Complex Systems **3** (1989) 153–171

Genetic Algorithms and Walsh Functions: Part II, Deception and Its Analysis

David E. Goldberg
Department of Engineering Mechanics, University of Alabama
Tuscaloosa, AL 35487, USA

Abstract. Part I considered the application of Walsh functions to the analysis of genetic algorithms operating on different coding-function combinations. In this paper, those methods are extended to permit rigorous analysis of deception by considering the expected disruption to schema processing caused by different genetic operators. Algebraic extensions of these methods are considered, and a sensitivity analysis is described.

1. Introduction

For some time, Walsh methods have been available for the analysis of genetic algorithms (GAs) operating on different coding-function combinations [1]; however, these methods have not been widely cited, nor have they been widely used. In Part I, I attempted to make the Walsh functions and Bethke's Walsh-schema transform more accessible by presenting the Walsh functions in polynomial form. Although such a presentation is unconventional, it is mathematically equivalent to other discussions, and the use of polynomials makes proofs and algebraic manipulation more direct and intuitive. In that paper, I also hammered away at the basic notions of Walsh-GA analysis by illustrating the techniques with several simple examples, including a bitwise linear function, a simple nonlinear function, and two *deceptive* functions. Although the paper demonstrated the analysis and design of deceptive functions by example and those functions were consistent with previous uses of the term [3,4], the definition of deception was less than rigorous. The main objective in this *rapprochement* is to remedy that omission.

Specifically, this paper considers a rigorous definition of what is meant by a deceptive function, and this definition is used to define a procedure for the analysis of deception in particular coding-function combinations. The paper also derives procedures for analyzing deception using algebraic methods alone, and sensitivity analysis is proposed for constrained function analysis when penalty methods are imposed.

© 1989 Complex Systems Publications, Inc.

In the remainder of the paper, Walsh functions and GA terminology are briefly reviewed. Deception is then rigorously defined, and an analysis of deception (ANODE) procedure is described and illustrated with two examples. Algebraic coding-function analysis is outlined and illustrated by a simple example. The use of sensitivity analysis is also briefly outlined, with some discussion of its use in the analysis of penalty functions. Finally, a number of extensions of these methods are discussed, including analyses of different operators, codings, and nonbinary alphabets.

2. A brief review of Walsh functions and GAs

As in the previous paper, we assume that the genetic algorithm processes l-bit strings

$$\mathbf{x} = x_l x_{l-1} \ldots x_2 x_1, \tag{2.1}$$

where $x_i \in \{0, 1\}$. Bit strings may be mapped to auxiliary form by the equation $y_i = 1 - 2x_i, 1, \ldots, l$, where the $y_i \in \{-1, 1\}$. With the strings so mapped, the Walsh functions may be defined as the complete set of monomials over the l auxiliary variables y_i as follows:

$$\psi_j(\mathbf{y}) = \prod_{i=1}^{l} y_i^{j_i}, \tag{2.2}$$

where the Walsh function index j is used bit by bit to determine whether the ith position is represented in the product. Any fitness function $f(\mathbf{x})$ may be written as a linear combination of the Walsh functions as

$$f(\mathbf{x}) = \sum_{j=0}^{2^l-1} w_j \psi_j(\mathbf{x}), \tag{2.3}$$

where the necessary inverse auxiliary string mappings $\mathbf{x}(\mathbf{y})$ in the terms $\psi_j(\mathbf{x})$ are understood.

In the analysis of genetic algorithms, strings are treated as representatives of subsets containing strings similar to one another at some number of positions. Formally, a similarity template or *schema* of length l may be taken as a string $\mathbf{h} = h_l h_{l-1} \ldots h_2 h_1$, where $h_i \in \{0, 1, *\}$. The * is a wildcard or don't-care character, matching either a 1 or a 0. A primary quantity necessary for understanding genetic algorithm processing is the average fitness of a schema, $f(\mathbf{h})$. Part I demonstrated how this quantity, calculated in the usual manner as

$$f(\mathbf{h}) = \frac{\sum_{\mathbf{x} \in \mathbf{h}} f(\mathbf{x})}{|\mathbf{h}|}, \tag{2.4}$$

where $|\mathbf{h}|$ is the cardinality of the subset $\mathbf{h}$ (the number of members of the subset), may be calculated as a partial, signed sum of the Walsh coefficients as follows:

$$f(\mathbf{h}) = \sum_{j \in J(\mathbf{h})} w_j \psi_j(\beta(\mathbf{h})), \tag{2.5}$$

where the function β maps $*$ characters to 0's and leaves 0's and 1's alone, and the index set $J = \{j : \exists i : \mathbf{h} \subseteq \mathbf{h}_i(j)\}$, where $\mathbf{h}_i(j)$ is one of the $2^{o(j)}$ partition subsets defined by cycling over all possible fixed substrings at the $o(j)$ positions that contain a 1 in the binary representation of the index j. In words, the index set definition takes the sum over only those Walsh coefficients whose Walsh function value is fixed by the fixed positions of the schema.

With this review, we may define deception in the context of genetic search.

3. Defining deception

To define deception, we first recast the schema theorem in operator-adjusted fitness form. Thereafter, we calculate operator-adjusted fitness values using the Walsh coefficients, and use this calculation to define a set of points likely to attract increasing genetic algorithm samples as a population converges.

3.1 Recasting the schema theorem

The workings of genetic algorithms are usually approached through the eyes of the schema theorem. Under reproduction, simple crossover, and mutation, it may be shown [4,5] that the expected number of representatives m of a particular schema $\mathbf{h}$ is at least

$$m(\mathbf{h}, t+1) \geq m(\mathbf{h}, t) \frac{f(\mathbf{h})}{\bar{f}} \left[1 - p_c \frac{\delta(\mathbf{h})}{l-1} - p_m o(\mathbf{h}) \right], \tag{3.1}$$

where $f(\mathbf{h})$ is the average fitness of the representatives of $\mathbf{h}$ in the current population, $\bar{f}$ is the average fitness of the population, p_c is the crossover probability, p_m is the mutation probability, $\delta(\mathbf{h})$ is the defining length of the schema (the physical distance between the outermost fixed positions of the schema), and $o(\mathbf{h})$ is the order of the schema (the number of fixed positions in the schema). This inequality applies to all schemata, but it is often forgotten that the computation is a lower bound and the actual expected disruption is generally a good bit less. This point has been made more forcefully elsewhere [2,3]. Here, if we think of the product of the schema average fitness $f(\mathbf{h})$ and schema survival probability bound $[1 - p_c \frac{\delta(\mathbf{h})}{l-1} - p_m o(\mathbf{h})]$ as an operator-adjusted fitness value $f'(\mathbf{h})$, we may write the theorem in the simpler form

$$m(\mathbf{h}, t+1) \geq m(\mathbf{h}, t) \frac{f'(\mathbf{h})}{\bar{f}}. \tag{3.2}$$

Notice that this equation has the same form as that for reproduction alone. Intuitively, if we consider the singleton subsets (the individual strings), this equation suggests that the genetic algorithm should do no worse than choose among the alternative strings according to their operator-adjusted fitness

values f'.[1] This leads us to consider whether the operator-adjusted schema average fitness may be calculated in terms of the Walsh coefficients.

3.2 Operator-adjusted fitness values from Walsh coefficients

The lower bound on the operator-adjusted fitness value presented in the previous section is not very practical. A more useful estimate may be determined directly from the Walsh coefficients. Specifically, the usual schema average calculation,

$$f(\mathbf{h}) = \sum_{j \in J(\mathbf{h})} w_j \psi_j(\beta(\mathbf{h})), \tag{3.3}$$

may be adjusted term by term depending on whether a particular operator disrupts the similarity partition fixed by that term.

For example, consider crossover acting on the term $j = 7$ in a length $l = 5$ string. Fixing $j = 7$ means that we are focused on a schema that at least fixes the last three positions of the string, **fff, where the f's indicate a fixed position in the string. Thus the Walsh term should be expected to survive crossover with probability $[1 - p_c\frac{\delta(j)}{l-1}]$, where $\delta(j)$ is the defining length of the positions fixed by the Walsh term. Noting that when disruption occurs, the fitness achieved by the crossover product over these positions has a net value of zero (an odd number of 1's is as likely as an even number of 1's), the expected fitness over those positions is simply the product of the original Walsh coefficient and the survival probability. Terming this the crossover-adjusted Walsh-coefficient, w_j^c, we write

$$w_j^c = w_j \left[1 - p_c \frac{\delta(j)}{l-1}\right]. \tag{3.4}$$

Under mutation a similar adjustment may be made, but the analysis must consider the post-mutation product more carefully. Using the previous example, we expect no change under mutation when all bits fixed within a Walsh term survive mutation; more precisely, we expect a probability of survival $p_s(j) = [1-p_m]^{o(j)}$. On the other hand, when the bits of a Walsh term are changed by mutation, the outcomes are not equally likely. Most often a single bit among the defining bits of the term is changed; the probability of having more than one mutation is of order $O(p_m^2)$. When a single bit is changed, the corresponding sign of the Walsh function must also change because the number of bits has changed from odd to even or vice versa. Thus, the expected fitness of a Walsh term under mutation is well approximated as

$$w_j^m = w_j[1 - p_d] - w_j p_d. \tag{3.5}$$

[1]Strictly speaking, the calculation is not very meaningful when performed with full strings because the disruption is so high. Nonetheless, the notion that a GA should, in a static sense, do no worse than choose among the best on the basis of operator-adjusted fitness will prove useful as we turn to calculating better bounds using the Walsh coefficients.

In the usual case, the destruction probability, $p_d = 1 - p_s$, itself is well approximated by the expression $p_m o(j)$, resulting in the mutation-adjusted Walsh coefficient as

$$w_j^m = w_j[1 - 2p_m o(j)]. \tag{3.6}$$

Multiplying the multipliers for both mutation and crossover adjustment together (assuming independence of the operators) we obtain the following relationship for the operator-adjusted Walsh coefficient, w_j', after dropping cross-product terms:

$$w_j' = w_j \left[1 - p_c \frac{\delta(j)}{l-1} - 2p_m o(j)\right]. \tag{3.7}$$

The approximations made should not be detrimental to the calculation as long as the adjustment is made only when the sum within the brackets is greater than zero. Later, it will be useful to think of the operator-adjusted w as the sum of the original coefficient and a change in that coefficient:

$$w_j' = w_j + \Delta w_j', \tag{3.8}$$

where by simple algebra the change due to operator adjustment may be written as

$$\Delta w_j' = -w_j \left[p_c \frac{\delta(j)}{l-1} + 2p_c o(j)\right]. \tag{3.9}$$

With this calculation — or an analogous calculation for any set of proposed operators — we may calculate the operator-adjusted fitness for any schema by summing over the relevant operator-adjusted Walsh coefficients in a straightforward manner:

$$f'(\mathbf{h}) = \sum_{j \in J(\mathbf{h})} w_j' \psi_j(\beta(\mathbf{h})). \tag{3.10}$$

With the operator-adjusted coefficients and fitness values so defined, we may define deception rigorously.

3.3 Interpreting operator-adjusted fitness values

These calculations are straightforward enough, and in a moment we will consider how to use them in a rigorous definition of deception. But before we do, it may be useful to ponder the meaning of the operator adjustment. Is there some physical or intuitive interpretation we may attach to this adjustment to help us better understand the underlying assumptions of deception analysis?

Probably the easiest way to interpret operator-adjusted fitness is to imagine the performance of crossover and mutation on a single string contained in a large, randomly generated population. The calculation of that string's operator-adjusted fitness amounts to calculating the expected value of fitness following the application of the subject operators. If we assume that

reproduction acts independently of the genetic operators, we should expect it to select those strings with highest operator-adjusted fitness values. Thus, strings with the highest operator-adjusted fitness values should attract the most trials as the run progresses. This view, of course, ignores all dynamical and stochastic considerations, and that is why the word "static" is sometimes attached to this type of analysis. Nonetheless, despite its ignorance of dynamics and chance, static analysis of convergence is a reasonable starting point in coding-function analysis because of its proper inclusion of operator expected effect and its reasonable assumption regarding the elevating role of reproduction. With this somewhat idealized view of the workings of GAs, we move toward defining deception and an analysis of deception procedure.

3.4 Deception defined

On the road to defining deception, we yield a time or two to define other useful concepts. Writing the fitness value of the global optimum as f^*, we may define a near-optimal set N as all those points that possess fitness values within an ϵ of the best:

$$N = \{x : f^* - f(x) \leq \epsilon\}. \tag{3.11}$$

Setting $\epsilon = 0$, the set N becomes the globally optimal set.

Similarly, global operator-adjusted fitness value may be written as f'^* and the corresponding operator-adjusted near-optimal set may be defined as

$$N' = \{x : f'^* - f'(x) \leq \epsilon'\}, \text{where } \epsilon' = \frac{f'^* - w_0}{f^* - w_0}\epsilon. \tag{3.12}$$

In this way, ϵ' is scaled to be the same proportion of the difference between maximum and average fitness in both the ordinary and operator-adjusted near-optimal sets.

With these definitions, deception itself may be defined straightaway. A function is *statically deceptive* or just *deceptive* when the intersection of the near-optimal set and the operator-adjusted near-optimal set loses some members when compared to the near-optimal set — when the set difference is nonempty.[2]

Definition 1. *A function-coding combination is* **statically deceptive** *at the level* ϵ *when* $N - N' \neq \emptyset$.

A function-coding combination is said to be *strictly statically deceptive* when the intersection of N and N' is empty (when $N - N' = N$).

A function-coding combination is said to be *statically easy* or just *easy* when all members of the near-optimal set are present in the corresponding operator-adjusted set.

[2]Recall that the set difference $A - B$ contains all points in A not in B: $A - B = \{x : x \in A \text{ and } x \notin B\}$.

Definition 2. *A function-coding combination is said to be statically easy at the level ϵ when $N - N' = \emptyset$.*

A function-coding combination is said to be *strictly statically easy* when the operator-adjusted near-optimal set coincides with the near-optimal set; that is, when $N = N'$.

Intuitively, if we view the operator-adjusted set as the set of points preferred by the genetic operators acting in concert with reproduction, simplicity and deception indicate whether or not those points coincide with the optima or near-optima of the function.

4. Analysis of deception

In a static sense, operator-adjusted fitness defines the fixed points — the convergence points or attractors — of genetic adaptive search. Whether or not these attractors correspond to the optimal or near-optimal points of the unadulterated function is the essence of whether that function is simple or deceptive. The definitions of the previous section permit the construction of an efficient procedure for determining whether and to what degree a coding-function combination is deceptive. This section considers the apparatus required for so doing and applies the technique to a number of simple examples.

4.1 Whom to check and whom to forget?

As the definition of deception suggests, a rigorous analysis of deception procedure is a relatively straightforward matter of comparing pre- and post-adjustment near-optimal sets. The naive approach to this comparison would calculate the near-optimal set, calculate the Walsh transform (the w's), calculate the operator-adjusted w's, calculate the inverse Walsh transform of the adjusted w's, calculate the operator-adjusted near-optimal set, and determine the overlap between the two sets. A more sophisticated approach tries to limit the number of points requiring consideration and attempts to take advantage of the usual sparsity of the Walsh coefficients. To establish such a procedure, we consider the pictorial representation of the search space in figure 1.

In the figure, we imagine a fitness-ordered list of points, starting with the globally optimal point at the top and proceeding in descending order. The points contained within the near-optimal set N are shown within an ϵ of the best. The complement of the near-optimal set, the set N^c, is shaded and marked, and normally we would expect to have to perform the aforementioned computations on all the points in the space, but some straightforward reasoning can eliminate many points from evaluation. The only points that must be considered in the complementary set are those that have some hope of surpassing the least fit points of the original near-optimal set. A bound may be obtained on the maximum change in fitness that a single point may

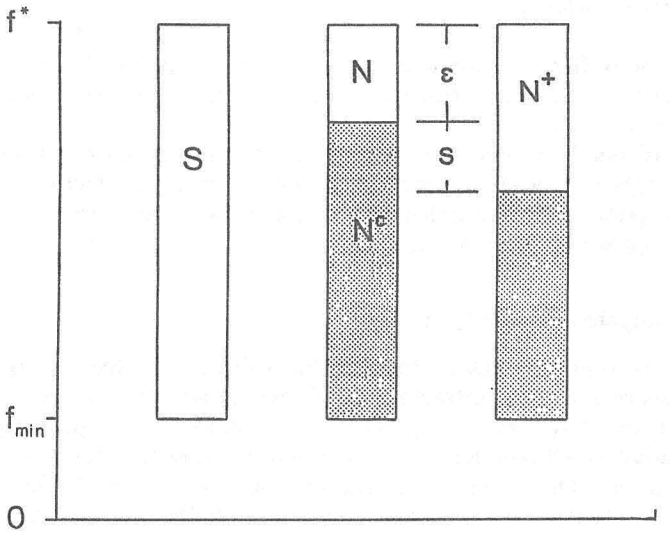

Figure 1: Three representations of the search space, S, showing the near-optimal set, N, its complement, N^c, the enlarged ANODE set, N^+, and their relation to the near-optimal set tolerance, ϵ, and the potential shift, s.

undergo due to operator adjustment. A single point may change at most an amount $\Delta f'_{max}$ given as

$$\Delta f'_{max} = \sum_{j=0}^{2^l-1} |\Delta w'_j|, \tag{4.1}$$

where the $\Delta w'_j$ are the operator-adjusted values determined as discussed in the previous section. In the extreme, if a low point were to have its fitness raised by this amount, and if the lowest point in the near-optimal set were to have its fitness lowered by $\Delta f'_{max}$, it would be necessary to check the fitness values of all points within $s = 2\Delta f'_{max}$ of the least fit point of N. We call this quantity s the *potential shift* and use it in the analysis of deception algorithm to determine the number of points that must be checked for membership in the operator-adjusted near-optimal set N'.

4.2 An analysis of deception (ANODE) algorithm

Using the potential shift value, an analysis of deception (ANODE) algorithm may be defined quite directly:

1. Calculate f^*.

2. Determine the w_j from the f_i.
3. For all nonzero w_j, determine the operator-adjusted $\Delta w'_j$.
4. Determine the potential shift, $s = 2\sum_{j:w_j \neq 0} |\Delta w'_j|$.
5. Determine the elements of $N^+ = \{i : f_i \geq f^* - \epsilon - s\}$. If $N^+ = N$, the coding-function combination is strictly simple and no further computations are required.
6. Determine the fitness value changes, $\Delta f'_i = \Delta f'_i(\Delta w'_j)$, as a result of operator adjustment for all elements of the enlarged set, $i \in N^+$, and calculate the operator-adjusted fitness values, f'_i.
7. Calculate f'^* over the elements of N^+.
8. For the near-optimal set, N, check whether its elements are members of N' (whether for $j \in N$, $f'_j \geq f'^* - \epsilon'$). If all members of N are in N' then f is simple. If not all members of N are in N' then f is deceptive. If no members of N are in N' then f is strictly deceptive.
9. If the previous step indicates a simple function-coding combination, test the members of N^+ not in N (test the members of $N^+ - N$) for membership in the operator-adjusted near-optimal set, N'. If there are no such members, f is strictly simple.

The ANODE algorithm is straightforward and may be used to determine whether a function is deceptive or not, and if it is, to what degree. Whether the algorithm is practical, even as a research tool, depends upon its complexity, and even though a thorough complexity analysis is beyond the scope of this treatment, some consideration of the computational bottlenecks and how they can be eliminated is in order.

Steps 2 and 6 are the two places likely to dominate the complexity of the ANODE algorithm. In step 2, the usual method of calculating Walsh coefficients from function values, the fast Walsh transform, has complexity of order $O(|S| \log_2 |S|)$, where $|S|$ is the cardinality of the search space ($|S| = 2^l$). This bound on the computation is difficult to beat, because the FWT is the optimal algorithm for calculating the Walsh transform over non-sparse functions. If there is some prior knowledge of function structure, such as additive or multiplicative separability, such structure may be used by performing FWT's on the subfunctions (or the log of the subfunctions in the case of multiplicative separability) and performing the remainder of the ANODE algorithm on each subfunction.

If there is no such regularity in the function-coding structure, but the algebraic structure of the function and the coding is known, there is one final possibility for improving the efficiency of step 2: algebraic reduction. This notion is sufficiently important that we will discuss it more fully in a later section; briefly, if the fitness and coding functions may be expressed in polynomial form, the overall relationship between the function values and the

j	x	$f(\mathbf{x})$	w_j
0	000	10.00	7.55
1	001	15.00	-2.50
2	010	0.00	5.00
3	011	5.00	0.00
4	100	10.10	-0.05
5	101	15.10	0.00
6	110	0.10	0.00
7	111	5.10	0.00

Table 1: Walsh coefficients of the bitwise linear function from Part I.

bit variables may be expressed as a polynomial. Thereafter, straightforward algebraic reduction makes it possible to read off w values directly without performing a transform. Whether the FWT or the algebraic method should be used depends upon the complexity of the algebraic computations.

In step 6, in the worst case, there will be many nonzero Δw's and a full, fast Walsh transform may have to be performed, requiring $O(|S|\log_2|S|)$ additions. If the Δw_j are sparse, then the definition of the Walsh transform may be used to calculate the required $\Delta f_i'$. Assuming there are n_n nonzero Δw_j, and assuming there are $|N^+|$ elements of the enlarged set, then f' calculations may be performed in $O(n_n|N^+|)$ additions (assuming the signs are stored as data). A rough guide to choosing the FWT over the calculation by definition may be obtained by comparing the two complexity estimates. If the number of nonzero elements in the set of Walsh coefficients meets the criterion

$$n_n \leq \frac{|S|\log_2|S|}{|N^+|}, \tag{4.2}$$

then the definition of the Walsh transform should be used to calculate individual $\Delta f'$ values; otherwise, the fast Walsh transform should be used.

With the ANODE algorithm defined and partially analyzed, we turn to several examples of its application.

4.3 Using ANODE

Let's consider two applications of the ANODE algorithm. In the first, we resurrect the bitwise linear function of Part I:

$$f(x_3x_2x_1) = 10 + 5x_1 - 10x_2 + 0.1x_3, \quad x_i \in \{0,1\}.$$

The function and its Walsh transform are displayed in table 1. Analysis of deception proceeds quite smoothly in this case. Assuming an allowable $\epsilon = 0$ (we are interested only in globally optimal points), a low mutation rate ($p_m \approx 0$) and maximum crossover disruption ($p_c = 1.0$), we note an interesting thing. Only order two or greater Walsh coefficients undergo any change due to crossover, and in a bitwise linear function no such coefficients

j	x	w_j	f_j	$-p_c\frac{\delta(j)}{l-1}$	$\Delta w'_j$	$\Delta f'_j$	f'_j
0	000	15	28	0.0	0.0	-1.5	26.5
1	001	1	26	0.0	0.0	-4.5	21.5
2	010	2	22	0.0	0.0	-8.5	11.5
3	011	3	0	-0.5	-1.5	14.5	14.5
4	100	4	14	0.0	0.0	-1.5	12.5
5	101	5	0	-1.0	-5.0	7.5	7.5
6	110	6	0	-0.5	-3.0	11.5	11.5
7	111	-8	30	-1.0	8.0	-17.5	11.5

Table 2: ANODE analysis of the three-bit deceptive function of Part I.

exist. Thus we conclude immediately that the potential shift $s = 0.0$ and no points outside the best point must be checked for membership in the operator-adjusted globally optimal set: the bitwise linear function is strictly simple. This result generalizes immediately to any bitwise linear function under simple crossover and mutation; however, the inclusion of mutation requires a more careful analysis.[3]

In the second example, consider the three-bit deceptive function designed in Part I. The function and its Walsh transform are presented in table 2 (the table corresponds to the function after design; the corresponding table in Part I was used to design the function). Performing the ANODE algorithm is straightforward. Assuming low mutation rate and high crossover rate, we calculate the operator-adjustment factors coefficient by coefficient. These calculations are also tabulated in table 3. Assuming we are interested in the globally optimal set $\epsilon = 0.0$, we determine the potential shift as $s = 2(1.5 + 5.0 + 3.0 + 8.0) = 35.0$. Thus, the entire space must be checked, because all have fitness values greater than $30 - 0 - 35 = -5$. Taking the fast Walsh transform of the $\Delta w'$ values yields the changes to the fitness values, $\Delta f'$, shown in the table. Adding these to the function values of the unadulterated function yields the operator-adjusted fitness values, f'_j, shown in the table. The best point is now 000 and the pre-adjustment best, 111, is now among the worst. Thus, this function is strictly deceptive, as was the intent of the design procedure.[4]

[3]The example sidestepped the effect of mutation by setting the mutation rate to zero. To include mutation cleanly, simply split the deception analysis in two parts. In the first, consider the effect of mutation by calculating the post-mutation fitness of all order-one coefficients by multiplying the pre-operator coefficient by the multiplier $1-2p_m$ and leaving w_0 untouched. Since each order-one coefficient is so affected, and since all coefficients contribute to each fitness value sum, mutation simply scales the range of the values about the mean value of the original function. When ϵ' is similarly scaled, there will be no change in the post-mutation near-optimal set and the argument above concerning crossover may be applied to the scaled function without modification.

[4]This function would also be strictly deceptive if it were imbedded in a larger string, unless the linkage were sufficiently tight to drop the disruption probabilities to the point where 111 remained the best. The ANODE procedure may be used in a straightforward manner to roughly determine the required linkage.

5. Algebraic coding-function analysis

The previous section suggested how to avoid much of the computational overhead of analysis of deception procedures by using algebraic methods. Although we come to these methods in a quest for ANODE efficiency, their existence provides a tool for coding-function analysis and coding design that sometimes proceeds by simple inspection.

In this section, we examine how straightforward algebraic reduction may be used to obtain Walsh coefficients without resorting to fast Walsh transforms. This algebraic coding-function analysis procedure is applied to the simple nonlinear function of Part I, and a useful algebraic theorem is also discussed.

5.1 Defining a method

Assuming that a fitness function f is defined over a set of decision variables $\mathbf{d}$, and further assuming known mappings between the bit string $\mathbf{x}$ and the decision variables, $\mathbf{d}$, it is a straightforward matter to write the mapping from the strings into the fitness values as the composition of functions:

$$f = f(\mathbf{d}(\mathbf{x})). \tag{5.1}$$

If the functions f and the components of $\mathbf{d}$ are restricted to polynomials,[5] then it is clear that the composite function is itself a polynomial over the binary variables x_i. Taking this one step further and substituting the mapping from the auxiliary string (over $+1$ and -1) to the bits (over 0 and 1) usually written as

$$x_i = \frac{1}{2}[1 - y_i], \; i = 1, \ldots, l, \tag{5.2}$$

the fitness function may be written as a polynomial over the auxiliary string variables.

This polynomial may be further reduced by recalling from Part I that even powers of the y_i reduce to 1 and that odd powers reduce to the variable itself:

$$y_i^a = \begin{cases} 1, & \text{if } a \text{ even;} \\ y_i, & \text{if } a \text{ odd.} \end{cases} \tag{5.3}$$

After reducing the polynomial thus, coefficients of identical monomial terms may be added, whereupon we notice an interesting thing. The fitness function has been reduced to a linear combination of proper monomials over the auxiliary string variables, y_i: the fitness function has been reduced to a linear combination of the Walsh functions. Thus, it is a straightforward matter to obtain the Walsh coefficients simply by reading the coefficient

[5]This restriction is not too limiting, because to any desired accuracy tolerance, many functions of interest can be written as truncated polynomials.

associated with the appropriate monomial. This is most easily done by calculating the coefficient index as a binary integer where the ith position of the integer is set to 1 if the ith auxiliary string variable is in the product and 0 otherwise (e.g., the term y_2y_3 generates the index $j = 110_2 = 6$).

To drive these ideas home, let's reconsider the simple nonlinear function of Part I.

5.2 Algebraic analysis of a simple nonlinear function

We resurrect the simple nonlinear function of Part I to illustrate the workings of algebraic coding-function analysis. Specifically, we consider the function $f = d^2$, where d is coded as an unsigned binary integer. Substituting the relationship $d = \sum_{i=1}^{3} x_i 2^{i-1}$ into the fitness function, an equation relating f and the bit values x_i is obtained:

$$\begin{aligned} f(d(\mathrm{x})) &= [d(\mathrm{x})]^2; \\ &= \left(\sum_{i=1}^{3} 2^{i-1} x_i\right)^2; \\ &= (x_1 + 2x_2 + 4x_3)^2; \\ &= x_1^2 + 2x_1x_2 + 4x_1x_3 + 4x_2^2 + 16x_2x_3 + 16x_3^2. \end{aligned}$$

To complete the analysis, substitute the relationship between bit values and auxiliary string variables $x_i = \frac{1}{2}[1 - y_i]$ into $f(\mathrm{x})$, obtaining the following result:

$$\begin{aligned} f(\mathrm{y}) = \frac{1}{4}(49 &- 14y_1 + y_1^2 - 28y_2 + 4y_2y_1 + 4y_1^2 - 56y_3 + 8y_3y_1 \\ &+ 16y_3y_2 + 16y_3^2). \end{aligned}$$

Using the reduction relationship above, we note that the squared terms become constants, thus reducing the relationship to the Walsh function form:

$$f(\mathrm{y}) = \frac{1}{4}(70 - 14y_1 - 28y_2 + 4y_2y_1 - 56y_3 + 8y_3y_1 + 16y_3y_2).$$

Picking off coefficients term by term, we see that the result agrees with the independently calculated results of Part I, reproduced here as table 4. The rest of the ANODE procedure may be performed in the usual manner. Calculating the potential shift as

$$s = 2(0.5 \cdot 1 + 1.0 \cdot 2 + 0.5 \cdot 4 + 0.0) = 9.0,$$

we recognize that the function is strictly simple because the the potential shift of 9 is less than the difference between the highest and second-highest fitness values, $49 - 36 = 13$ (the assumption of a more poorly ordered string requires the analysis of only one additional point because the maximum potential shift $s = 2(4 + 2 + 1) = 14$ requires the inclusion of the top two points; such an analysis demonstrates strict simplicity regardless of ordering, because of the negative sign associated with all order-one w's).

j	$\mathbf{x}$	$f(\mathbf{x})$	w_j
0	000	1	17.5
1	001	2	-3.5
2	010	4	-7.0
3	011	9	1.0
4	100	16	-14.0
5	101	25	2.0
6	110	36	4.0
7	111	49	0.0

Table 3: Function values and Walsh coefficients of $f(d) = d^2$, d an unsigned binary integer.

5.3 A useful theorem

Algebraic reduction calculations can yield much information about a function's deception or lack thereof when we are willing to grind through the necessary symbolic manipulations. The growing availability of symbolic algebra packages such as Reduce, MACSYMA, and Mathematica will put these tools at many doorsteps; however, even when we are unwilling to do such calculations, algebraic analysis can yield useful information concerning the degree of bitwise nonlinearity — the degree of epistasis — inherent in a particular function. Here, we consider a straightforward theorem that bounds the order of the nonzero Walsh terms.

Assume that a fitness function f is a polynomial over a vector of decision variables $\mathbf{d}$, $f = f(\mathbf{d})$, and further assume that the degree of the f polynomial is k_f (the degree of a monomial is simply the sum of the exponents of the variables in the product — the degree of $d_1^2 d_2^3 d_4$ is 6 — and the degree of a polynomial is the maximum degree among all monomial terms). Furthermore, we assume that the components of the decision vector $\mathbf{d}$ may each be written as a polynomial over a subset of the Boolean string variables $\mathbf{x}$, where k_d is defined as the maximum degree among all the component functions. With these definitions, it is a straightforward matter to prove the following theorem.

Theorem 1. *If a function $f = f(\mathbf{d})$ is a polynomial of degree k_f in the components of the vector $\mathbf{d}$, and if the vector function $\mathbf{d} = \mathbf{d}(\mathbf{x})$ has component functions that are polynomials of maximum degree k_d in the string variables $\mathbf{x}$, the maximum order of any nonzero Walsh coefficient, $o^* = \max\{o(j) : w_j \neq 0\}$, is bounded as $o^* \leq k_f k_d$.*

Proof of the theorem follows immediately from elementary theorems in the algebraic theory of polynomials [7] and the fact that the auxiliary string mapping is of degree one.

Applying the theorem to the simple nonlinear function, we note that the maximum order o^* must be less than $2 \cdot 1 = 2$; this coincides with the observation that the coefficient $w_7 = 0$. Other applications of the theorem include

consideration of the epistasis of Gray and other nonlinear codes. The theorem seems to suggest that nonlinear codes may be more susceptible to deception, because of the potential for significant nonzero, higher-order Walsh coefficients, but such speculation must be checked before firm conclusions may be drawn, because high-order Walsh coefficients alone are insufficient evidence to prove deception.

6. Sensitivity analysis

Analysis of deception can be a powerful tool for understanding what makes coding-function combinations potentially difficult for genetic algorithms. Yet, once such an analysis has been performed on a particular function, there may be a variety of questions to ask regarding the change in the function's deception for given changes in the function's values. Furthermore, we might like to answer those questions without returning to square one on the original analysis. This may be easily accomplished using sensitivity analysis. In this section, we examine such a procedure and outline its use in constrained function analysis with penalty functions.

6.1 How much do the coefficients change when a given function value changes?

The crucial questions to ask and answer when pondering sensitivity analysis of deception are the following: when a particular function value is modified by some amount, which Walsh coefficients change and by what amount? These may be answered easily if we return to the definition of the Walsh transform obtained by orthogonality arguments in Part I:

$$w_j = \frac{1}{2^l} \sum_{i=0}^{2^l-1} f_i \psi_j(i). \tag{6.1}$$

The linearity of this equation requires that

$$\Delta w_j = \frac{1}{2^l} \sum_{i=0}^{2^l-1} \Delta f_i \psi_j(i). \tag{6.2}$$

Writing the total differential of Δw_j as

$$\Delta w_j = \sum_{i=0}^{2^l-1} \frac{\partial w_j}{\partial f_i} \Delta f_i, \tag{6.3}$$

the sensitivity coefficients $\frac{\partial w_j}{\partial f_i}$ may be recognized:

$$\frac{\partial w_j}{\partial f_i} = \frac{\psi_j(i)}{2^l}. \tag{6.4}$$

Since $\psi_j(i)$ is +1 or -1, every Walsh coefficient either increases or decreases by the amount of change in the particular function value being considered divided by 2^l. Of course, applying those changes to the w_j and then back-calculating f values is a glorious exercise in chasing one's tail; however, if the calculation of the Δw's is used in conjunction with the operator adjustment of a previous section, the sensitivity analysis provides a meaningful way to evaluate the sensitivity of deception to changes in function values.

6.2 Sensitivity analysis of deception

Because different operators affect different Walsh coefficients differently, changing a function value of a single point can affect the presence or absence of other points in the post-operator near-optimal set, N'. Defining the pre-operator changes (the sensitivity changes) as Δw_j and the post-operator sensitivity changes as $\Delta w_j''$, the following straightforward calculation follows immediately:

$$\Delta w_j'' = \Delta w_j \left[1 - p_c \frac{\delta(j)}{l-1} - 2p_m o(j) \right]. \tag{6.5}$$

The adjustment is performed to the sensitivity changes in the same manner as with the original w values.

Thereafter, the ANODE procedure may be adopted as before except that the potential shift should be calculated as

$$s = \sum_{j=0}^{2^l-1} |\Delta w_j' + \Delta w_j''|. \tag{6.6}$$

Once the enlarged set is redetermined (more or fewer members may be required), calculation of the post-operator, post-sensitivity function values may be determined as

$$f_i'' = f_i' + \Delta f_i'', \tag{6.7}$$

where the $\Delta f_i''$ values are summed over the post-operator $\Delta w_j''$ values in the usual Walsh way.

6.3 Constrained function optimization, penalty functions, and sensitivity analysis

One candidate application for analyzing deception using sensitivity analysis is function optimization with inequality constraints. The usual method for optimizing such problems with GAs [4] is to impose penalty functions on the unconstrained objective function, where the penalty is a linear or superlinear function of the constraint violation. We will not review this in detail, except to note that the use of a penalty method imposes changes on the unconstrained function at points exterior to the feasible region. The aggregate effect of these changes can be calculated using the sensitivity analysis just

developed, and deception can be determined by comparing the constrained near-optimal set to the constrained, post-operator near-optimal set. Careful application of sensitivity methods may then enable the use of penalty methods that permit near-optima to be found while introducing a minimum of additional deception into the problem.

7. Extensions

This work may be extended in a number of ways:

application of ANODE to operators other than simple crossover and mutation,

generalized Walsh analysis of standard codings,

the use of Walsh analysis and ANODE on functions with nonbinary codings,

string ordering design using algebraic methods, and

dynamic extensions of the ANODE analysis to determine whether a function-coding-operator combination diverges from a desired near-optimal set.

Analysis of deception may be performed for GAs that use operators other than simple crossover and mutation, as long as survival probability bounds can be estimated. Comparative deception analysis of different operators acting on the same function (or set of functions) might prove instructive. For example, it might be possible to compare single-point and multiple-point crossover operators using ANODE over a suite of difficult test functions.

Codings other than the usual unsigned binary integer may be handled using the methods of this paper. Gray codes, floating point codes, and other commonly used mappings may be analyzed. Algebraic methods or transform methods may be used on the coding functions individually, and the algebraic methods of section 5 may be applied to examine the interaction of function and coding. Comparative analysis of deception of different codings may prove useful here again. There are a number of theoretically unjustified claims regarding the superiority of this or that coding in the literature, and these may be subjected to careful scrutiny using the methods of this paper.

The Walsh analysis and analysis of deception may be extended to nonbinary alphabets. 2^k-ary alphabet analysis is particularly easy to visualize. Simply code the problem as the concatenation of l, length-k binary codes and perform the analysis as per usual except that $\delta(j)$ must be taken as the defining length of the substring written in the nonbinary alphabet. Careful analysis of mutation may be somewhat difficult, but approximate handling of various nonbinary mutation operators may not be too taxing.

Algebraic methods may prove useful in coding design. Knowledge of multiplicative or additive separability may be exploited immediately by placing those subsets of variables contained in different subfunctions on different

chromosomes, since linkage is unnecessary in such cases. Other prior knowledge of coding and function algebraic structure may be used to order strings to create short defining lengths for those building blocks that have the highest Walsh coefficients. In many cases, the required information may be available without algebraic computation of all Walsh coefficients.

Holland's [6] extension of the Walsh calculation to populations containing uneven proportions of strings opens the door to the possibility of complete dynamic analysis of GA-hardness (the word "hard" should be reserved for dynamic convergence to undesirable points, while "deceptive" should be saved for the static notion defined in this paper). The computational convenience of the Walsh basis may permit firmer conclusions than are usually possible when the full finite difference equations are written in terms of the canonical basis directly [2].

8. Conclusions

This paper has developed a number of tools for analyzing and designing the interaction of codings, functions, and operators within genetic algorithms. Specifically, the concepts of static deception and simplicity have been defined in terms of the overlap of two different near-optimal subsets, one defined in terms of fitness and the other in terms of operator-adjusted fitness. These definitions have led to the development of an analysis of deception or ANODE algorithm for calculating deception or simplicity in particular problems. In combination with the monomial representation of the Walsh functions developed in Part I, these tools have led to the development of algebraic coding-function analysis procedures, where bounding maximum-order calculations or full sets of Walsh coefficients may be obtained without resorting to expensive transform computations. A sensitivity analysis procedure has also been outlined to permit the direct evaluation of the change in deception that can occur when one or more of a function's values change; specific recommendations have also been made to apply this sensitivity analysis to problems with inequality constraints adjoined as a penalty.

These analysis and design tools should provide a useful analytical vantage point for understanding why genetic algorithms work and how they can be improved. Their future application should add a useful measure of rigor to the study of genetic algorithm performance.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant MSM-8451610. I also acknowledge research support provided by the Digital Equipment Corporation and by Texas Instruments Incorporated.

References

[1] A.D. Bethke, *Genetic algorithms as function optimizers* (Doctoral dissertation, University of Michigan, 1980) and *Dissertation Abstracts International*, **41(9)** (1980) 3503B (University Microfilms No. 8106101).

[2] C.L. Bridges and D.E. Goldberg, "An analysis of reproduction and crossover in a binary-coded genetic algorithm," *Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms* (1987) 9-13.

[3] D.E. Goldberg, "Simple genetic algorithms and the minimal, deceptive problem," In L. Davis, ed., *Genetic algorithms and simulated annealing* (Pitman, London, 1987) 74–88.

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

[5] J.H. Holland, *Adaptation in natural and artificial systems* (University of Michigan Press, Ann Arbor, 1975).

[6] J.H. Holland, "Searching nonlinear functions for high values," *Applied Mathematics and Computation*, in press.

[7] T.W. Hungerford, *Algebra*. (Springer-Verlag, New York, 1974).