WALSH FUNCTION ANALYSIS OF GENETIC ALGORITHMS OF NON-BINARY STRINGS

BY

CHRISTOPHER KIANKHO OEI

B.S., California Institute of Technology, 1990
M.S., University of Illinois, 1991

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1992

Urbana, Illinois

WALSH FUNCTION ANALYSIS OF GENETIC ALGORITHMS OF NON-BINARY STRINGS

Christopher Kiankho Oei, M.S.
Department of Computer Science
University of Illinois at Urbana-Champaign, 1992
David E. Goldberg, Advisor

The Walsh transform is used extensively as a tool in determining whether a fitness function over a binary string is deceptive or not. This thesis shows that the Walsh transform method for detecting deception is easily generalized to functions over non-binary strings such as ternary strings, strings with real parameters, and strings with some binary and ternary characters and some real parameters. A generalization of the Hadamard transform is then used to organize the generalized Walsh coefficients into conditions for static deception for non-binary alphabets. The variances of fitness of schemata are calculated using generalized Walsh coefficients. Mathematica code for performing most of the calculations mentioned is included.

Contents

Chapter 1

Introduction

1.1 Preview

Walsh functions, introduced by the mathematician J. L. Walsh in 1923, recently have become popular tools in fields such as telecommunications engineering, radar systems, image recognition and processing, speech processing, coding systems, and spectroscopy (Beauchamp, 1975). Bethke (1981) introduced the Walsh-schema transform, a method of using Walsh functions for computing schema averages. Since then, the Walsh-schema transform has become a cornerstone in the theory of genetic algorithms (GAs).

Most of the theory of genetic algorithms applies to GAs over binary strings, real numbers, or permutations. Attempts to generalize this theory borrowed some notions

from set theory (Radcliffe, 1991; Vose & Liepins, 1991). Recently, Mason (1991) extended the concept of partition coefficients (Bethke, 1981) from the theory of binary-coded genetic algorithms to GAs over non-binary strings.

This thesis generalizes Walsh functions to non-binary strings and reworks some of the existing theory of GAs to incorporate these new functions. Topics that will be covered and extended include Bethke's Walsh schema transform, real-coded GAs (Bledsoe, 1961; Goldberg, 1990a; Wright, 1991), detecting static deception and Hadamard transforms (Homaifar & Qi, 1990; Homaifar, Qi, & Fost, 1991), and the variance of fitness (Goldberg, Deb, & Clark, 1991; Goldberg & Rudnick, 1991; Rudnick & Goldberg, 1991).

1.2 Notation

Throughout this thesis, n will refer to the length of the strings in the domain of the fitness function. k will refer to the cardinality of the alphabet, or if each character of a string is taken from a different alphabet, k_1 refers to the alphabet for the first character, k_2 for the second, and so on. A k-ary alphabet is an alphabet with k characters. The characters in a k-ary alphabet will be represented by the integers 0 through $k-1$. A k-ary string is a string all of whose characters are taken from the same k-ary alphabet. A string that belongs to a $\vec{k}$-ary alphabet is a string whose first character belongs to a k_1-ary alphabet, whose second character belongs to a k_2-ary

alphabet, etc; such strings will be referred to as $\vec{k}$-ary strings. A string will be represented by a vector whose components are the integer representations of the characters in the string. For example, the hexidecimal (16-ary) string "3F2" would be denoted by (3,15,2). k-ary Walsh functions are the generalization of Walsh functions to k-ary alphabets, and $\vec{k}$-ary Walsh functions are the generalization to strings that belong to $\vec{k}$-ary alphabets. Both of these generalizations of Walsh functions, as well as generalizations of the Walsh functions to strings with characters which are real numbers, will be referred to as generalized Walsh functions. The corresponding transforms will be referred to as k-ary and $\vec{k}$-ary Walsh transforms.

Chapter 2

History

The question of what kind of problem is difficult for a GA started with Bledsoe (1961). Bledsoe described a situation he called *lethal dependence* in which a mutation in each of two genes would be an improvement, but a mutation in either gene alone would lead to death.

Following Holland's schema theorem (Holland, 1975), Bethke (1981) introduced the Walsh-schema transform for computing schema averages and gave some intuitive conditions for problem difficulty that depended on the smoothness of the fitness function and the asymptotic behavior of the Walsh coefficients.

The smoothness arguments were also used by Weinberger, who started with Eigen's model for natural selection (Weinberger, 1987) and studied correlation lengths on the "landscape" of the fitness function (Weinberger, 1988; Weinberger, 1990). Kauffman

in (Kauffman, 1989; Kauffman, 1990; Kauffman & Levin, 1987) created and analyzed a model fitness landscape with tunable ruggedness, which was analyzed further in Manderick, de Weger, & Spiessens (1991). Lipsitch (1991) used cellular automata rules to generate fitness landscapes and performed simulations to discover which classes of cellular automaton created the hardest landscapes. These analyses, however, dealt with hill-climbing on fitness functions and neglected the effects of crossover and the usefulness of schemata. Also, Goldberg (1990b) pointed out that the asymptotic behavior of the Walsh coefficients (and therefore the smoothness of the function) is not enough to insure the growth of important schemata.

Holland's schema theorem came back into play when Goldberg (1987) defined deception and introduced the minimal deceptive problem. The conditions for full static deception, a situation in which all low-order schemata are misleading, was introduced in Goldberg (1989b). The analysis of full static deception taking into account the schema disruption due to crossover and mutation was done in Goldberg (1989c) using operator-adjusted Walsh coefficients.

This line of reasoning gave rise to a host of techniques: a method for analyzing a GA population in which schemata are not distributed uniformly (Bridges, Goldberg, 1989), a method that uses Hadamard transforms to organize the deceptive conditions (Homaifar & Qi, 1990; Homaifar, Qi, & Fost, 1991), methods for constructing fully deceptive and intermediate deceptive functions (Deb & Goldberg, 1991; Goldberg,

1990b; Liepins & Vose, 1991; Whitley, 1991a), and a simpler set of criteria sufficient to insure deception (Deb & Goldberg, 1992). This theory of deception has been used to explain some experimental results, such as why a problem that is easy for a GA might be difficult for a hill-climber (Wilson, 1991), and why certain GA test suites were solved so easily (Das & Whitley, 1991; Davis, 1991). It also lead to some attacks (Forrest & Mitchell, 1991; Grefenstette, 1991; Mitchell & Forrest, 1991; Mitchell, Forrest, & Holland, 1991; Tanese, 1989) that the lack of deception, where deception is defined by misleading schemata averages, is not enough by itself to insure GA convergence to the global optimum.

Another mode of GA failure, apart from deception, has to do with the variance of schema fitnesses and sampling error (Davidor, 1991; Liepins & Vose, 1990b; Schaffer, Eschelman & Offut, 1991). There were a number of studies that introduced techniques for calculating schema fitness variances and signal-to-noise ratios and explain how to size a GA population accordingly (Goldberg, Deb, & Clark, 1991; Goldberg & Rudnick, 1991; Rudnick, 1991; Rudnick & Goldberg 1991). Connected with the issue of variance is multimodality, or having many high peaks that may confuse the GA (Goldberg, Deb & Horn, 1992). Goldberg, Deb, & Clark (1991) give an overview and a summary of conditions for GA success.

Generalizations of this basic theory of convergence include applications to permutation problems (Kargupta, Deb, & Goldberg, 1992; Sikora, 1991) and fitness

functions with real parameters (Goldberg, 1990a; Wright, 1991). On a more abstract level are generalizations of the schema notion to arbitrary linear combinations of bits (Liepins & Vose, 1990a), and arbitrary predicates (Vose & Liepins, 1991; Radcliffe, 1991). Mason (1991) generalized the notions of partition coefficients and static deception to finite non-binary alphabets.

The other approaches to the question of GA convergence are few and sparse. Hart and Belew (1991) points out that the general problem that the GA tries to solve is NP-hard. Kauffman (1990) uses some concepts from information theory and the physics of phase transitions to show a connection between information redundancy and evolvability: minimal systems are not evolvable due to lethally dependent parameters.

The work done on binary-coded GAs provides an extensive theoretical framework that hinges on the Walsh-schema transform. This thesis will focus on the theory of static deception generalized to non-binary alphabets and begins by generalizing the Walsh transform.

Chapter 3

Generalized Walsh Functions and Transforms

3.1 Functions Over k-ary Strings

This chapter examines functions over strings of length n whose characters are taken from a k-ary alphabet. For example, we can use this method to analyze a function over a ternary alphabet string. The basic idea is to treat each character in the string as a separate dimension, then take the n-dimensional Fourier series. The exact similarity between the generalized Walsh transform and Fourier series will be discussed later. The point is that it is useful to think of the Walsh functions as functions of n variables rather than functions of a single variable.

3.1.1 k-ary Walsh Functions

Definition 1 *Define the k-ary Walsh functions as*

$$\Psi_{\vec{j}}^{(k)}(\vec{x}) = \frac{1}{\sqrt{k^n}} e^{\frac{2\pi\imath}{k}\vec{x}\cdot\vec{j}}, \tag{3.1}$$

where the vector $\vec{j} = (j_1, j_2, \ldots, j_n)$ *is the k-ary representation of* j*, and the vector* $\vec{x} = (x_1, x_2, \ldots, x_n)$ *is the k-ary representation of* x*.*

Theorem 1 *The k-ary Walsh functions satisfy the following normalization condition:*

$$\sum_{\vec{x}} \overline{\Psi}_{\vec{j}}^{(k)}(\vec{x}) \Psi_{\vec{l}}^{(k)}(\vec{x}) = \delta_{j_1 l_1} \delta_{j_2 l_2} \ldots \delta_{j_n l_n} \tag{3.2}$$

where barred quantities refers to the complex conjugate and δ *is the Kronecker delta. The sum is over all distinct values of* $\vec{x}$*.*

Proof

$$\begin{aligned}
\sum_{\vec{x}} \overline{\Psi}_{\vec{j}}^{(k)}(\vec{x}) \Psi_{\vec{l}}^{(k)}(\vec{x}) &= \frac{1}{k^n} \sum_{x_1=0}^{k-1} \sum_{x_2=0}^{k-1} \ldots e^{-\frac{2\pi\imath}{k} x_1 j_1} e^{\frac{2\pi\imath}{k} x_1 l_1} e^{-\frac{2\pi\imath}{k} x_2 j_2} e^{\frac{2\pi\imath}{k} x_2 l_2} \ldots; \\
&= (\frac{1}{k} \sum_{x_1=0}^{k-1} e^{\frac{2\pi\imath}{k} x_1 (l_1 - j_1)}) (\frac{1}{k} \sum_{x_2=0}^{k-1} e^{\frac{2\pi\imath}{k} x_2 (l_2 - j_2)}) \ldots; \\
&= \delta_{j_1 l_1} \delta_{j_2 l_2} \ldots \delta_{j_n l_n}.
\end{aligned} \tag{3.3}$$

When $k = 2$, the k-ary Walsh functions become the usual Walsh functions, up to a normalization constant. Throughout this thesis, the normalization is chosen so that the inner product of generalized Walsh functions is a Kronecker delta function. This differs from the usual normalization convention for Walsh functions, but it has an

intuitive appeal when deriving theorems and proofs; these functions are orthonormal, not merely orthogonal.

Example Take strings of length 2, $n = 2$, using a ternary alphabet, $k = 3$. Evaluating the fourth function, $\vec{j} = (1,1)$, at the second position, $\vec{x} = (0,2)$ gives the following:

$$\Psi^{(3)}_{(1,1)}((0,2)) = \frac{1}{\sqrt{3^2}} e^{\frac{2\pi\imath}{3}(1,1)\cdot(0,2)} = -\frac{1}{6} - \frac{\sqrt{3}}{6}\imath. \tag{3.4}$$

Another way of looking at the Walsh functions $\Psi^{(k)}_{\vec{j}}(\vec{x})$ is to think of them as (up to normalization) $e^{\imath\phi}$ where the phase ϕ is the inner product between the index of the function $\vec{j}$ and the position $\vec{x}$ and the distance metric for the inner product is such that the phase increases by 2π as we traverse a dimension.

3.1.2 k-ary Walsh Transform

Now that the k-ary Walsh functions are defined, the k-ary Walsh transform can be stated. To put it simply: a k-ary Walsh coefficient is the inner product of the fitness function with a k-ary Walsh function, and the k-ary Walsh transform gives the k-ary Walsh coefficients in terms of the fitness values.

Definition 2 (k-ary Walsh Transform) *The Walsh coefficients w of a function f are given by*

$$w_{\vec{\jmath}} = \sum_{\vec{x}} \overline{\Psi}^{(k)}_{\vec{\jmath}}(\vec{x}) f(\vec{x}). \tag{3.5}$$

Notice that we use the complex conjugate of the Walsh function.

Example Use a ternary alphabet and length 2 string again. The Walsh coefficient $w_{(0,1)}$ of a function f is given as follows:

$$\begin{aligned} w_{(0,1)} &= \frac{1}{3}e^{0} f((0,0)) + \frac{1}{3}e^{-2\pi\imath/3} f((0,1)) + \frac{1}{3}e^{-4\pi\imath/3} f((0,2)) \\ &+\frac{1}{3}e^{0} f((1,0)) + \frac{1}{3}e^{-2\pi\imath/3} f((1,1)) + \frac{1}{3}e^{-4\pi\imath/3} f((1,2)) \\ &+\frac{1}{3}e^{0} f((2,0)) + \frac{1}{3}e^{-2\pi\imath/3} f((2,1)) + \frac{1}{3}e^{-4\pi\imath/3} f((2,2)). \end{aligned} \tag{3.6}$$

Definition 3 (Inverse k-ary Walsh Transform) *The inverse transform is given by*

$$f(x) = \sum_{\vec{\jmath}} \Psi^{(k)}_{\vec{x}}(\vec{\jmath}) w_j, \tag{3.7}$$

where the sum is over all possible values of the k-ary string $\vec{\jmath}$.

Notice that the Walsh function is not conjugated in the inverse transform. In practice, the generalized Walsh transform and its inverse would be computed by the generalized Fast Walsh Transform algorithm and the inverse generalized Fast Walsh Transform listed in Appendix A.

Theorem 2 *The inverse k-ary Walsh transform of a k-ary Walsh transform is the identity transformation.*

Proof Notice from the definition of the k-ary Walsh function (3.1) that $\Psi_{\vec{x}}^{(k)}(\vec{\jmath}) = \Psi_{\vec{\jmath}}^{(k)}(\vec{x})$.

$$\begin{aligned}
\sum_{\vec{\jmath}} \Psi_{\vec{x}}^{(k)}(\vec{\jmath}) w_{\vec{\jmath}} &= \sum_{\vec{\jmath}} \Psi_{\vec{x}}^{(k)}(\vec{\jmath}) \sum_{\vec{y}} \overline{\Psi}_{\vec{\jmath}}^{(k)}(\vec{y}) f(\vec{y}); \\
&= \sum_{\vec{\jmath}} \Psi_{\vec{x}}^{(k)}(\vec{\jmath}) \sum_{\vec{y}} \overline{\Psi}_{\vec{y}}^{(k)}(\vec{\jmath}) f(\vec{y}); \\
&= \sum_{\vec{y}} f(\vec{y}) \sum_{\vec{\jmath}} \Psi_{\vec{x}}^{(k)}(\vec{\jmath}) \overline{\Psi}_{\vec{y}}^{(k)}(\vec{\jmath}); \\
&= \sum_{\vec{y}} f(\vec{y}) \delta_{x_1 y_1} \delta_{x_2 y_2} \ldots \delta_{x_n y_n}; \\
&= f(\vec{x}). \qquad (3.8)
\end{aligned}$$

The method of using these functions to determine deception will be discussed in a later chapter. The procedure is a straightforward extension of the binary case. First, the orthogonality of the k-ary Walsh functions is used to find k-ary Walsh coefficients for the fitness function to be analyzed. These coefficients are used to compute the schema averages, and the schema averages are used to say something about deception.

3.1.3 Some Theorems and Proofs

The following are a few theorems about the k-ary Walsh transforms. Most of them are simple and add to our intuition of how they work.

	(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)	(2,0)	(2,1)	(2,2)
f_1	0	0	0	3	3	3	12	12	12
f_2	0	1	2	0	1	2	0	1	2
f	0	1	2	3	4	5	12	13	14

Table 3.1: Values of f, f_1, and f_2 for various values of x.

	(0,0)	(0,1)	(0,2)	(1,0)	(1,1)	(1,2)	(2,0)	(2,1)	(2,2)
w_1	15	0	0	$\frac{-15+3^{5/2}\imath}{2}$	0	0	$\frac{-15+3^{5/2}\imath}{2}$	0	0
w_2	3	$\frac{-3+3^{1/2}\imath}{2}$	$\frac{-3+3^{1/2}\imath}{2}$	0	0	0	0	0	0
w	18	$\frac{-3+3^{1/2}\imath}{2}$	$\frac{-3+3^{1/2}\imath}{2}$	$\frac{-15+3^{5/2}\imath}{2}$	0	0	$\frac{-15+3^{5/2}\imath}{2}$	0	0

Table 3.2: Generalized Walsh coefficients of the functions in Table 3.1.

Theorem 3 *A function is additively separable if $f(\vec{x}) = f_1(x_1)+f_2(x_2)+\ldots$. A function is additively separable if and only if its k-ary Walsh transform is also additively separable.*

Proof The proof of this comes immediately from the fact that the transform is linear.

Example Consider again the ternary alphabet and let $f((x_1, x_2)) = 3x_1^2 + x_2$. Let $f_1(x_1, x_2) = 3x_1^2$ and $f_2(x_1, x_2) = x_2$ so that $f(x_1, x_2) = f_1(x_1, x_2) + f_2(x_1, x_2)$. See Tables 3.1 and 3.2 for the function values and generalized Walsh coefficients.

Note that $w_1 + w_2 = w$. Although this theorem is straightforward, it stresses the idea that it is useful to think of the characters in the string as being independent variables. Also, functions which are partially additively separable are often used in testing genetic algorithms. Partially additively separable functions also arise in real applications. For example, in designing a high-performance engine using genetic algorithms, the first three characters in the string might code for the type of steel used, while the next two characters might code for the fuel mixture. Intuitively, these two characteristics are mostly independent; the optimal fuel mixture does not depend strongly upon the type of steel used, and the best kind of steel to use does not depend strongly upon the fuel mixture. We can express the ideas above mathematically as follows:

$$f((x_1, x_2, x_3, x_4, x_5)) = f_{123}(x_1, x_2, x_3) + f_{45}(x_4, x_5) + O(\epsilon) \tag{3.9}$$

where ϵ is small compared to one, and all the functions f, f_{123}, and f_{45}, are of order one. In this case,

$$w_{(j_1,j_2,j_3,j_4,j_5)} = u(j_1, j_2, j_3) + v(j_4, j_5) + O(\epsilon). \tag{3.10}$$

To put it more simply: if the fitness function is well-approximated by a partially additively separable function, then the k-ary Walsh transform is also well-approximated by a partially additively separable function.

Theorem 4 *The average of the function keeping the first character fixed at* 0 *and varying the other characters is given by the following sum:*

$$\frac{1}{k^{n/2}}(w_{(0,0,0,\ldots)} + w_{(1,0,0,\ldots)} + w_{(2,0,0,\ldots)} + \ldots + w_{(k-1,0,0,\ldots)}). \tag{3.11}$$

Note that w_0 is the average of the function (times a normalization constant).

Notice that the summation over all the variables of the function except the first character has been reduced to a single summation in transform space. This is why the k-ary Walsh transform is useful. The average of a function of strings of length n over a schema with m fixed positions can be expressed as a sum over k^m k-ary Walsh coefficients. This idea is used later in the chapter on using generalized Walsh functions to calculate schema averages.

Example Let us take $f((x_1, x_2)) = 3x_1^2 + x_2$ and use a ternary alphabet as in a previous example above. Then the average of the function setting $x_1 = 0$ and letting x_2 vary is

$$\frac{1}{3}\{f((0,0)) + f((0,1)) + f((0,2))\} = \frac{1}{3}(w_{(0,0)} + w_{(1,0)} + w_{(2,0)}). \tag{3.12}$$

Had f been a function of strings of length 3 instead of 2, the left hand side of the above equation would have 9 terms, while the right hand side would still have 3 terms.

Theorem 5 *The average of the square of the absolute value of the function is equal to the average of the square of the absolute value of the k-ary Walsh transform of the function.*

This is analogous to the same relation in Fourier transforms. The usefulness of taking the average of $|f|^2$ will become apparent in the chapter on the variance of fitness.

Proof

$$\begin{aligned}
\sum_{\vec{x}} \overline{f}(\vec{x}) f(\vec{x}) &= \sum_{x} \overline{\sum_{\vec{p}} \Psi^{(k)}_{\vec{x}}(\vec{p}) w_{\vec{p}}} \sum_{\vec{q}} \Psi^{(k)}_{\vec{x}}(\vec{q}) w_{\vec{q}}; \\
&= \sum_{\vec{p},\vec{q}} \sum_{\vec{x}} \overline{\Psi}^{(k)}_{\vec{x}}(\vec{p}) \Psi^{(k)}_{\vec{x}}(\vec{q}) \overline{w}_{\vec{p}} w_{\vec{q}}; \\
&= \sum_{\vec{p},\vec{q}} \sum_{\vec{x}} \overline{\Psi}^{(k)}_{\vec{p}}(\vec{x}) \Psi^{(k)}_{\vec{q}}(\vec{x}) \overline{w}_{\vec{p}} w_{\vec{q}}; \\
&= \sum_{\vec{p},\vec{q}} \delta_{p_1 q_1} \delta_{p_2 q_2} \ldots \delta_{p_n q_n} \overline{w}_{\vec{p}} w_{\vec{q}}; \\
&= \sum_{\vec{p}} \overline{w}_{\vec{p}} w_{\vec{p}}. \qquad (3.13)
\end{aligned}$$

Example Let $f((x_1, x_2)) = 3x_1^2 + x_2$ as before. The sum of squares of f over the entire x space is 564, and so is the sum of the squares of the absolute values of the generalized Walsh coefficients.

Instead of summing the squares of f over the entire x space, let us sum it over just the strings with $x_1 = 0$:

$$\begin{aligned}
\sum_{x_2, x_3, \ldots, x_n} \overline{f(\vec{x})} f(x) &= \sum_{x_2, x_3, \ldots, x_n} \overline{\sum_{\vec{p}} \Psi^{(k)}_{\vec{x}}(\vec{p}) w_{\vec{p}}} \sum_{\vec{q}} \Psi^{(k)}_{\vec{x}}(\vec{q}) w_{\vec{q}}; \\
&= \sum_{\vec{p},\vec{q}} \sum_{x_2, x_3, \ldots, x_n} \overline{\Psi}^{(k)}_{\vec{x}}(\vec{p}) \Psi^{(k)}_{\vec{x}}(\vec{q}) \overline{w}_{\vec{p}} w_{\vec{q}}; \\
&= \sum_{\vec{p},\vec{q}} \sum_{x_2, x_3, \ldots, x_n} \overline{\Psi}^{(k)}_{\vec{p}}(\vec{x}) \Psi^{(k)}_{\vec{q}}(\vec{x}) \overline{w}_{\vec{p}} w_{\vec{q}}; \\
&= \sum_{\vec{p},\vec{q}} \delta_{p_2 q_2} \delta_{p_3 q_3} \ldots \delta_{p_n q_n} \overline{w}_{\vec{p}} w_{\vec{q}};
\end{aligned}$$

$$= \sum_{p_1,q_1} \sum_{p_2,p_3,\ldots,p_n} \overline{w}_{\vec{p}} w_{(q_1,p_2,p_3,p_4,\ldots,p_n)}. \tag{3.14}$$

Notice that the sum on the left has k^{n-1} terms, while the sum on the right has k^{n+1} terms. So working in the transform space makes for more work in this case.

3.2 Functions Over $\vec{k}$-ary Strings

This section considers functions over $\vec{k}$-ary strings, strings whose characters are taken from different alphabets. The ideas from the previous sections in this chapter are still valid, and the definitions need to be modified only slightly.

3.2.1 $\vec{k}$-ary Walsh Functions

Recall that k_p is the size of the alphabet for the p-th character in a string. Define the $\vec{k}$-ary Walsh functions in the following manner:

$$\Psi_{\vec{j}}^{\vec{k}}(\vec{x}) = \frac{1}{\sqrt{k_1 k_2 \ldots k_n}} \prod_{m=1}^{n} \exp(\frac{2\pi\imath}{k_m} x_m j_m). \tag{3.15}$$

Again, the normalization condition is given by the following:

$$\sum_{\vec{x}} \overline{\Psi}_{\vec{p}}^{(\vec{k})}(\vec{x}) \Psi_{\vec{q}}^{(\vec{k})}(\vec{x}) = \delta_{p_1 q_1} \delta_{p_2 q_2} \ldots \delta_{p_n q_n}. \tag{3.16}$$

The proof of the normalization works exactly the same way as it did before. The form of the transform (3.5) and inverse transform (3.7) also remain the same.

3.3 Functions Over Reals

As a computer internally represents real numbers as integers, one might suspect that there is some similarity between analyzing genetic algorithms whose fitness functions are defined over strings whose characters are taken over large alphabets and analyzing GAs with fitness functions over reals. This suspicion is correct, and with this in mind, real variables will be referred to as characters taken from ∞-ary alphabets, strings that have one real variable followed by one ternary character will be referred to as $(\infty, 3)$-ary strings, and so on. Despite the similarities, the analysis of functions with real variables is not exactly the same as the analysis of functions with characters from large alphabets; therefore the ∞ symbol should be taken to signify a real variable rather than a character from an infinitely large alphabet.

Previously, it was demonstrated that the generalized Walsh transform was equivalent to taking the Fourier transform along each dimension. All one needs to do is apply this same idea. If the fitness function is a function of two real variables and three discrete variables, then simply take the Fourier transform along all five dimensions.

To avoid normalization constants, this thesis assumes that the real variables of the fitness function are always in the unit interval [0,1]. One can often rescale the real variables in the fitness function so that this is true. Later, the analysis of functions whose variables are in the range $[0, \infty]$ and $[-\infty, \infty]$ will be discussed.

Consider a function over $(\infty, \infty, 3, 3, 3)$-ary strings. That is, a function that takes two real variables and three ternary characters. The generalized Walsh coefficients of this function are given by:

$$w_{(j_1,j_2,j_3,j_4,j_5)} = 3^{-3/2} \int_0^1 dx_1 \int_0^1 dx_2 \sum_{x_3=0}^{2} \sum_{x_4=0}^{2} \sum_{x_5=0}^{2} f(\vec{x}) e^{-2\pi\imath(x_1 j_1 + x_2 j_2 + x_3 j_3/3 + x_4 j_4/3 + x_5 j_5/3)}. \tag{3.17}$$

And the inverse transform is given by:

$$f(\vec{x}) = 3^{-3/2} \sum_{j_1=-\infty}^{\infty} \sum_{j_2=-\infty}^{\infty} \sum_{j_3=0}^{2} \sum_{j_4=0}^{2} \sum_{j_5=0}^{2} w_{(j_1,j_2,j_3,j_4,j_5)} e^{2\pi\imath(x_1 j_1 + x_2 j_2 + x_3 j_3/3 + x_4 j_4/3 + x_5 j_5/3)}. \tag{3.18}$$

Notice that the sum over j_1 and j_2 run from $-\infty$ to ∞, and that there is no normalization factor associated with x_1 and x_2 because they range from 0 to 1.

In practice, the sum from $-\infty$ to ∞ would be approximated by a sum from $-A$ to A, where A is some large integer constant. This will yield an arbitrarily accurate approximation whenever the infinite sum converges. Although the necessary and sufficient conditions for convergence are beyond the scope of this thesis and is a topic for future research, piecewise smooth functions can always be approximated by this method (Tolstov, 1962), and will be discussed in the next section of this chapter.

3.4 Reduction to Smaller Alphabets

One of the most useful properties about using these generalized Walsh transforms is that functions over reals or large alphabets can sometimes be well-approximated by functions over small alphabets. Consider a function over a single real variable. The generalized Walsh transform is equivalent to the Fourier transform in this case. It is well-known that if the first m derivatives of the function are continuous, and the $m+1$-th derivative is discontinuous, then for large j, the magnitude of the Fourier coefficients fall as j^{-m-2} (Lighthill, 1959). Thus, if the function is smooth enough, then one can get good approximations to schema averages by dropping the generalized Walsh coefficients with high spatial frequencies. Goldberg (1990a) theorizes that a high-cardinality GA performs a reduction to smaller alphabets; this section gives an explanation of when this can occur, what it means in terms of generalized Walsh coefficients, and how to take advantage of it when analyzing a fitness function.

Example Consider the following test function:

Definition 4 (Test Function 1)

$$f(x) = \begin{cases} x & \text{if } x < 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{3.19}$$

This function has no continuous derivatives, and is discontinuous itself. Thus, the generalized Walsh coefficients w_j fall as j^{-1}. See Figure 3.1 for comparisons of the

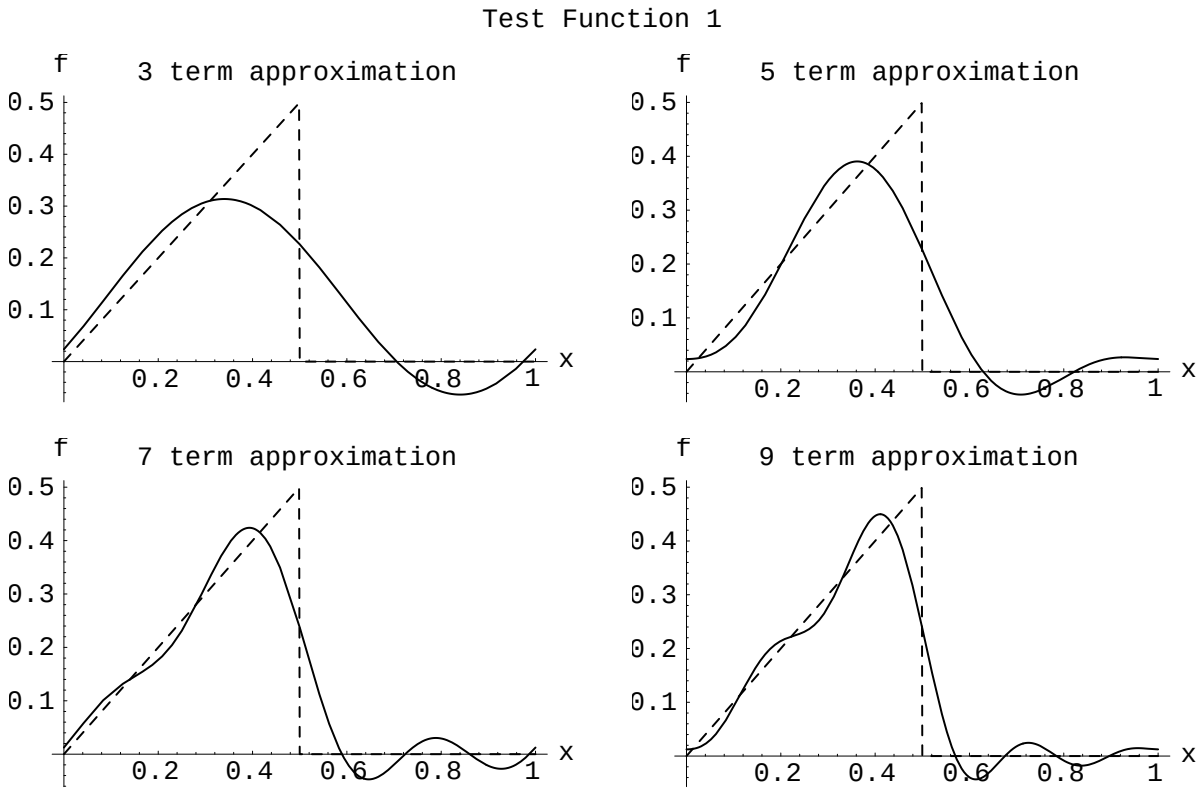

Figure 3.1: Test Function 1. Jump discontinuity slows convergence.

function with the approximations. Notice that since f is real-valued, the generalized Walsh coefficients w_j and w_{-j} are complex conjugates; and to get a real-valued approximation, if we keep w_j in our sum, we must also keep w_{-j}.

Example Consider the function:

Definition 5 (Test Function 2)

$$f(x) = x^2(1-x)^2(\frac{1}{2} - x)^3. \tag{3.20}$$

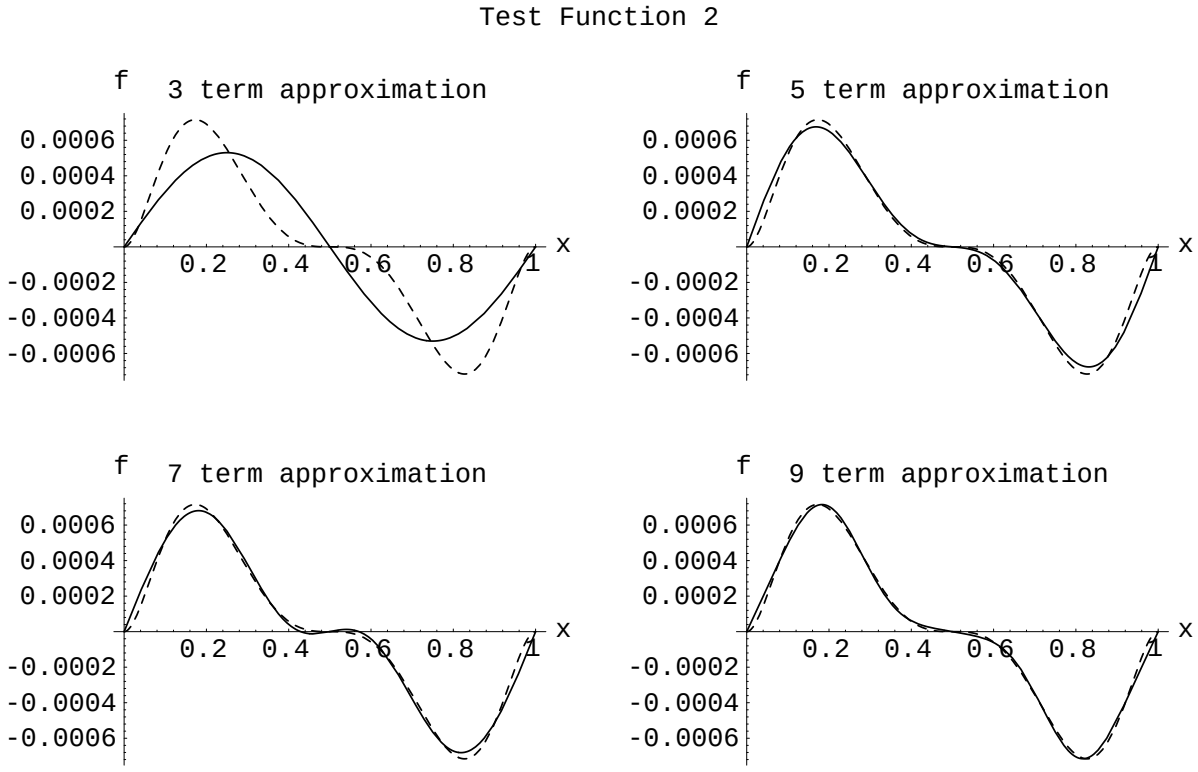

Figure 3.2: Test Function 2. Smoother functions converge faster.

Since $\partial^2 f/\partial x^2$ equals $1/4$ at $x = 0$ and $-1/4$ at $x = 1$, this function has a discontinuous second derivative. Therefore, the generalized Walsh coefficients fall as j^{-3}. See Figure 3.2 for comparisons of this function with the approximations. Notice how much faster the approximations converge to this function than the previous one. If a k-term generalized Walsh approximation is satisfactory, then the function can be treated as if it were a function over a k-ary alphabet. Not only does this make taking schema averaging simpler, but it also makes determining deception much easier.

Example Consider a function $f((x_1, x_2))$ where x_1 is a real variable in [0,1] and x_2 is a character from a ternary alphabet. Let $w_{(j_1,j_2)}$ be the generalized Walsh transform of f. Let $w'_{(j_1,j_2)}$ be the (5,2)-ary approximation of w. Then we have the following relations between w' and w:

$$\begin{aligned} w'_{(0,j_2)} &= 5^{1/2} w_{(0,j_2)}; \\ w'_{(1,j_2)} &= 5^{1/2} w_{(1,j_2)}; \\ w'_{(2,j_2)} &= 5^{1/2} w_{(2,j_2)}; \\ w'_{(3,j_2)} &= 5^{1/2} w_{(-2,j_2)}; \\ w'_{(4,j_2)} &= 5^{1/2} w_{(-1,j_2)}. \end{aligned} \tag{3.21}$$

The reason this works is that k-ary generalized Walsh functions are periodic with period k, and therefore the Walsh coefficients with indices $-m$ correspond to Walsh coefficients with indices $k - m$. The mapping used above keeps the 5×2 Walsh coefficients with the lowest spatial frequency and fixes the normalization.

Similarly, to reduce a function over (100,2)-ary strings to a function over a (7,2)-ary strings, one would use the following mapping:

$$\begin{aligned} w'_{(0,j_2)} &= \frac{7^{1/2}}{100^{1/2}} w_{(0,j_2)}; \\ w'_{(1,j_2)} &= \frac{7^{1/2}}{100^{1/2}} w_{(1,j_2)}; \\ w'_{(2,j_2)} &= \frac{7^{1/2}}{100^{1/2}} w_{(2,j_2)}; \\ w'_{(3,j_2)} &= \frac{7^{1/2}}{100^{1/2}} w_{(3,j_2)}; \end{aligned}$$

$$
\begin{aligned}
w'_{(4,j_2)} &= \frac{7^{1/2}}{100^{1/2}} w_{(97,j_2)}; \\
w'_{(5,j_2)} &= \frac{7^{1/2}}{100^{1/2}} w_{(98,j_2)}; \\
w'_{(6,j_2)} &= \frac{7^{1/2}}{100^{1/2}} w_{(99,j_2)}.
\end{aligned} \tag{3.22}
$$

Again, one simply takes the 7×2 Walsh coefficients with the lowest spatial frequency and fixes the normalization.

Let us do one final example. Consider again the Test Function 2 (3.20) we used earlier. The Walsh coefficients are given by the following:

$$
\begin{aligned}
w_0 &= 0; \\
w_1 &= -0.000265108\imath; \\
w_{-1} &= 0.000265108\imath; \\
w_2 &= -0.000124861\imath; \\
w_{-2} &= 0.000124861\imath; \\
w_3 &= -0.0000175818\imath; \\
w_{-3} &= 0.0000175818\imath; \\
\vdots &\quad \vdots.
\end{aligned} \tag{3.23}
$$

The 7-ary approximation to Test Function 2 (3.20) would have Walsh coefficients w', which are the following:

$$
w'_0 = 7^{1/2} w_0 = 0;
$$

$$
\begin{aligned}
w'_1 &= 7^{1/2} w_1 = -0.000701409\imath; \\
w'_2 &= 7^{1/2} w_2 = -0.000330351\imath; \\
w'_3 &= 7^{1/2} w_3 = 0.0000465171\imath; \\
w'_4 &= 7^{1/2} w_{-3} = -0.0000465171\imath; \\
w'_5 &= 7^{1/2} w_{-2} = 0.000330351\imath; \\
w'_6 &= 7^{1/2} w_{-1} = 0.00070149\imath. \qquad (3.24)
\end{aligned}
$$

Since a genetic algorithms population is always finite in practice, it is impossible to have individuals distributed over the entire real line from $-\infty$ to ∞ uniformly. In the initial population, one must distribute the individuals according to some probability distribution with a finite integral. For every probability distribution $P(x)$ with a finite integral, one can take a random variable uniformly distributed between 0 and 1 and transform it into $P(x)$ with an appropriate change of variable: $x = g(r)$. For instance, if r is a random variable uniformly distributed between 0 and 1, then $x = \frac{1}{r} - 1$ is a random variable distributed between 0 and ∞, and $y = \tan(\pi(r - 1/2))$ is a random variable distributed between $-\infty$ and ∞.

Example Consider an initial population of ∞-ary strings x of length 1 distributed with the normalized probability distribution $P(x)$. Let the fitness function be $f(x)$. Rather than considering $f(x)$, it is easier to consider $f(g(x'))$, where the function g

is defined such that x' is distributed evenly between 0 and 1:

$$\begin{aligned} g(\int_{-\infty}^{x} P(q)dq) &= x; \\ \int_{-\infty}^{x} P(q)dq &= g^{-1}(x). \end{aligned} \tag{3.25}$$

Here, $g^{-1}(x)$ refers to the inverse of the function g. Under the change of variable $x' = g^{-1}(x)$, the initial population of strings x' are uniformly distributed in the interval [0,1].

This method of transforming an unbounded real variable into a real variable uniformly distributed in [0,1] allows us to analyze GAs over unbounded real variables without introducing arbitrary cutoffs or generalizing nonuniform Walsh schema transforms (Bridges & Goldberg, 1989).

3.5 Generalizing the Fast Walsh Transform

The Fast Walsh Transform (Goldberg, 1989b) can be generalized in a similar manner to the Walsh transform. In the ordinary Fast Walsh Transform, one places the function values $f(x)$ on a binary tree; the position of $f(x)$ on the tree corresponds to the representation of x. For example, $f((0,1,0,0))$ would be found by starting at the root, taking the left branch, then the right, then a left, and finally another left.

One descends down and process the tree level by level. At each level, one applies the algorithm described below to each node in that level before descending down to

the next level. The algorithm is the following: take the left subtree l of the node, add it to the right subtree of the node, and call the result l'; take the left subtree l of the node, subtract the right subtree r and call it r'. Now replace the left subtree with l' and the right subtree with r'. To put it more briefly: $(l, r) \rightarrow (l + r, l - r)$ where the tree-wise addition and subtraction means to add like components of l and r.

To generalize this to functions over k-ary strings, form a k-ary tree and descend level by level. Instead of applying the rule $(l, r) \rightarrow (l+r, l-r)$, we apply the following rule:

$$\begin{aligned}(c_1, c_2, \ldots, c_k) \quad \rightarrow \\ & (c_1 + c_2 + \ldots + c_k, \\ & c_1 + e^{\frac{2\pi\imath}{k}} c_2 + e^{\frac{4\pi\imath}{k}} c_3 + \ldots + e^{\frac{2(k-1)\pi\imath}{k}} c_k, \\ & \cdots).\end{aligned} \tag{3.26}$$

The basic idea is the same as with the ordinary Fast Walsh Transform, but instead of simply adding and subtracting the children at each node, one performs a Fourier transform upon the children. That is, $c \rightarrow FT[c]$. Note that if we have a large alphabet, it is worthwhile to perform this Fourier transform using a Fast Fourier transform.

Note that if we have a $\vec{k}$-ary alphabet, then we can still perform the Fast Walsh Transform. The root node of our tree would have k_1 children; all the nodes at the

next lower level would have k_2 children, etc. We still go down the tree level by level and apply the Fourier transform upon the children of each node.

Example Figure 3.3 shows a generalized Fast Walsh Transform for a function over (2,3)-ary strings.

Mathematica 2.0 programs that compute k-ary and $\vec{k}$-ary Fast Walsh Transforms and their inverses are given in Appendix A.

This chapter has shown that the Walsh functions and transforms can be generalized to non-binary strings. The next chapter shows that the generalized Walsh functions and transforms can be used to compute schema averages.

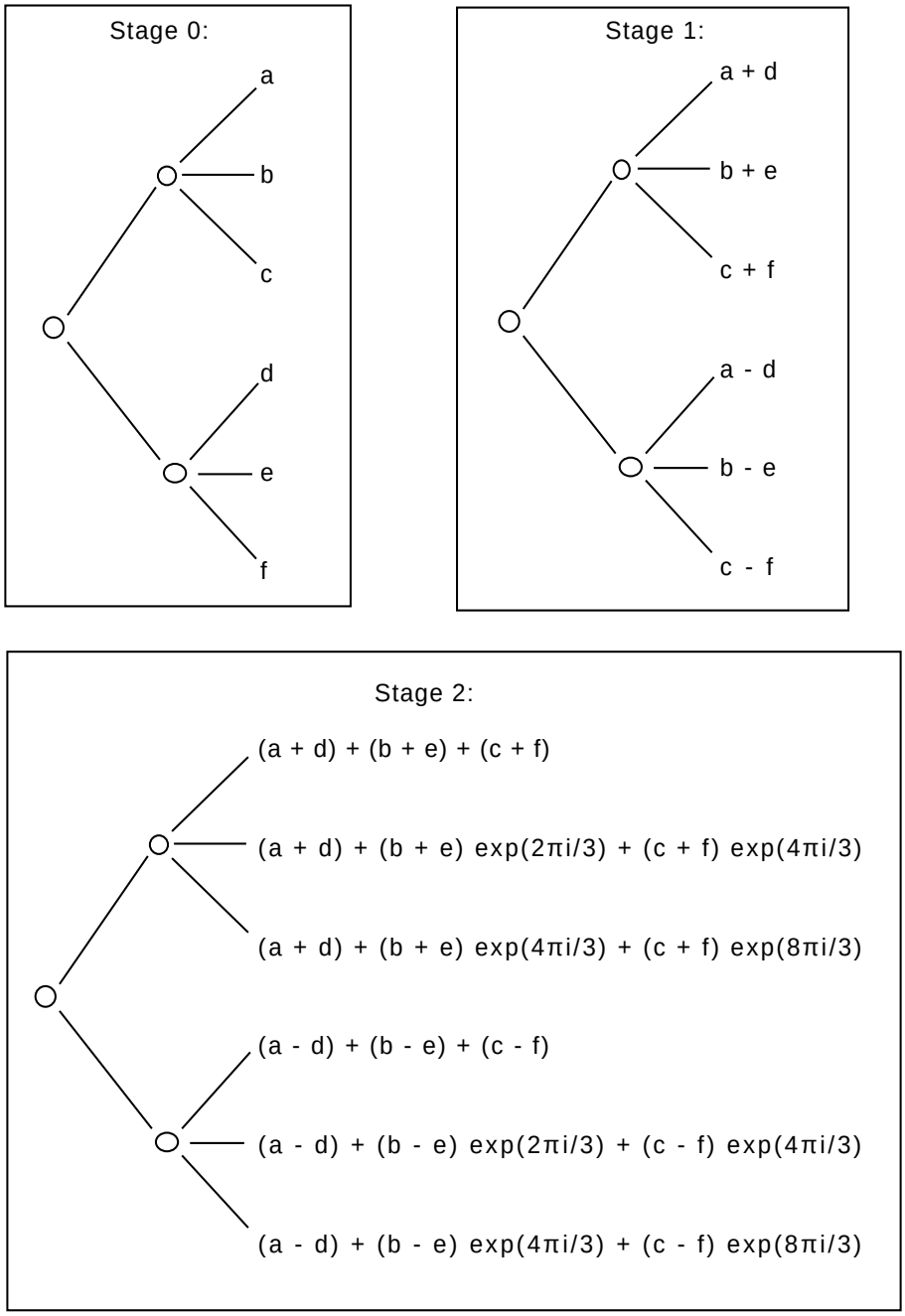

Figure 3.3: Generalized Fast Walsh Transform for (2,3)-ary strings.

Chapter 4

Using Generalized Walsh Coefficients to Determine Schema Averages

Determining schema averages is at the heart for the reason of using these transform methods. In Chapter 3, this thesis stated that a sum of f over m characters of a string of length n turned into a sum of generalized Walsh coefficients over $n - m$ characters. For this reason, the average of a function over a schema with m positions fixed turns into a sum over m characters of the Walsh coefficients.

Example Consider strings of length 3 taken from a (2,4,3)-ary alphabet. Some examples of schema averages are as follows:

$$
\begin{aligned}
f((*,*,*)) &= \frac{1}{4}w_{(0,0,0)};\\
f((0,*,*)) &= \frac{1}{4}(w_{(0,0,0)} + w_{(1,0,0)});\\
f((1,*,*)) &= \frac{1}{4}(w_{(0,0,0)} - w_{(1,0,0)});\\
f((*,0,*)) &= \frac{1}{4}(w_{(0,0,0)} + w_{(0,0,1)} + w_{(1,0,0)} + w_{(1,0,1)});\\
f((*,1,*)) &= \frac{1}{4}(w_{(0,0,0)} + \imath w_{(0,0,1)} - w_{(1,0,0)} - \imath w_{(1,0,1)});\\
f((*,2,*)) &= \frac{1}{4}(w_{(0,0,0)} - w_{(0,0,1)} + w_{(1,0,0)} - w_{(1,0,1)});\\
f((*,3,*)) &= \frac{1}{4}(w_{(0,0,0)} - \imath w_{(0,0,1)} - w_{(1,0,0)} + \imath w_{(1,0,1)});\\
f((*,*,0)) &= \frac{1}{4}(w_{(0,0,0)} + w_{(0,0,1)} + w_{(0,0,2)});\\
f((*,*,1)) &= \frac{1}{4}(w_{(0,0,0)} + e^{2\pi\imath/3}w_{(0,0,1)} + e^{4\pi\imath/3}w_{(0,0,2)});\\
f((0,0,*)) &= \frac{1}{4}(w_{(0,0,0)} + w_{(0,1,0)} + w_{(0,2,0)} + w_{(0,3,0)}\\
&\quad + w_{(1,0,0)} + w_{(1,1,0)} + w_{(1,2,0)} + w_{(1,3,0)});\\
f((0,1,1)) &= \frac{1}{4}(w_{(0,0,0)} + e^{2\pi\imath/3}w_{(0,0,1)} + e^{4\pi\imath/3}w_{(0,0,2)}\\
&\quad + \imath w_{(0,1,0)} + \imath e^{2\pi\imath/3}w_{(0,1,1)} + \imath e^{4\pi\imath/3}w_{(0,1,2)}\\
&\quad - w_{(0,2,0)} - e^{2\pi\imath/3}w_{(0,2,1)} - e^{4\pi\imath/3}w_{(0,2,2)}\\
&\quad - \imath w_{(0,3,0)} - \imath e^{2\pi\imath/3}w_{(0,3,1)} - \imath e^{4\pi\imath/3}w_{(0,3,2)}\\
&\quad + w_{(1,0,0)} + e^{2\pi\imath/3}w_{(1,0,1)} + e^{4\pi\imath/3}w_{(1,0,2)}
\end{aligned}
$$

$$+\imath w_{(1,1,0)} + \imath e^{2\pi\imath/3} w_{(1,1,1)} + \imath e^{4\pi\imath/3} w_{(1,1,2)}$$

$$-w_{(1,2,0)} - e^{2\pi\imath/3} w_{(1,2,1)} - e^{4\pi\imath/3} w_{(1,2,2)}$$

$$-\imath w_{(1,3,0)} - \imath e^{2\pi\imath/3} w_{(1,3,1)} - \imath e^{4\pi\imath/3} w_{(1,3,2)}). \tag{4.1}$$

Note that the difference between the sums for $f((0,*,*))$ and $f((1,*,*))$ is that the Walsh coefficients are subtracted rather than added. In general, when we have a fixed character p in the j-th position in the schema, the corresponding sum in the transform has a phase of $e^{2\pi p/k_j}$.

It is straightforward to make a general theorem from these observations.

Theorem 6 *Consider a schema with m fixed characters p_i at positions j_i. Then the average of f over that schema is*

$$\prod_q k_q^{-1/2} \sum_{l_1} \sum_{l_2} \cdots \sum_{l_m} \quad e^{2\pi\imath(l_1 p_1/k_{j_1} + l_2 p_2/k_{j_2} + \ldots + l_m p_m/k_{j_m})}$$

$$w_{(0,0,\ldots,0,l_1,0,\ldots,0,l_2,0,\ldots,0,l_m,0,\ldots)}. \tag{4.2}$$

Proof Recall that when we write f in terms of its generalized Walsh coefficients, the result has n sums, where n is the length of the strings. There is a sum for each of the n characters in the string of the argument. Now, when we take the average of f over a schema which has a $*$ in the j-th position, only the terms that have no phase change as we traverse the j-th dimension survive; this means that only the generalized Walsh coefficients with a 0 in the j-th position survive in the final result.

We can also define schemata in the space of real numbers, and take schema averages using generalized Walsh coefficients as before. These schemata are analogous to the slices used in Goldberg (1990a) rather than the schemata for real parameters defined in Wright (1991), and are a natural extension of schemata in finite alphabets.

Example Consider a function f over $(\infty, \infty, 3, 3, 3)$-ary strings. Let (0.67,*,*,*,1) refer to the set of strings whose first variable x_1 equals 0.67 and last variable x_5 equals 1. Then the average of the function f over that schema is

$$f((0.67, *, *, *, 1)) = 3^{-3/2} \sum_{j_1=-\infty}^{\infty} \sum_{j_5=0}^{2} w_{(j_1,0,0,0,j_5)} e^{2\pi\imath(0.67 j_1 + 1 j_5/3)}. \tag{4.3}$$

To get more familiar with using these generalized Walsh coefficients on real variables, let us do some more schema averages:

$$\begin{aligned} f((*,*,*,*,*)) &= 3^{-3/2} w_{(0,0,0,0,0)}; \\ f((*,*,*,*,2)) &= 3^{-3/2}(w_{(0,0,0,0,0)} + e^{2\pi\imath/3} w_{(0,0,0,0,1)} + e^{4\pi\imath/3} w_{(0,0,0,0,2)}); \\ f((0,*,*,*,*)) &= 3^{-3/2} \sum_{j_1=-\infty}^{\infty} w_{(j_1,0,0,0,0)}. \end{aligned} \tag{4.4}$$

This chapter has generalized the notion of schema used in analyzing genetic algorithms of binary strings and showed how to use generalized Walsh functions and transforms to compute schema averages. A method of comparing schema averages to determine deception is given in the next chapter.

Chapter 5

Using Generalized Hadamard Transforms to Detect Deception

5.1 Deception in Non-binary Strings

Hadamard transforms provide a convenient way of checking deceptive conditions in a systematic manner (Homaifar, Qi, Fost, 1991; Homaifar, Qi, 1990). In this chapter, the Hadamard transform will be generalized to non-binary strings.

Deception in functions over non-binary strings is qualitatively different than deception in functions over binary strings. In order to get full deception with binary strings, all schemas of order $n-1$ or less must point towards the complement of the global optimum. With non-binary strings, all that is required is that all schemas

of order $n-1$ or less point away from the global optimum. Mason uses this as his definition of deception (Mason, 1991).

Example Consider a function f over a ternary string of length 3 whose global optimum is at (2,2,2). Then some of the conditions necessary for deception are as follows:

$$\begin{aligned}
f((2,*,*)) &< f((0,*,*)) \text{ or } f((1,*,*)); \\
f((2,2,*)) &< f((0,0,*)) \text{ or } f((0,1,*)) \text{ or } f((1,0,*)) \text{ or } f((1,1,*)); \\
f((2,2,*)) &< f((1,2,*)) \text{ or } f((0,2,*)); \\
f((2,2,*)) &< f((2,0,*)) \text{ or } f((2,1,*)); \\
f((1,2,*)) &< f((1,0,*)) \text{ or } f((1,1,*)). \qquad (5.1)
\end{aligned}$$

Mutation for non-binary strings works differently than mutation for binary strings. Some mutation operators, such as creeping mutation, mutate characters to nearby characters. For example, using a 10-ary alphabet, the string (0,0,0) is much more likely to be perturbed to (1,0,0) than (9,0,0). Similarly, mutations on a real number might be implemented by adding noise with a gaussian distribution. If the fitness function is not too discontinuous, then these kinds of mutations can work as a sort of gradient descent to help convergence (Bledsoe, 1961). The analysis of how the distribution of perturbations affects convergence is beyond the scope of this thesis

and overlaps the theory of simulated annealing (Kirkpatrick, Gelatt, & Vecci, 1983; Szu & Hartley, 1987). Assuming that the algorithm uses localized perturbations, it is reasonable to define a fully deceptive function as a function whose order $n-1$ or less schemas point as far away from the global optima as possible. Now the deceptive conditions become more straightforward:

$$\begin{aligned}
f((0,*,*)) &> f((1,*,*)),\\
f((0,*,*)) &> f((2,*,*)),\\
f((0,0,*)) &> f((0,1,*)),\\
f((0,0,*)) &> f((0,2,*)),\\
f((0,0,*)) &> f((1,0,*)),\\
f((0,0,*)) &> f((1,1,*)),\\
f((0,0,*)) &> f((1,2,*)),\\
f((0,0,*)) &> f((2,0,*)),\\
f((0,0,*)) &> f((2,1,*)),\\
f((0,0,*)) &> f((2,2,*)),
\end{aligned} \tag{5.2}$$

and all permutations of the strings above.

5.2 Hadamard Transform

Consider the competition partition in which the fixed positions are at $j_1, j_2, \ldots, j_p$, where p is the order of the partition.

Definition 6 (Generalized Hadamard Transform) *Define the generalized Hadamard transform matrix H as:*

$$H = h_1 \otimes h_2 \otimes \ldots \otimes h_p, \tag{5.3}$$

where $\otimes$ is the tensor product:

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \otimes \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae & af & be & bf \\ ag & ah & bg & bh \\ ce & cf & de & df \\ cg & ch & dg & dh \end{pmatrix}, \tag{5.4}$$

and h_m is defined as follows:

$$h_m = \begin{pmatrix} 1 & 1 & 1 & \ldots & 1 \\ 1 & e^{2\pi\imath/k_{j_m}} & e^{4\pi\imath/k_{j_m}} & \ldots & e^{(k_{j_m}-1)2\pi\imath/k_{j_m}} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & e^{(k_{j_m}-1)2\pi\imath/k_m} & e^{(k_{j_m}-1)4\pi\imath/k_m} & \ldots & e^{(k_{j_m}-1)(k_{j_m}-1)2\pi\imath/k_{j_m}} \end{pmatrix}. \tag{5.5}$$

The generalized Hadamard transform will be used in the next section to compute the conditions for deception.

5.3 Detecting Deception

Now we make the important assumption that the global optimum is at $(k_1 - 1, k_2 - 1, \ldots, k_n - 1)$ and all schemas of order $n - 1$ or less must point to $(0, 0, \ldots, 0)$. This definition of deception requires that the lower-order schemata lead as far away from the global optimum as possible. This differs from Mason's definition of deception, which requires only that the lower-order schemata do not point to the global optimum. Using this new definition, we can now define the matrix M such that the deceptive conditions for that partition become $MW > 0$, where

$$M = \begin{pmatrix} \text{1st row of H-2nd row of H} \\ \text{1st row of H-3rd row of H} \\ \vdots \\ \text{1st row of H-last row of H} \end{pmatrix}, \tag{5.6}$$

and W is the vector of generalized Walsh coefficients used in the competition partition.

Example Let w be the Walsh coefficients for the fitness function f over (3,2,2)-ary strings. Consider the competition partition $(F, *, *)$, where F represents a fixed position and * represents a wildcard character. Then

$$\begin{aligned} W &= (w_{(0,0,0)}, w_{(1,0,0)}, w_{(2,0,0)})^T, \\ H &= \begin{pmatrix} 1 & 1 & 1 \\ 1 & e^{2\pi\imath/3} & e^{4\pi\imath/3} \\ 1 & e^{4\pi\imath/3} & e^{8\pi\imath/3} \end{pmatrix}, \end{aligned}$$

$$M = \begin{pmatrix} 0 & 1-e^{2\pi\imath/3} & 1-e^{4\pi\imath/3} \\ 0 & 1-e^{4\pi\imath/3} & 1-e^{8\pi\imath/3} \end{pmatrix}. \quad (5.7)$$

The condition that $MW > 0$ becomes the following:

$$\begin{aligned} (1-e^{2\pi\imath/3})w_{(1,0,0)} + (1-e^{4\pi\imath/3})w_{(2,0,0)} &> 0; \\ (1-e^{4\pi\imath/3})w_{(1,0,0)} + (1-e^{8\pi\imath/3})w_{(2,0,0)} &> 0. \end{aligned} \quad (5.8)$$

When we substitute function values for the Walsh coefficients in the above expression, we get

$$\begin{aligned} f((0,*,*)) &> f((1,*,*)), \\ f((0,*,*)) &> f((2,*,*)), \end{aligned} \quad (5.9)$$

which are indeed the deceptive conditions corresponding to that partition.

Example Let us repeat the above calculations for the competition partition (F, D, F):

$$W = (w_{(0,0,0)}, w_{(0,0,1)}, w_{(1,0,0)}, w_{(1,0,1)}, w_{(2,0,0)}, w_{(2,0,1)})^T,$$

$$H = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & e^{2\pi\imath/3} & e^{2\pi\imath/3} & e^{4\pi\imath/3} & e^{4\pi\imath/3} \\ 1 & -1 & e^{2\pi\imath/3} & -e^{2\pi\imath/3} & e^{4\pi\imath/3} & -e^{4\pi\imath/3} \\ 1 & 1 & e^{4\pi\imath/3} & e^{4\pi\imath/3} & e^{8\pi\imath/3} & e^{8\pi\imath/3} \\ 1 & -1 & e^{4\pi\imath/3} & -e^{4\pi\imath/3} & e^{8\pi\imath/3} & -e^{8\pi\imath/3} \end{pmatrix},$$

$$M = \begin{pmatrix} 0 & 2 & 0 & 2 & 0 & 2 \\ 0 & 0 & 1-A & 1-A & 1-B & 1-B \\ 0 & 2 & 1-A & 1+A & 1-B & 1+B \\ 0 & 0 & 1-B & 1-B & 1-A & 1-A \\ 0 & 2 & 1-B & 1+A & 1-A & 1+B \end{pmatrix}, \tag{5.10}$$

where $A = e^{2\pi\imath/3}$ and $B = e^{-2\pi\imath/3}$. $MW > 0$ gives

$$\begin{aligned} 2w_{(0,0,1)} + 2w_{(1,0,1)} + 2w_{(2,0,1)} &> 0, \\ (1-A)w_{(1,0,0)} + (1-A)w_{(1,0,1)} + (1-B)w_{(2,0,0)} + (1-B)w_{(2,0,1)} &> 0, \\ 2w_{(0,0,1)} + (1-A)w_{(1,0,0)} + (1+A)w_{(1,0,1)} + (1-B)w_{(2,0,0)} + & \\ (1+B)w_{(2,0,1)} &> 0, \\ (1-B)w_{(1,0,0)} + (1-B)w_{(1,0,1)} + (1-A)w_{(2,0,0)} + (1-A)w_{(2,0,1)} &> 0, \\ 2w_{(0,0,1)} + (1-B)w_{(1,0,0)} + (1+B)w_{(1,0,1)} + (1-A)w_{(2,0,0)} + & \\ (1+A)w_{(2,0,1)} &> 0. \end{aligned} \tag{5.11}$$

Translated into function values, this gives the following set of inequalities:

$$\begin{aligned} f((0,*,0)) &> f((0,*,1)), \\ f((0,*,0)) &> f((1,*,0)), \\ f((0,*,0)) &> f((1,*,1)), \\ f((0,*,0)) &> f((2,*,0)), \end{aligned}$$

$$f((0,*,0)) \quad > \quad f((2,*,1)), \tag{5.12}$$

which are the deceptive conditions corresponding to the competition partition (F, D, F).

For a function to be deceptive at order p requires that all order p schemas lead as far away to the global optimum as possible. Since there are $\binom{n}{p}$ ways of choosing p fixed positions among the n characters, there are that many competition partitions at that order. In the above example, the function is deceptive at order 1 if it is deceptive in the partitions (F,D,D), (D,F,D), and (D,D,F). For full deception, the function must be deceptive at all orders between 1 and $n-1$ inclusive. Mathematica routines for computing all of the above are included in Appendix A.

This chapter has defined the deceptive conditions for functions over non-binary alphabets. As mentioned in Chapter 2, deception is only one of the many reasons why a genetic algorithm may fail. The next chapter focusses on another mode of failure: the variances of schema fitness.

Chapter 6

Variance of Fitness

Although the schema theorem predicts the expected growth rate of a schema in a population, the finite population size introduces a sampling error that can lead to large deviations from expectations and can affect GA convergence (Goldberg & Rudnick, 1991; Goldberg, Deb, & Clark, 1991; Rudnick & Goldberg, 1991; Schaffer, Eschelman, & Offutt, 1991).

The full theory of how this sampling error affects GA behavior is beyond the scope of this thesis. Only the methods for calculating the signal and noise in a competition partition (Rudnick & Goldberg, 1991) will be extended to non-binary alphabets.

Definition 7 (Index Set) *The index set $G(h)$ of a schema h is the set of indices of the generalized Walsh coefficients used to express the function average over the schema h.*

Example In the space of strings of length 3 taken from a ternary alphabet,

$$\begin{aligned} G((*,*,*)) &= \{(0,0,0)\}, \\ G((0,*,*)) &= G((1,*,*)) = G((2,*,*)) = \{(0,0,0),(1,0,0),(2,0,0)\} = (*,0,0), \\ G((0,0,*)) &= G((1,2,*)) = (*,*,0). \end{aligned} \tag{6.1}$$

This notation helps to make the definitions of signal and noise clearer and more compact.

Definition 8 (Collateral Noise) *The collateral noise σ_h for a schema h is defined by the following:*

$$\begin{aligned} \sigma_h^2 = \mathrm{var}(f(h)) &= \frac{1}{|h|}\sum \left| f(x) - \frac{1}{|h|}\sum f(y) \right|^2; \\ &= \frac{1}{|h|}\sum |f(x)|^2 - \left| \frac{1}{|h|}\sum f(x) \right|^2; \\ &= \left\langle f(x)^2 \right\rangle_h - \langle f(x) \rangle_h^2. \end{aligned} \tag{6.2}$$

$\langle f(x) \rangle_h$ indicates the average of $f(x)$ where x ranges over schema h, $\langle f^2(x) \rangle_h$ indicates the average of $f^2(x)$ over h, and $|h|$ is the number of individuals represented by schema h.

This definition differs from Rudnick and Goldberg (1991), which defines the collateral noise as σ_h^2.

Definition 9 (Partition Signal) *The signal $S(J)$ of a competition partition J is*

defined as

$$S^2(J) = \left\langle |f(h)|^2 \right\rangle_J - |\langle f(h) \rangle_J|^2. \tag{6.3}$$

$\langle f(h) \rangle_J$ denotes the average of schema averages $f(h)$ where h runs over J. $\langle |f(h)|^2 \rangle_J$ denotes the average of the absolute value squares of schema averages in the competition partition J.

The first term in Equation 6.3 can be written as

$$\left\langle |f(h)|^2 \right\rangle_J = \frac{1}{|J|} \sum_{h \epsilon J} |f(h)|^2, \tag{6.4}$$

where $|J|$ is the number of schemata in J. Substituting the expression for $f(h)$ in terms of generalized Walsh coefficients gives

$$\left\langle |f(h)|^2 \right\rangle_J = \frac{1}{|J|} \prod_m k_m^{-1} \sum_{h \epsilon J} \left| \sum_{\vec{j} \epsilon G(h)} w_{\vec{j}} \Psi_{\vec{j}}^{(\vec{k})}(h) \right|^2. \tag{6.5}$$

Expanding the quadratic yields

$$\left\langle |f(h)|^2 \right\rangle_J = \frac{1}{|J|} \prod_m k_m^{-1} \sum_{h \epsilon J} \sum_{\vec{j},\vec{l} \epsilon G(h)} w_{\vec{j}} \overline{w}_{\vec{l}} \Psi_{\vec{j} \ominus \vec{l}}^{(\vec{k})}(h), \tag{6.6}$$

where $j \ominus l = (j_1 - l_1 \bmod k_1, j_2 - l_2 \bmod k_2, \ldots, j_n - l_n \bmod k_n)$. Due to the orthogonality of the generalized Walsh functions, this reduces to

$$\left\langle |f(h)|^2 \right\rangle_J = \frac{1}{|J|} \prod_m k_m^{-1} \sum_{\vec{j} \epsilon G(h)} |w_{\vec{j}}|^2. \tag{6.7}$$

The second term in Equation 6.3 reduces to the following product:

$$|\langle f(h) \rangle_J|^2 = \prod_m k_m^{-1} |w_0|^2. \tag{6.8}$$

Thus, Equation 6.3 simplifies to:

$$\begin{aligned} S^2(J) &= \sum_{\vec{\jmath}\epsilon G(J)} (|w_{\vec{\jmath}}|^2 - |w_0|^2); \\ &= \sum_{\vec{\jmath}\epsilon G(J)-\{0\}} |w_{\vec{\jmath}}|^2. \end{aligned} \tag{6.9}$$

Example Consider (3,3,3)-ary strings and the competition partition $J = (F, F, *)$. The square of the partition signal of J is

$$\begin{aligned} S^2(J) &= |w_{(0,1,0)}|^2 + |w_{(0,2,0)}|^2 \\ &\quad + |w_{(1,0,0)}|^2 + |w_{(1,1,0)}|^2 + |w_{(1,2,0)}|^2 \\ &\quad + |w_{(2,0,0)}|^2 + |w_{(2,1,0)}|^2 + |w_{(2,2,0)}|^2. \end{aligned} \tag{6.10}$$

Definition 10 (Partition RMS Noise) *The root-mean-squared noise $C(J)$ of a competition partition J is defined as the root-mean-square of the collateral noises for each of the schema h in the partition.*

$$C^2(J) = \left\langle \sigma_h^2 \right\rangle_J \tag{6.11}$$

In Chapter 3, there were some examples (3.13,3.14) of computing the $\langle f(x)^2 \rangle_h$ term in the expression for σ_h^2 (6.2). These examples can be generalized to give the following result. For the complete derivations for binary strings, see Goldberg & Rudnick (1991) and Rudnick & Goldberg (1991).

$$\sigma_h^2 = \prod_m k_m^{-1} \sum_{(\vec{\jmath},\vec{l})\epsilon G^2_{\ominus}(h) - G^2(h)} \overline{w}_{\vec{\jmath}} w_{\vec{l}} \Psi^{(\vec{k})}_{\vec{l}\ominus\vec{\jmath}}(h) \tag{6.12}$$

$G^2(h) = G(h) \times G(h)$, $G^2_{\ominus}(h) = \{(\vec{j},\vec{l}) : \vec{j} \ominus \vec{l} \epsilon G(h)\}$, and $\vec{j} \ominus \vec{l} = (j_1 - l_2 \bmod k_1, j_2 - l_2 \bmod k_2, \ldots, j_n - l_n \bmod k_n)$. As before, the off-diagonal terms vanish, leaving

$$C^2(J) = \sum_{\vec{j} \epsilon \tilde{G}(J)} |w_{\vec{j}}|^2 \tag{6.13}$$

where $\tilde{G}(J)$ is the complement of the set $G(J)$.

Example Consider again the above example using (3,3,3)-ary strings and the competition partition $J = (F, F, *)$. The square of the partition RMS noise of J is

$$\begin{aligned} C^2(J) &= |w_{(0,0,1)}|^2 + |w_{(0,0,2)}|^2 + |w_{(0,1,1)}|^2 + |w_{(0,1,2)}|^2 \\ &\quad + |w_{(0,2,1)}|^2 + |w_{(0,2,2)}|^2 + |w_{(1,0,1)}|^2 + |w_{(1,0,2)}|^2 \\ &\quad + |w_{(1,1,1)}|^2 + |w_{(1,1,2)}|^2 + |w_{(1,2,1)}|^2 + |w_{(1,2,2)}|^2 \\ &\quad + |w_{(2,0,1)}|^2 + |w_{(2,0,2)}|^2 + |w_{(2,1,1)}|^2 + |w_{(2,1,2)}|^2 \\ &\quad + |w_{(2,2,1)}|^2 + |w_{(2,2,2)}|^2. \end{aligned} \tag{6.14}$$

For large alphabets and fairly smooth fitness functions, we can use the alphabet-size reduction described earlier to get a good approximation to the variances with relatively little effort. For instance, the variance of Test Function 2 is

$$\begin{aligned} &\int_0^1 (x^2(1-x)^2(1/2-x)^3)^2 dx \\ &- \left(\int_0^1 (x^2(1-x)^2(1/2-x)^3) dx \right)^2 \quad = \quad 1.734 \times 10^{-7}. \end{aligned} \tag{6.15}$$

We can approximate the same calculation using the 7-ary reduction.

$$\frac{1}{7}(|w_1|^2 + |w_2|^2 + |w_3|^2 + |w_4|^2 + |w_5|^2 + |w_6|^2) = 1.723 \times 10^{-7}. \tag{6.16}$$

This chapter has shown that the signal and noise calculations for functions over binary strings can be generalized to non-binary strings as well. The signal-to-noise calculations and the Hadamard transforms are tools which examine two independent modes by which a genetic algorithm can fail to converge to the global optimum.

Chapter 7

Conclusion

This thesis has shown how the Walsh functions and many of the techniques that use it can be generalized to non-binary alphabets in a natural and straightforward way. This conclusion focuses on possible topics for future research that use generalized Walsh functions.

Directions for future research using generalized Walsh functions and transforms include generalizing more of the theory for binary strings, such as the nonuniform Walsh-schema transform (Bridges & Goldberg, 1991), creating deceptive problems (Liepins & Vose, 1990b; Liepins & Vose, 1991; Whitley, 1991a), operator-adjusted Walsh coefficients (Goldberg, 1989c), and the sufficient conditions for deception (Deb, Goldberg & 1992).

Another possibility is a probabilistic approach towards deception. Checking for

deception requires evaluating the fitness function over every single point in the entire space, and for this reason, it has never been done for anything but tractable problems or problems that were handmade to be difficult. Given the empirical evidence that smoother functions are easier to optimize, it seems possible that the probability that a function is deceptive depends on the asymptotic behavior of the generalized Walsh coefficients. It would be useful to have either analytical results or a table of probabilities that relate how quickly the coefficients decay, the length of the string, the cardinality, and the order of static deception. This approach might give a way of determining which representation is the likeliest to work for a given problem, and whether a GA is likely to solve the problem or not, using only knowledge about the smoothness of the fitness function.

In the theory of nonlinear systems, there are two methods for finding solutions. The first is to make a linear approximation; the second is to use some symmetry of the system to reduce the dimensionality of the problem. The current theory for genetic algorithms is largely based on the first method; it makes predictions based on a linearized model of the system at generation zero. As Grefenstette (1991) pointed out, the validity of this linear approximation as time progresses is open to question. Methods of analysis based on a symmetry of the system would be valid for any length of time. N. Packard (personal communication, 1991) and this author have speculated that a renormalization group technique such as that used in analyzing lattice processes

in physics could also be used for analyzing GAs. The genetic algorithm, not including the user-defined fitness function, appears to have no fundamental length scale other than the string length, which corresponds to the lattice size in physics. Also, the way that low-order schemata combine to form higher-order schemata is reminiscent of a phase transition. These two facts, along with some numerical experiments, suggest that scaling does occur in some situations and a renormalization group approach would be promising. A renormalization group approach would involve treating blocks of characters in the string as a single character of higher cardinality, and then asking what would a genetic algorithm with this new representation do. This thesis, which provides a framework for a theory of genetic algorithms over non-binary strings, is a step in that direction.

Appendix A

Mathematica 2.0 Programs

A.1 Generalized Walsh Functions Over k-ary Strings

```
k is the size of the alphabet

n is the maximum length of the strings

j is the index (in integer form) of the generalized Walsh function

x is the argument (in integer form) of the generalized Walsh function


Psi[k_Integer,n_Integer,j_Integer,x_Integer]:=

        E^(2 Pi I IntegerDigits[j,k,n].IntegerDigits[x,k,n] / k)/

        Sqrt[k^n]
```

Example From the ternary alphabet and strings of length two example in the text above, we evaluate the fourth Walsh function at position two.

```
In[3]:= Psi[3,2,4,2]


          (-2 I)/3 Pi
         E
Out[3]= ------------
              3
```

A.2 Generalized Walsh Functions Over More Arbitrary Strings

```
k is a list of the sizes of the alphabet used in each character
j is the index (in vector form) of the generalized Walsh function
x is the argument (in vector form) of the generalized Walsh function


Psi[k_List,j_List,x_List]:=
        E^(2 Pi I Plus @@ MapThread[#1 #2/#3 &, {j, x, k}])/
        Sqrt[Times@@k]
```

Example Consider the set of strings with length two. The first character is chosen from a binary alphabet, and the second from a ternary alphabet. Evaluate the (0,2) Walsh function at position (1,2).

```
In[6]:= Psi[{2,3},{0,2},{1,2}]

          (2 I)/3 Pi
         E
Out[6]= -----------
           Sqrt[6]
```

A.3 Generalized Fast Walsh Transform Over k-ary Strings

```
k is the size of the alphabet

l is the list of function values whose length is a power of k


GFWT[k_Integer,l_List]:=

        Flatten[
```

```
        Nest[ Function[z,Table[E^-(2 Pi I j x/k),{j,0,k-1},{x,0,k-1}].z] /@
                Partition[#,k]&,l,Round[Log[k,Length[l]]]]]/
                        Sqrt[k^Round[Log[k,Length[l]]]]

InverseGFWT[k_Integer,l_List]:=
        Flatten[
        Nest[ Function[z,Table[E^(2 Pi I j x/k),{j,0,k-1},{x,0,k-1}].z] /@
                Partition[#,k]&,l,Round[Log[k,Length[l]]]]]/
                        Sqrt[k^Round[Log[k,Length[l]]]]
```

Example Create a list of 9 random numbers, find the transform for a ternary alphabet, and then untransform the data to recover the original 9 numbers.

```
In[3]:= Table[Random[],{9}]

Out[3]= {0.177361, 0.503785, 0.387824, 0.615398, 0.254133, 0.488114,

>    0.00850412, 0.539387, 0.788743}

In[4]:= GFWT[3,%] //N
```

```
Out[4]= {1.25442, -0.226577 + 0.106052 I, -0.226577 - 0.106052 I,

>    -0.0927234 - 0.00606561 I, -0.0247775 - 0.362999 I,

>    -0.0170897 - 0.156521 I, -0.0927234 + 0.00606561 I,

>    -0.0170897 + 0.156521 I, -0.0247775 + 0.362999 I}

In[5]:= InverseGFWT[3,%] //N //Chop

Out[5]= {0.177361, 0.503785, 0.387824, 0.615398, 0.254133, 0.488114,

>    0.00850412, 0.539387, 0.788743}
```

A.4 Generalized Fast Walsh Transform Over More Arbitrary Strings

```
k is a list of the sizes of the alphabets in the string

l is the list of function values whose length is a the product of

     the sizes of the alphabets
```

```
GFWT[k_List,l_List]:=
        Flatten[Fold[
        Function[z, Table[E^-(2 Pi I i j/#2),{i,0,#2-1},{j,0,#2-1}] . z] /@
                Partition[#1,#2] &, 1, Reverse[k]]]/
                        Sqrt[Times@@k]

InverseGFWT[k_List,l_List]:=
        Flatten[Fold[
        Function[z, Table[E^(2 Pi I i j/#2),{i,0,#2-1},{j,0,#2-1}] . z] /@
                Partition[#1,#2] &, 1, Reverse[k]]]/
                        Sqrt[Times@@k]
```

Example Create a list of 6 random numbers and transform it over strings of length 2 whose first character is chosen from a binary alphabet and whose second character is chosen from a ternary alphabet.

```
In[3]:= Table[Random[],{6}]
```

```
Out[3]= {0.222531, 0.656238, 0.587434, 0.827656, 0.34676, 0.514834}

In[4]:= GFWT[{2,3},%] //N

Out[4]= {1.28821, -0.000998969 + 0.0350975 I, -0.000998969 - 0.0350975 I,

>    -0.0910589, -0.325032 - 0.0837492 I, -0.325032 + 0.0837492 I}

In[5]:= InverseGFWT[{2,3},%] //N //Chop

Out[5]= {0.222531, 0.656238, 0.587434, 0.827656, 0.34676, 0.514834}
```

A.5 Converting Schema Averages to Walsh Coefficients

```
k is a list containing the sizes of the alphabets

schema is the schema which we want to evaluate the fitness of.

D is the don't-care symbol

w is the name of the generalized Walsh coefficients


SchemaToWalsh[k_List,schema_List,w_]:=
```

```
Sum @@
Prepend[
Delete[MapIndexed[{j[#2[[1]]],0,#1-1}&,k],Position[schema,D]],
w[(j[#]&/@Range[Length[k]]).
        Reverse[FoldList[Times,1,Reverse[Rest[k]]]]]
E^(2 Pi I Plus @@
MapIndexed[j[#2[[1]]] schema[[#2[[1]]]]/#1 &,k])
/. (j[#] ->0 &/@ Flatten[Position[schema,D]])
] / Sqrt[Times@@k]
```

Example Consider strings of length 3 whose characters are taken from alphabets with cardinality 2, 4, and 2. Calculate several schema averages in terms of the generalized Walsh coefficients.

```
In[14]:= SchemaToWalsh[{2,4,2},{D,D,D},w]

          w[0]
Out[14]= ----
           4
```

```
In[15]:= SchemaToWalsh[{2,4,2},{0,D,D},w]

          w[0] + w[8]
Out[15]= -----------
               4

In[16]:= SchemaToWalsh[{2,4,2},{1,D,D},w]

          w[0] - w[8]
Out[16]= -----------
               4

In[17]:= SchemaToWalsh[{2,4,2},{0,0,D},w]

          w[0] + w[2] + w[4] + w[6] + w[8] + w[10] + w[12] + w[14]
Out[17]= ---------------------------------------------------------
                                  4

In[18]:= SchemaToWalsh[{2,4,2},{D,0,D},w]
```

```
          w[0] + w[2] + w[4] + w[6]
Out[18]= -------------------------
                       4
```

A.6 Hadamard Transforms

```
(* tensor[a,b] returns the tensor produce of a and b *)
tensor[a_,b_]:=With[{s1=Length[a],s2=Length[b]},
       Table[a[[Ceiling[i/s2],Ceiling[j/s2]]]
                b[[Mod[i-1,s2]+1,Mod[j-1,s2]+1]],{i,s1 s2},{j,s1 s2}]]

(* returns a Hadamard matrix for a single character *)
h[k_]:= Table[E^(2 Pi I i j/k),{i,0,k-1},{j,0,k-1}]

(* returns a Hadamard matrix for a set of characters *)
H[k_List]:= Fold[tensor[#1,#2]&,h[First[k]],h/@Rest[k]]

(* returns the matrix for finding deceptive conditions *)
```

```
M[k_List]:= With[{hmat=H[k]},

        Table[hmat[[1]]-hmat[[i]],{i,2,Length[hmat]}]]
```

A.7 Determining Deception Using Hadamard Transforms

```
(* returns a list of Walsh coefficients in the partition *)

(* specified by the schema *)

WalshList[k_List,schema_List,w_]:=

        Flatten[

        Table @@

        Prepend[

        Delete[MapIndexed[{j[#2[[1]]],0,#1-1}&,k],Position[schema,D]],

        w[(j[#]&/@Range[Length[k]]).

                Reverse[FoldList[Times,1,Reverse[Rest[k]]]]]

        /. (j[#] ->0 &/@ Flatten[Position[schema,D]])

        ]]


(* returns the Walsh sums needed for the deceptive conditions *)

(* F=fixed position, D = don't care *)

Decep[k_List,schema_List,w_]:=
```

```
M[k[[#[[1]]]]& /@ Position[schema,F]].WalshList[k,schema,w]
```

Example Consider a function over a (3,2,2)-ary alphabet and the competition partition (F,D,D). `Decep[{3,2,2},{F,D,D},w]` returns a vector whose components must all be positive for the function to be deceptive.

```
In[3]:= Decep[{3,2,2},{F,D,D},w]

                (2 I)/3 Pi                (-2 I)/3 Pi
Out[3]= {(1 - E           ) w[4] + (1 - E            ) w[8],

            (-2 I)/3 Pi                (2 I)/3 Pi
>    (1 - E            ) w[4] + (1 - E           ) w[8]}
```

Example Now we consider the above function over the competition partition (F,F,D).

```
In[4]:= Decep[{3,2,2},{F,F,D},w]
```

```
Out[4]= {2 w[2] + 2 w[6] + 2 w[10],

           (2 I)/3 Pi               (2 I)/3 Pi
>    (1 - E          ) w[4] + (1 - E          ) w[6] +

            (-2 I)/3 Pi               (-2 I)/3 Pi
>     (1 - E           ) w[8] + (1 - E           ) w[10],

                  (2 I)/3 Pi               (2 I)/3 Pi
>    2 w[2] + (1 - E          ) w[4] + (1 + E          ) w[6] +

            (-2 I)/3 Pi               (-2 I)/3 Pi
>     (1 - E           ) w[8] + (1 + E           ) w[10],

           (-2 I)/3 Pi               (-2 I)/3 Pi
>    (1 - E           ) w[4] + (1 - E           ) w[6] +

            (2 I)/3 Pi               (2 I)/3 Pi
>     (1 - E          ) w[8] + (1 - E          ) w[10],
```

```
                    (-2 I)/3 Pi                (-2 I)/3 Pi
>    2 w[2] + (1 - E           ) w[4] + (1 + E           ) w[6] +

             (2 I)/3 Pi                 (2 I)/3 Pi
>     (1 - E          ) w[8] + (1 + E          ) w[10]}
```

A.8 Determining Deception to Specified Order

```
GenDecep[k_List,order_,w_]:=
        Flatten[
        Decep[k,#,w]& /@
        Permutations[Join[Table[F,{order}],
                Table[D,{Length[k]-order}]]]]
```

Example Consider the function used in the previous example. In order for the function to be deceptive at the 1-st order, all the components of `GenDecep[{3,2,2},1,w]` must be positive.

```
In[5]:= GenDecep[{3,2,2},1,w]
```

```
                (2 I)/3 Pi                (-2 I)/3 Pi
Out[5]= {(1 - E          ) w[4] + (1 - E           ) w[8],

            (-2 I)/3 Pi                (2 I)/3 Pi
>    (1 - E           ) w[4] + (1 - E          ) w[8], 2 w[2], 2 w[1]}
```

A.9 Fully Deceptive Conditions

```
FullDecep[k_List,w_]:=
        Flatten[Table[GenDecep[k,q,w],{q,1,Length[k]-1}]]
```

Example Consider again a function over strings taken from a (3,2,2)-ary alphabet. Then the conditions for full deception are that each of the components of the vector returned by `FullDecep[{3,2,2},w]` are positive.

```
In[6]:= FullDecep[{3,2,2},w]

                (2 I)/3 Pi                (-2 I)/3 Pi
Out[6]= {(1 - E          ) w[4] + (1 - E           ) w[8],
```

```
                (-2 I)/3 Pi                  (2 I)/3 Pi
>    (1 - E             ) w[4] + (1 - E             ) w[8], 2 w[2], 2 w[1],

>    2 w[2] + 2 w[6] + 2 w[10],

                (2 I)/3 Pi                  (2 I)/3 Pi
>    (1 - E             ) w[4] + (1 - E             ) w[6] +

                 (-2 I)/3 Pi                  (-2 I)/3 Pi
>     (1 - E             ) w[8] + (1 - E             ) w[10],

                       (2 I)/3 Pi                  (2 I)/3 Pi
>    2 w[2] + (1 - E             ) w[4] + (1 + E             ) w[6] +

                 (-2 I)/3 Pi                  (-2 I)/3 Pi
>     (1 - E             ) w[8] + (1 + E             ) w[10],

                (-2 I)/3 Pi                  (-2 I)/3 Pi
>    (1 - E             ) w[4] + (1 - E             ) w[6] +
```

```
                (2 I)/3 Pi                  (2 I)/3 Pi
>      (1 - E           ) w[8] + (1 - E           ) w[10],

                    (-2 I)/3 Pi                 (-2 I)/3 Pi
>     2 w[2] + (1 - E            ) w[4] + (1 + E            ) w[6] +

                (2 I)/3 Pi                  (2 I)/3 Pi
>      (1 - E           ) w[8] + (1 + E           ) w[10],

                                        (2 I)/3 Pi
>     2 w[1] + 2 w[5] + 2 w[9], (1 - E           ) w[4] +

                (2 I)/3 Pi                  (-2 I)/3 Pi
>      (1 - E           ) w[5] + (1 - E            ) w[8] +

                (-2 I)/3 Pi                         (2 I)/3 Pi
>      (1 - E            ) w[9], 2 w[1] + (1 - E           ) w[4] +

                (2 I)/3 Pi                  (-2 I)/3 Pi
```

```
>     (1 + E          ) w[5] + (1 - E           ) w[8] +

            (-2 I)/3 Pi              (-2 I)/3 Pi
>     (1 + E           ) w[9], (1 - E           ) w[4] +

            (-2 I)/3 Pi               (2 I)/3 Pi
>     (1 - E           ) w[5] + (1 - E          ) w[8] +

            (2 I)/3 Pi                       (-2 I)/3 Pi
>     (1 - E          ) w[9], 2 w[1] + (1 - E           ) w[4] +

            (-2 I)/3 Pi               (2 I)/3 Pi
>     (1 + E           ) w[5] + (1 - E          ) w[8] +

            (2 I)/3 Pi
>     (1 + E          ) w[9], 2 w[1] + 2 w[3], 2 w[2] + 2 w[3],

>    2 w[1] + 2 w[2]}
```

Appendix B

Another Basis for Generalized Transforms

As in Fourier series, we can use sines and cosines as well as complex exponentials as our basis functions. The disadvantage of using sines and cosines as basis functions is that the basic theorems become somewhat more complex. However, the advantage is that all the Walsh coefficients of a real function are real in this basis. We will refer to the generalized Walsh functions and transforms using sines as cosines as real generalized Walsh functions and transforms.

Since $\cos(x) = \frac{e^{\imath x}+e^{-\imath x}}{2}$ and $\sin(x) = \frac{e^{\imath x}-e^{-\imath x}}{2\imath}$, one can convert between the generalized Walsh coefficients used in the previous sections to the real generalized Walsh coefficients.

To start, consider a function over strings of length 1. There are two cases for the transform itself, one for even k and one for odd k.

Odd k The real generalized Walsh transform is given by

$$\begin{aligned} a_0 &= w_0, \\ a_j &= \frac{w_j + w_{k-j}}{2^{1/2}} \text{ for } j \neq 0, \\ b_j &= \frac{\imath}{2^{1/2}}(w_j - w_{k-j}). \end{aligned} \tag{B.1}$$

The inverse real generalized Walsh transform is given by the following:

$$f(x) = \frac{1}{k^{1/2}}(a_0 + \sum_{j=1}^{\frac{k-1}{2}} (a_j 2^{1/2} \cos(2\pi jx/k) + b_j 2^{1/2} \sin(2\pi jx/k))). \tag{B.2}$$

Of course, one can express the a_js and b_js in terms of the inner products of the cosines and sines with the function f:

$$\begin{aligned} a_0 &= \frac{1}{k^{1/2}} \sum_{x=0}^{k-1} f(x); \\ a_j &= \frac{2^{1/2}}{k^{1/2}} \sum_{x=0}^{k-1} \cos(2\pi jx/k) f(x) \text{ for } j \neq 0; \\ b_j &= \frac{2^{1/2}}{k^{1/2}} \sum_{x=0}^{k-1} \sin(2\pi jx/k) f(x). \end{aligned} \tag{B.3}$$

Even k

$$\begin{aligned} a_0 &= w_0; \\ a_{k/2} &= 2^{-1/2} w_{k/2}; \\ a_j &= \frac{w_j + w_{k-j}}{2^{1/2}} \text{ for } j \neq 0 \text{ and } j \neq k/2; \\ b_j &= \frac{\imath}{2^{1/2}}(w_j - w_{k-j}). \end{aligned} \tag{B.4}$$

$$f(x) = \frac{1}{k^{1/2}}(a_0 + \sum_{j=1}^{\frac{k}{2}-1} (a_j 2^{1/2} \cos(2\pi jx/k) \quad + \quad b_j 2^{1/2} \sin(2\pi jx/k)) \\ + \quad 2^{1/2} \cos(\pi x) a_{k/2}). \qquad \text{(B.5)}$$

Again, the coefficients can be expressed in terms of the inner product of sines and cosines with $f(x)$:

$$\begin{aligned} a_0 &= \frac{1}{k^{1/2}} \sum_{x=0}^{k-1} f(x); \\ a_j &= \frac{2^{1/2}}{k^{1/2}} \sum_{x=0}^{k-1} \cos(2\pi jx/k) f(x) \text{ for } j \neq 0; \\ b_j &= \frac{2^{1/2}}{k^{1/2}} \sum_{x=0}^{k-1} \sin(2\pi jx/k) f(x). \end{aligned} \qquad \text{(B.6)}$$

The proof that the inverse real generalized Walsh transform of the real generalized Walsh transform of a function is the function itself comes from substituting the expressions for a_j and b_j (B.1,B.4) into the expressions for $f(x)$ (B.2,B.5) and verifying that it indeed is the inverse generalized Walsh transform (3.7).

There are two ways of generalizing the above method to n dimensions. The first is simply to take the Fourier transform along each dimension as we did before, but to use sines and cosines as we have done above. The problem with using this method is mainly notational complexity, although the idea is just as simple as the generalized Walsh transforms we discussed before; all we are doing is taking the Fourier transform in an n-dimensional space, using sines and cosines. As this first method of generalizing the sine and cosine transform becomes cumbersome for long strings, it will not be

pursued any further in this thesis.

The second method of generalizing the Walsh transform using sines and cosines works as follows:

$$\begin{aligned}
a_{\vec{0}} &= \prod_m k_m^{-1/2} \sum_{\vec{x}} f(\vec{x}); \\
a_{\vec{j}} &= 2^{1/2} \prod_m k_m^{-1/2} \sum_{x_1,x_2,\ldots,x_n} \cos(2\pi(j_1x_1/k_1 + j_2x_2/k_2 + \ldots j_nx_n/k_n) f(\vec{x}) \text{ for } j \neq 0; \\
b_{\vec{j}} &= 2^{1/2} \prod_m k^{-1/2} \sum_{x_1,x_2,\ldots,x_n} \cos(2\pi(j_1x_1/k_1 + j_2x_2/k_2 + \ldots j_nx_n/k_n) f(\vec{x}). \qquad \text{(B.7)}
\end{aligned}$$

$$\begin{aligned}
f(\vec{x}) &= \prod_m k_m^{-1/2} \sum_{\vec{j}} \{a_{\vec{0}} + a_{\vec{j}} 2^{1/2} \cos(2\pi(j_1x_1/k_1 + j_2x_2/k_2 + \ldots j_nx_n/k_n)) \\
&\quad + b_{\vec{j}} 2^{1/2} \sin(2\pi(j_1x_1/k_1 + j_2x_2/k_2 + \ldots j_nx_n/k_n))\}. \qquad \text{(B.8)}
\end{aligned}$$

This method has the disadvantage that the Walsh functions in n dimensions is not just the product of Walsh functions in one dimension. Explicitly, they are the following:

Definition 11 (Real $\vec{k}$-ary Walsh Functions)

$$\begin{aligned}
\Omega_0^{(\vec{k})}(\vec{x}) &= \prod_m k_m^{-1/2}; \\
\Omega_{\vec{j}}^{(\vec{k})}(\vec{x}) &= 2^{1/2} \prod_m k_m^{-1/2} \cos(2\ pi(j_1x_1/k_1 + j_2x_2/k_2 + \ldots j_nx_n/k_n)) \textit{ for } j \neq 0; \\
\Lambda_{\vec{j}}^{(\vec{k})}(\vec{x}) &= 2^{1/2} \prod_m k_m^{-1/2} \sin(2\pi(j_1x_1/k_1 + j_2x_2/k_2 + \ldots j_nx_n/k_n)). \qquad \text{(B.9)}
\end{aligned}$$

This transform also has the useful property that function averages over schema with m fixed positions become sums over $n - m$ positions in the transformed space.

Example Consider again a function f over strings of length 2 whose characters are

taken from a ternary alphabet. Some schemas averages in terms of the transform coefficients are

$$\begin{aligned} f(**) &= \frac{1}{3}a_{(0,0)}; \\ f(*0) &= \frac{1}{3}(a_{(0,0)} + a_{(0,1)}); \\ f(*1) &= \frac{1}{3}(a_{(0,0)} - \frac{1}{2}a_{(0,1)} + \frac{3^{1/2}}{2}b_{(0,1)}); \\ f(*2) &= \frac{1}{3}(a_{(0,0)} - \frac{1}{2}a_{(0,1)} - \frac{3^{1/2}}{2}b_{(0,1)}). \end{aligned} \tag{B.10}$$

Theorem 7 *Consider a schema has m fixed characters p_i at positions j_i. Then the average of f over that schema is*

$$\begin{aligned} \prod_q k_q^{-1/2} \sum_{l_1} \sum_{l_2} \cdots \sum_{l_m} & \quad \cos(2\pi(l_1 p_1/k_{j_1} + l_2 p_2/k_{j_2} + \ldots + l_m p_m/k_{j_m})) \\ & \quad a_{(0,0,\ldots,0,l_1,0,\ldots,0,l_2,0,\ldots,0,l_m,0,\ldots)} \\ + & \quad \sin(2\pi(l_1 p_1/k_{j_1} + l_2 p_2/k_{j_2} + \ldots + l_m p_m/k_{j_m})) \\ & \quad b_{(0,0,\ldots,0,l_1,0,\ldots,0,l_2,0,\ldots,0,l_m,0,\ldots)}. \end{aligned} \tag{B.11}$$

References

Beauchamp, K. G. (1975). *Walsh functions and their applications*, Academic Press.

Bethke, A. D. (1981), *Genetic algorithms as function optimizers* (Doctoral dissertation, University of Michigan). *Dissertation Abstracts International, 41(9),* 3503B. (University Microfilms No. 8106101).

Bledsoe, W. W. (1961). *The use of biological concepts in the analytical study of systems.* Paper presented at the ORSA-TIMS National Meeting, San Francisco, CA.

Bridges, C., Goldberg, D. E. (1989). *A note on the nonuniform Walsh-schema transform.* TCGA Report No. 89004. Tuscaloosa, AL: University of Alabama, Department of Engineering Mechanics, The Clearinghouse for Genetic Algorithms.

Das, R., & Whitley, D. (1991). The only challenging problems are deceptive: global search by solving order-1 hyperplanes. *Proceedings of the Fourth International Conference on Genetic Algorithms,* 166-173.

Davis, L. (1991). Bit-climbing, representational bias, and test suite design. *Proceedings of the Fourth International Conference on Genetic Algorithms,* 18-23.

Deb, K. & Goldberg, D. E. (1991). *Analyzing Deception in Trap Functions.* (IlliGAL Report No. 91009). Urbana: University of Illinois, Illinois Genetic Algorithms

Laboratory.

Deb, K., & Goldberg, D. E. (1992). *Sufficient conditions for deceptive and easy binary functions.* (IlliGAL Report No. 92001). Urbana: University of Illinois, Illinois Genetic Algorithms Laboratory.

Davidor, Y. (1991). Epistasis variance: a viewpoint on GA-hardness. *Foundations of Genetic Algorithms,* 23-35.

Forrest, S., & Mitchell, M. (1991). The performance of genetic algorithms on Walsh polynomials: some anomalous results and their explanation. *Proceedings of the Fourth International Conference on Genetic Algorithms,* 182-189.

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal, deceptive problem. *Genetic Algorithms and Simulated Annealing*, 74-88. Los Altos: Morgan Kaufmann Publishers, Inc.

Goldberg, D. E. (1989a). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

Goldberg, D. E. (1989b). Genetic algorithms and Walsh functions: Part I, a gentle introduction. *Complex Systems, 3,* 129-152.

Goldberg, D. E. (1989c). Genetic algorithms and Walsh functions: Part II, deception and its analysis. *Complex Systems, 3,* 153-171.

Goldberg, D. E. (1990a). Real-coded genetic algorithms, virtual alphabets, and blocking. *Complex Systems, 5*(2), 139-168.

Goldberg, D. E. (1990b). *Construction of higher-order deceptive functions using low-order Walsh coefficients.* (IlliGAL Report No. 90002). Urbana: University of Illinois, Illinois Genetic Algorithms Laboratory.

Goldberg, D. E., Deb, K., & Clark, J. H. (1991). *Genetic Algorithms, noise, and the sizing of populations.* (IlliGAL Report No. 91010). Urbana: University of Illinois, Illinois Genetic Algorithms Laboratory.

Goldberg, D. E., Deb, K., & Horn, J. (1992). *Massive multimodality, deception, and genetic algorithms.* (IlliGAL Report No. 92005). Urbana: University of Illinois, Illinois Genetic Algorithms Laboratory.

Goldberg, D. E., & Rudnick, M. (1991). Genetic algorithms and the variance of fitness. *Complex Systems, 5,* 265-278.

Grefenstette, J. J. (1991). Building block hypothesis considered harmful. *Genetic Algorithms Digest* [email journal], *5*(19).

Hart, W. & Belew, R. Optimizing an arbitrary function is hard for the genetic algorithm. *Proceedings of the Fourth International Conference on Genetic Algorithms,* 190-195.

Holland, J. H. (1975). *Adaption in natural and artificial systems.* Ann Arbor, MI: University of Michigan Press.

Homaifar, A., Qi, X., & Fost, J. (1991). Analysis and design of a general GA deceptive problem, *Proceedings of the Fourth International Conference on Genetic Algorithms,* 196-203.

Homaifar, A., & Qi, X. (1990). Analysis of GAs deception by Hadamard transform. IASTED International Symposium Machine Learning and Neural Networks. New York.

Kargupta, H., Deb, K., & Goldberg, D. E. (1992). *Ordering genetic algorithms and deception.* (IlliGAL Report No. 92006). Urbana: University of Illinois, Illinois Genetic Algorithms Laboratory.

Kauffman, S. A. (1989). Adaption on rugged fitness landscapes. In D. L. Stein, ed., *Lectures in the sciences of complexity,* pp. 527-618. Reading: Addison-Wesley.

Kauffman, S. A. (1990). Requirements for evolvability in complex systems: orderly dynamics and frozen components. *Physica D, 42,* 135-152.

Kauffman, S. & Levin, S. (1987). Towards a general theory of adaptive walks on rugged landscapes. *Journal of theoretical Biology 128,* 11-45.

Kirkpatrick, S. , Gelatt, C., & Vecci, M. (1983). Optimization by simulated annealing. *Science, 220*, 671-680.

Liepins, G. E., & Vose, M. D. (1990a). Polynomials, basic sets, and deceptiveness in genetic algorithms. *Complex Systems, 5,* 45-61.

Liepins, G. E., & Vose, M. D. (1990b). Representational issues in genetic algorithms. *Journal of Experimental and Theoretical Artificial Intelligence, 2,* 101-115.

Liepins, G. E., & Vose, M. D. (1991). Deceptiveness and genetic algorithm dynamics. *Foundations of Genetic Algorithms,* 36-50.

Lipsitch, M. (1991). Adaption on rugged landscapes generated by iterated local interactions of neighboring genes. *Proceedings of the Fourth International Conference on Genetic Algorithms,* 128-135.

Lighthill, M. J. (1959). *Introduction to Fourier analysis and generalised functions.* New York: Cambridge University Press.

Manderick, B., de Weger, M., & Spiessens, P. (1991). The genetic algorithm and the structure of the fitness landscape. *Proceedings of the Fourth International Conference on Genetic Algorithms,* 143-150.

Mason, A. J. (1991). Partition coefficients, static deception and deceptive problems for non-binary alphabets. *Proceedings of the Fourth International Conference*

on Genetic Algorithms, 210-214.

Mitchell, M. & Forrest, S. (1991). *What is deception anyway? And what does it have to do with GAs?* Unpublished manuscript.

Mitchell, M. Forrest, S., & Holland, J. (1991). *The royal road for genetic algorithms: fitness landscapes and GA performance.* Unpublished manuscript.

Radcliffe, N. (1991). Forma analysis and random respectful mutations. *Proceedings of the Fourth International Conference on Genetic Algorithms,* 222-229.

Rudnick, M. (1991). *Genetic algorithms and the variance of fitness.* Unpublished Doctoral Dissertation, Oregon Graduate Institute, Department of Computer Science and Engineering, Beaverton, OR.

Rudnick, M., & Goldberg, D. E. (1991). *Signal, noise, and genetic algorithms.* (IlliGAL Report No. 91005). Urbana: University of Illinois, Illinois Genetic Algorithms Laboratory.

Schaffer, J. D., Eshelman, L. J., & Offutt, D. (1991). Spurious correlations and premature convergence in genetic algorithms. *Foundations of Genetic Algorithms,* 102-112.

Sikora, R. (1991). *Analysis of deception for permutation problems.* Unpublished manuscript.

Szu, H., & Hartley, R. (1987). Fast simulated annealing. *Physics Letters A, 122* (3,4), 157.

Tanese, R. (1989). Distributed genetic algorithms for function optimization. (Doctoral Dissertation, University of Michigan). *Dissertation Abstracts International, 50,* 5180B. (University Microfilms No. 90-01722). , Ann Arbor, MI).

Tolstov, G. P. (1962). *Fourier series.* Englewood Cliffs, NJ: Prentice-Hall, Inc.

Vose, M., & Liepins, G. (1991). Schema disruption. *Proceedings of the Fourth International Conference on Genetic Algorithms,* 237-242.

Weinberger, E. (1987). A stochastic generalization of Eigen's model of natural selection (Doctoral dissertation, New York University). *Dissertation Abstracts International, 48,* 2000B. (University Microfilms No. 87-22798)

Weinberger, E. (1988). A more rigorous derivation of some properties of uncorrelated fitness landscapes [Letters to the editor]. *Journal of Theoretical Biology, 134,* 125-129.

Weinberger, E. (1990). Correlated and uncorrelated fitness landscapes and how to tell the difference. *Biological Cybernetics, 63,* 325-336.

Whitley, D. (1991a). Fundamental principles of deception in genetic search. *Foundations of Genetic Algorithms,* 221-241.

Whitley, D. (1991b). Deception, dominance and implicit parallelism (Technical Report No. CS-91-120). Fort Collins: Colorado State University, Department of Computer Science.

Wilson, S. (1991). GA-easy does not imply steepest-ascent optimizable. *Proceedings of the Fourth International Conference on Genetic Algorithms,* 85-89.

Wright, A. H. (1991). Genetic algorithms for real parameter optimization. *Foundations of Genetic Algorithms,* 205-218.