation average is shown by the heavy line for comparison.

better regions of the search space, despite its order of magnitude speed gain over back-propagation. As a result, back-propagation was used for most of the GAND runs rather than Quickprop.

Bibliography

- Ackley, D. (1987). *A connectionist machine for genetic hillclimbing*. Kluwer Academic Publishers.
- Ash, T. (1989, February). *Dynamic node creation in backpropagation networks* (Tech. Rep. Institute for Cognitive Science #8901). UC San Diego.
- Bailey, J., & Hammerstrom, D. (1986, July). *How to make a billion connections* (Tech. Rep. Tech. Report CS/E-86-007). Department of Computer Science and Engineering, Oregon Graduate Institute.
- Bailey, J., & Hammerstrom, D. (1988, July). Why VLSI implementations of associative VLCNs require connection multiplexing. In *International Conference on Neural Networks*.
- Baker, T. (1990). Implementation limits for artificial neural networks. Master's thesis, Oregon Graduate Institute.
- Baker, T., & Hammerstrom, D. (1989a). Characterization of artificial neural network algorithms. In *Proceedings of the 1989 IEEE International Symposium on Circuits and Systems*. Portland, OR.
- Baker, T., & Hammerstrom, D. (1989b). *Modifications to artificial neural networks models for digital hardware implementation* (Tech. Rep. CS/E 88-035). Department of Computer Science and Engineering, Oregon Graduate Institute.
- Baldwin, J. M. (1896). A new factor in evolution. *American Naturalist*, 30, 441-451.
- Barnard, E., & Cole, R. A. (1989, July). *A neural-net training program based on conjugate-gradient optimization* (Tech. Rep. Technical Report No. CS/E 89-014). Oregon Graduate Institute, Department of Computer Science and Engineering.
- Barto, A. G. (1985, April). *Learning by statistical cooperation of self-interested neuron-like computing elements* (Tech. Rep. Technical Report 85-11). COINS.

- Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983, September/October). Neuron-like adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5).
- Baum, E. B., Moody, J., & Wilczek, F. (1988). Internal representations for associative memory. *Biological Cybernetics*, 59, 217-228.
- Beck, J. V., & Arnold, K. J. (1977). *Parameter estimation in engineering and science*. John Wiley & Sons.
- Belew, R., McInerney, J., & Schraudolph, N. (1990, June). *Evolving networks: Using the genetic algorithm with connectionist learning* (Tech. Rep. CS90-174). San Diego: University of California, Computer Science and Engineering Department.
- Belew, R. K. (1989, September). *Evolution, learning and culture: Computational metaphors for adaptive search* (Tech. Rep. CSE Technical Reoprt #CS89-156). UC San Diego.
- Bethke, A. D. (1980). *Genetic algorithms as function optimizers*. PhD thesis, University of Michigan. *Dissertation Abstracts Internations*, 41(9), 3503B. (University Microfilms No. 8106101).
- Brady, R. M. (1985, October). Optimization strategies gleaned from biological evolution. *Nature*, 317.
- Bridges, C. L., & Goldberg, D. E. (1991). The nonuniform Walsh-schema transformation. In G. J. E. Rawlings (Ed.), *Foundations of Genetic Algorithms*, 13-22. Morgan Kaufman.
- Carpenter, G. A., & Grossberg, S. (1986, February). A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer Vision, Graphics, and Image Processing*.
- Caruana, R. A., & Schaffer, J. D. (1988). Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In *Proceedings of the 5th International Conference on Machine Learning*. Morgan Kaufman.
- Casti, J. L., & Karlqvist, A. (Eds.) (1986). *Complexity, language, and life: Mathematical approaches*. Springer-Verlag.
- Chua, L. O., & Yang, L. (1988, October). Cellular neural networks: Theory and applications. *IEEE Transactions on Circuits and Systems*, 35(10), 1257-1290.

- Daugman, J. G. (1988, July). Complete discrete 2-d Gabor transforms by neural networks for image analysis and compression. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7).
- Davidor, Y. (1991). Epistasis variance: A viewpoint on GA-hardness. In G. J. E. Rawlings (Ed.), *Foundations of Genetic Algorithms*, 23–35. Morgan Kaufman.
- Davis, L. (Ed.) (1987). *Genetic algorithms and simulated annealing*. Pitman: London.
- Davis, L. (Ed.) (1991). *Handbook of genetic algorithms*. Van Nostrand - Reinhold.
- Davis, T. E., & Principe, J. C. (1991). A simulated annealing like convergence theory for the simple genetic algorithm. In R. K. Belew, & L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, 174–181. Morgan Kaufmann.
- Dodd, N. (1989). Optimisation of network structure using genetic techniques. Presented at NIPS89 Workshop: Genetic Algorithms and Artificial Neural Networks (workshops not included in proceedings).
- Dodd, N. (1991). Optimisation of network structure using genetic techniques. In G. Rzevski, & R. A. Adey (Eds.), *Applications of Artificial Intelligence in Engineering VI*.
- Dress, W. B. (1987a). Darwinian optimization of synthetic neural systems. In *Proceedings of the IEEE First Annual International Conference on Neural Networks*.
- Dress, W. B. (1987b). Frequency-coded artificial neural networks: An approach to self-organizing systems. In *Proceedings of the IEEE First Annual International Conference on Neural Networks*.
- Dress, W. B., & Knisley, J. R. (1987, October). A Darwinian approach to artificial neural systems. In *1987 IEEE Conference on Systems, Man, and Cybernetics*.
- Eiben, A. E., Aarts, E. H. L., & Hee, K. M. V. (1990). Global convergence of genetic algorithms: On infinite Markov chain analysis. In *First International Workshop on Problem Solving from Nature*.
- Eshelman, L. J. (1991, July). The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In G. J. E. Rawlings (Ed.), *Foundations of Genetic Algorithms*, 265–283, Bloomington, Indiana. Morgan Kaufmann.

- Fahlman, S. E. (1989a). Faster-learning variations on back-propagation: An empirical study. In D. Touretzky, G. Hinton, & T. Sejnowski (Eds.), *Proceedings of the 1988 Connectionist Models Summer School*, 38–51. Morgan Kaufmann.
- Fahlman, S. E. (1989b). Faster-learning variations on back-propagation: An empirical study. In D. Touretzky, G. Hinton, & T. Sejnowski (Eds.), *Proceedings of the 1988 Connectionist Models Summer School*. Morgan Kaufmann.
- Fahlman, S. E. (1990). Personal communication.
- Fahlman, S. E., & Lebiere, C. (1990). The cascade-correlation learning architecture. In *Advances in Neural Information Processing Systems 1*.
- Fitzpatrick, J. M., & Grefenstette, J. J. (1988). Genetic algorithms in noisy environments. *Machine Learning*, 3, 101–120.
- Fitzpatrick, J. M., Grefenstette, J. J., & Van Gucht, D. (1984). Image registration by genetic search. *Proceedings of IEEE Southeast Conference*, 460–464.
- Foldiak, P. (1989). Adaptive network for optimal linear feature extraction. In *IJCNN Proceedings*.
- Fukushima, K. (1981, January). *Cognitron: A self-organizing multilayered neural network model* (Tech. Rep. Technical Monograph No. 30). NHK.
- Fukushima, K., Miyake, S., & Ito, T. (1983, September/October). Neocognitron: A neural network model for a mechanism of visual pattern recognition. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5), 826–834.
- Gierer, A. (1988). Spatial organization and genetic information in brain development. *Biological Cybernetics*, 59, 13–21.
- Goldberg, D. E. (1983). Computer-aided gas pipeline operation using genetic algorithms and rule learning (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International*, 44(10), 3174B. (University Microfilms No. 8402282).
- Goldberg, D. E. (1989a). Genetic algorithms and Walsh functions: Part I, a gentle introduction. *Complex Systems*, 3, 129–152.
- Goldberg, D. E. (1989b). Genetic algorithms and Walsh functions: Part II, deception and its analysis. *Complex Systems*, 3, 153–171.

- Goldberg, D. E. (1989c). *Genetic algorithms in search, optimization, and machine learning*. Reading, MA: Addison-Wesley.
- Goldberg, D. E. (1990a). *Construction of higher-order deceptive functions using low-order Walsh coefficients* (Tech. Rep. Report Number 90002). IlliGAL: The Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois at Urbana-Champaign.
- Goldberg, D. E. (1990b). *A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing* (TCGA Report No. 90003). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.
- Goldberg, D. E. (1991a). Personal communication.
- Goldberg, D. E. (1991b). Personal communication.
- Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. In G. J. E. Rawlings (Ed.), *Foundations of Genetic Algorithms*, 69-93. Morgan Kaufman.
- Goldberg, D. E., Deb, K., & Korb, B. (1990). *An investigation of messy genetic algorithms* (TCGA Report No. 90005). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.
- Goldberg, D. E., Korb, B., & Deb, K. (1989). *Messy genetic algorithms: Motivation, analysis, and first results* (TCGA Report No. 89003). Tuscaloosa: University of Alabama, The Clearinghouse for Genetic Algorithms.
- Goldberg, D. E., & Rudnick, M. (1990). *Schema variance from Walsh-schema transforms* (Tech. Rep. OGI/CSE 90-011). Department of Computer Science and Engineering, Oregon Graduate Institute; and The Clearinghouse for Genetic Algorithms, University of Alabama.
- Goldberg, D. E., & Rudnick, M. (1991). Genetic algorithms and the variance of fitness. *Complex Systems*, 265-278.
- Goldberg, D. E., & Segrest, P. (1987). Finite Markov chain analysis of genetic algorithms. *Proceedings of the Second International Conference on Genetic Algorithms*, 1-8.
- Granger, R., Ambros-Ingerson, J., Henry, H., & Lynch, G. (1987). Partitioning of sensory data by a cortical network. In *NIPS*.

- Granger, R., Ambros-Ingerson, J., Staubli, U., & Lynch, G. (1989). Memorial operation of multiple, interacting simulated brain structures. In M. Gluck and D. Rumelhart (Ed.), *Neuroscience and Connectionist Models*. L. Erlbaum Assoc.
- Grefenstette, J. (Ed.) (1985). *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*. Lawrence Erlbaum Assoc.
- Grefenstette, J. (Ed.) (1987). *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*. Lawrence Erlbaum Assoc.
- Grefenstette, J. (year unknown). *A user's guide to genesis*. Navy Center for Applied Research in Artificial Intelligence, Naval Research Laboratory.
- Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(1), 122-128.
- Grefenstette, J. J., & Fitzpatrick, J. M. (1985). Genetic search with approximate functions. In J. Grefenstette (Ed.), *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*. Lawrence Erlbaum Assoc.
- Gutierrez, M., Wang, J., & Grondin, R. (1989). Estimating hidden unit number for two-layer perceptrons. In *IJCNN Proceedings*.
- Hammerstrom, D. (1986, August). *A connectivity analysis of recursive, auto-associative connection networks* (Tech. Rep. CS/E-86-009). Department of Computer Science and Engineering, Oregon Graduate Institute.
- Hammerstrom, D. (1988, November). The connectivity analysis of simple association — or — how many connections do you need? In D. Anderson (Ed.), *Advances in Neural Information Processing Systems 1*. American Institute of Physics.
- Hammerstrom, D. (1989). Willamette Valley ACM Lecture; Portland, OR.
- Hammerstrom, D., Bahr, C., Bailey, J., Baker, T., Beaver, G., Jagla, K., Mates, J., May, N., McCartor, H., & Rudnick, M. (1987). The OGC cognitive architecture project. In *Northcon 1988 Proceedings*.
- Hammerstrom, D., Bailey, J., Mates, J., & Rudnick, M. (1989). Silicon association cortex. In S. F. Zornetzer, J. L. Davis, & C. Lau (Eds.), *An Introduction to Neural and Electronic Networks* (307-316). Academic Press.

- Hancock, P. (1990, August). *Gannet: Design of a neural net for face recognition by genetic algorithm* (Tech. Rep. CCCN-6). Centre for Cognitive and Computation Neuroscience, Stirling University.
- Harp, S. A., Samad, T., & Guha, A. (1989a). *Genetic synthesis of neural networks* (Tech. Rep. CSDD-89-14852-2). Honeywell.
- Harp, S. A., Samad, T., & Guha, A. (1989b). Towards the genetic synthesis of neural networks. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, 360–369. Morgan Kaufmann.
- Harp, S. A., Samad, T., & Guha, A. (1990, November). Designing application-specific neural networks using the genetic algorithm. In *Advances in Neural Information Processing Systems 2*.
- Hecht-Nielsen, R. (1987, December). Counterpropagation networks. *Applied Optics*, 26(23).
- Hestenes, D. (1986). How the brain works: the next great scientific revolution. In C. R. Smith (Ed.), *Maximum Entropy and Bayesian Spectral Analysis and Estimation Problems*. publisher unknown.
- Hinton, G. (1987). Learning translation invariant recognition in a massively parallel network. In *Parle: Parallel Architectures and Languages Europe* (Vol. 258, 1–13). Springer-Verlag.
- Hinton, G. E., & Nolan, S. J. (1986). *How learning can guide evolution* (Tech. Rep. CMU-CS-86-128). CMU.
- Hinton, G. E., & Nowlan, S. J. (1987). How learning can guide evolution. *Complex Systems*, 495–502.
- Hinton, G. E., Sejnowski, T. J., & Ackley, D. H. (1984, May). Boltzmann machines: Constraint satisfaction networks that learn.
- Hoffmann, G. W., & Benson, M. W. (1986). Neurons with hysteresis form a network that can learn without any changes in synaptic connection strengths. In *Snowbird 1986 Proceedings*.
- Holdaway, R. M. (1989). Enhancing supervised learning algorithms via self-organization. In *IJCNN'89*.

- Holland, J. H. (1973). Genetic algorithms and the optimal allocations of trials. *SIAM Journal of Computing*, 2(2), 88-105.
- Holland, J. H. (1975a). *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press.
- Holland, J. H. (1975b). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press.
- Holland, J. H. (1984). Genetic algorithms and adaptation. In O. G. Selfridge, et al. (Eds.), *Adaptive control of ill defined systems*. Plenum Press.
- Honavar, V., & Uhr, L. (1989). A network of neuron-like units that learns to perceive by generation as well as reweighting of its links. In D. Touretzky, G. Hinton, & T. Sejnowski (Eds.), *Proceedings of the 1988 Connectionist Models Summer School*, 472-484. Morgan Kaufmann.
- Hopfield, J. J. (1982, April). Neural networks and physical systems with emergent collective computational abilities. *Proc. Natl. Acad. Sci.*, 79, 2554-2558.
- Hornik, K., Stinchcombe, M., & White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. *Neural Networks*, 551-560.
- Jong, K. A. D. (1975). *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan.
- Keeler, J. D. (1986). Basin of attraction of neural network models. In *Snowbird Conference Proceedings*, 259-264.
- Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598), 671-680.
- Kitano, H. (1990). Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 461-476.
- Klopf, A. H. (1987, October). *A neuronal model of classical conditioning* (Tech. Rep. AFWAL-TR-87-1139). AFWAL.
- Kohonen, T. (1988). *Self-organization and associative memory*. Springer Series in Information Sciences. Springer-Verlag.

- Kosko, B. (1987a). Adaptive bidirectional associative memories. *Applied Optics*, 26(23), 4947-4960.
- Kosko, B. (1987b). Constructing an associative memory. *Byte*.
- Kruschke, J. K. (1989). Creating local and distributed bottlenecks in hidden layers of back-propagation networks. In D. Touretzky, G. Hinton, & T. Sejnowski (Eds.), *Proceedings of the 1988 Connectionist Models Summer School*, 38-51. Morgan Kaufmann.
- Lansner, A., & Ekeberg, O. (1989). *A one-layered feedback artificial neural network with a Bayesian learning rule* (Tech. Rep. TRITA-NA-P8910). Stockholm, Sweden: Department of Numerical Analysis and Computing Science, Royal Institute of Technology.
- Lazzaro, J., Rychebusch, S., Mahowald, M. A., & Mead, C. A. (year unknown). Winner-take-all networks of $o(n)$ complexity.
- Le Cun, Y., Denker, J., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1990a). Handwritten digit recognition with a back-propagation network. In *Advances in Neural and Information Processing Systems 2*.
- Le Cun, Y., Denker, J., Solla, S. A., Howard, R. E., & Jackel, L. D. (1990b). Optimal brain damage. In *Advances in Neural and Information Processing Systems 2*.
- Leen, T., Rudnick, M., & Hammerstrom, D. (1990). Hebbian feature discovery improves classifier efficiency. In *IJCNN-90, San Diego*.
- Liepins, G. E., & Vose, M. D. (1990). Representational issues in genetic optimization. *J. Exp. Theor. Artif. Intell.*
- Lindenmayer, A. (1968). Mathematical models for cellular interactions in development. *J. Theor. Biol.*, 280-299.
- Lindenmayer, A. (1971). Development systems without cellular interactions, their languages and grammars. *J. Theor. Biol.*, 455-484.
- Linsker, R. (1988, March). Self-organization in a perceptual network. *Computer*.
- Lynch, G., Granger, R., Larson, J., & Baudry, M. (year unknown). Cortical encoding of memory: Hypotheses derived from analysis and simulation of physiological learning rules in anatomical structures.

- Marks II, R. J., Oh, S., Atlas, L. E., & Ritcey, J. A. (1987). *Alternating projection neural networks* (Tech. Rep. 11587). Interactive System Design Lab, University of Washington.
- McClelland, J. L., & Rumelhart, D. E. (Eds.) (1988). *Explorations in parallel distributed processing: A handbook of models, programs, and exercises*, Vol. 3. MIT Press.
- McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 115-133.
- McMahon, M. B., & Fox, B. (1991). Genetic operators for sequencing problems. In G. J. E. Rawlings (Ed.), *Foundations of Genetic Algorithms*, 284-300. Morgan Kaufman.
- Means, E. (1989). Personal Communication.
- Miller, G. F., Todd, P. M., & Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, 379-384. Morgan Kaufmann.
- Mjolsness, E., Sharp, D. H., & Alpert, B. K. (1987, June). Recursively generated neural networks. In *ICNN Proceedings*. V3.
- Mjolsness, E., Sharp, D. H., & Alpert, B. K. (1988a). Genetic parsimony in neural nets. Snowbird Abstracts.
- Mjolsness, E., Sharp, D. H., & Alpert, B. K. (1988b). *Scaling, machine learning, and genetic neural nets* (Tech. Rep. YALEU/DCS/TR-613). Yale.
- Mjolsness, E., Sharp, D. H., & Alpert, B. K. (1988c). *Scaling, machine learning, and genetic neural nets* (Tech. Rep. LA-UR-88-142). Los Alamos.
- Moody, J., & Darken, C. (1988, September). *Learning with localized receptive fields* (Tech. Rep. YALE/DCS/RR-649). Yale University.
- Mozer, M. C., & Smolensky, P. (1989). *Skeletonization: A technique for trimming the fat from a network via relevance assessment* (Tech. Rep. CU-CS-421-89). University of Colorado at Boulder, Department of Computer Science.
- Nix, A. E., & Vose, M. D. (year unknown). *Modeling genetic algorithms with Markov chains*. Submitted to Annals of Mathematics and Artificial Intelligence.

- Pearlmutter, B. A., & Hinton, G. E. (1986). G-maximization: An unsupervised learning procedure for discovering regularities. Snowbird.
- Pellionisz, A., & Llinas, R. (1982). Space-time representation in the brain: The cerebellum as a predictive space-time metric tensor. *Neuroscience*, 7.
- Personnaz, L., & Dreyfus, G. (1988). Investigations into the effect of numerical resolution on the performance of back propagation. In *Neural Networks from Models to Applications, E.S.P.C.I., France*.
- Peterson, C., & Anderson, J. R. (1987, August). *A mean field theory learning algorithm for neural networks* (Tech. Rep. EI-259-87). MCC.
- Pineda, F. J. (1987, November). Generalization of back-propagation to recurrent neural networks. *Physical Review Letters*, 59(19).
- Plumbley, M. D., & Fallside, F. (1988). An information-theoretic approach to unsupervised connectionist models. In D. Touretzky, G. Hinton, & T. Sejnowski (Eds.), *Proceedings of the Connectionists Models Summer School*. Morgan-Kaufmann.
- Reeke, G. N., Jr., & Edelman, G. M. (1987, spring). Selective neural networks and their implications for recognition automata. *The International Journal of Supercomputer Applications*, 1(1).
- Regier, T. (1990). C language version of Quickprop. Personal communication.
- Ritter, H. (1989). Combining self-organizing maps. In *IJCNN'89*.
- Rosenblat, F. (1962). *Principles of neurodynamics*. Spartan.
- Ross, S. M. (1987). *Introduction to probability and statistics for engineers and scientists*. John Wiley & Sons.
- Rudnick, M. (1990). *Bibliography of the intersection of genetic search and artificial neural networks* (Tech. Rep. CSE 90-001). Oregon Graduate Institute, Department of Computer Science and Engineering.
- Rudnick, M., Cadambi, S., & Hong, C. (1987). *Micro-study for the TBH test chip* (Tech. Rep.). Oregon Graduate Institute, Department of Computer Science and Engineering.

- Rudnick, M., & Hammerstrom, D. (1988a). An interconnect structure for wafer scale neurocomputers. *Neural Networks, 1*. International Neural Network Society Meeting Supplement; Boston, MA.
- Rudnick, M., & Hammerstrom, D. (1988b). An interconnect structure for wafer scale neurocomputers. In D. Touretzky, G. Hinton, & T. Sejnowski (Eds.), *Proceedings of the 1988 Connectionists Models Summer School* (498–512). Morgan Kaufmann.
- Rumelhart, D. (1988). Weight Decay Lecture at 1988 Connectionist Models Summer School.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985, September). *Learning internal representations by error propagation* (Tech. Rep. ICS Report 8506). ICS.
- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986, October 9). Learning representations by back-propagating errors. *Nature*, 323, 533–536.
- Rumelhart, D. E., & McClelland, J. L. (Eds.) (1986). *Parallel distributed processing: Explorations in the microstructure of cognition*, Vol. 1 & 2. MIT Press.
- Rumelhart, D. E., & Zipser, D. (1985). Feature discovery by competitive learning. *Cognitive Science*, 9, 75–112.
- Sanger, T. (1989a). Optimal unsupervised learning in a single-layer linear feedforward neural network. *Neural Networks*, 2(6), 459–473.
- Sanger, T. (1989b). Optimal unsupervised learning in feedforward neural networks. Master's thesis, MIT.
- Sanger, T. (1990). An optimality principle for unsupervised learning. In D. S. Touretzky (Ed.), *Advances in Neural Information Processing Systems 2*.
- Schaffer, J. D. (1984). *Some experiments in machine learning using vector evaluated genetic algorithms*. PhD thesis, Vanderbilt University.
- Schaffer, J. D. (Ed.) (1989). *Proceedings of the Third International Conference on Genetic Algorithms*. Morgan Kaufmann.
- Schaffer, J. D., Caruana, R. A., & Eshelman, L. J. (1990). Using genetic search to exploit the emergent behavior of neural networks. In S. Forest (Ed.), *Proceedings of the Emergent Computation 1989 Conference*.

- Schaffer, J. D., Eshelman, L. J., & Offutt, D. (1991, July). Spurious correlations and premature convergence in genetic algorithms. In G. J. E. Rawlins (Ed.), *Foundations of Genetic Algorithms*, 102-112, Bloomington, Indiana. Morgan Kaufmann.
- Schraudolph, N., & Belew, R. (1990, July). *Dynamic parameter encoding for genetic algorithms* (Tech. Rep. CS90-175). San Diego: University of California, Computer Science and Engineering Department.
- Scofield, C. L. (1988). Learning internal representations in the Coulomb energy network. In *ICNN '88 Proceedings*.
- Sirag, D. J., & Weisser, P. T. (1987). Toward a unified thermodynamic genetic operator. In J. J. Grefenstette (Ed.), *Proceedings of the Second International Conference on Genetic Algorithms*, 116-122. Lawrence Erlbaum Assoc.
- Smith, J. M. (1987, October 29). When learning guides evolution. *Nature*, 329, 761-762.
- Solla, S. A. (1988). Learning and generalization in layered neural networks: The contiguity problem.
- Specht, D. F. (1988). Probabilistic neural networks for classification, mapping, or associative memory. In *ICNN '88 Proceedings*.
- Sun, G. Z., Chen, H. H., & Lee, Y. C. (1988, April). *Parallel sequential induction network: A new paradigm of neural network architecture* (Tech. Rep. UMIACS-TR-88-26, CS-TR-2013). University of Maryland.
- Tesauro, G. (1986). Simple neural models of classical conditioning. *Biol. Cybern.*, 55, 187-200.
- Todd, P. (1988, March). Evolutionary methods for connectionist architectures. Psychology Department, Stanford University. Unpublished.
- Torras i Genis, C. (1986, September/October). Neural network model with rhythm-assimilation capacity. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-16(5).
- von Neumann, J. (1987). *Papers of John von Neumann on computing and computer theory*, Vol. 12 of *Charles Babbage Institute Reprint Series for the History of Computing*. MIT Press.

- Vose, M. D., & Liepins, G. E. (1991). Schema disruption. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, 237-243. Morgan Kaufmann.
- Waddington, C. H. (1942). Canalization of development and the inheritance of acquired characters. *Nature*, 150, 563-565.
- Waibel, A. (1989). Connectionist glue: Modular design of neural speech systems. In D. Touretzky, G. Hinton, & T. Sejnowski (Eds.), *Proceedings of the 1988 Connectionists Models Summer School*. Morgan Kaufmann.
- Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. J. (1989, March). Phoneme recognition using time-delay neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(3).
- Waibel, A., Sawai, H., & Shikano, K. (1988, August). *Modularity and scaling in large phonemic neural networks* (Tech. Rep. TR-I-0034). ATR Interpreting Telephony Research Laboratories.
- Whitley, D. (1989). The *genitor* algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, 116-121. Morgan Kaufmann.
- Whitley, D., Starkweather, T., & Bogart, C. (1989, November). *Genetic algorithms and neural networks: Optimizing connections and connectivity* (Tech. Rep. CS-89-117 (subsumes CS-89-113 & CS-89-114)). Colorado State University.
- Wieland and Leighton (1988). Shaping schedules as a method for accelerating learning. *Neural Networks*, 1, Supplement 1, 231. Abstracts of the First Annual INNS Meeting.
- Williams, R. J. (1987, February). *Reinforcement-learning connectionist systems* (Tech. Rep. NU-CCS-87-3). Institution unknown.
- Wilson, S. W. (1987). Classifier systems and the animat problem. *Machine Learning*, 2, 199-228.
- Wilson, S. W. (1990). Preceptor redux: Evolution of structure. In S. Forest (Ed.), *Proceedings of the Emergent Computation 1989 Conference*.

Biographical Note

Mike Rudnick, a native Oregonian since his 1949 birth in Portland, graduated from Tigard High School in 1967. After his graduation in mathematics from Portland State University in 1972, he joined Tektronix, Inc., where he worked as a software engineer until 1983, when he left to perform contract software engineering.

In 1986 he began his graduate career at the Oregon Graduate Institute (then called Oregon Graduate Center) where he completed his Ph.D. in the Department of Computer Science and Engineering in 1992. Throughout graduate school he performed research in biologically inspired artificial intelligence, initially as a member of Dan Hammerstrom's Cognitive Architecture Project. During the last two years he performed dissertation research working with Dr. David E. Goldberg, first at The Clearinghouse for Genetic Algorithms at the University of Alabama, and later at The Illinois Genetic Algorithms Laboratory at the University of Illinois Urbana-Champaign.

The author attended the 1988 Connectionists Models Summer School at Carnegie-Mellon University and is co-inventor on a neurocomputing patent application. He also created and currently administers the Neuro-evolution electronic forum, which is dedicated to all aspects of the joint use of artificial neural networks and genetic algorithms, and especially evolutionary network design.

Mike currently has a temporary faculty appointment at Willamette University, Salem, Oregon, and an adjunct faculty appointment at the Oregon Graduate Institute of Science & Technology.