What Makes a Problem Hard for a Genetic Algorithm? Some Anomalous Results and Their Explanation

Stephanie Forrest
Dept. of Computer Science
University of New Mexico
Albuquerque, N.M. 87131-1386
Email: forrest@unmvax.cs.unm.edu

Melanie Mitchell
Artificial Intelligence Laboratory
University of Michigan
Ann Arbor, MI 48109-2110
Email: melaniem@eecs.umich.edu

Machine Learning 13, 285-319, 1993.

Abstract

What makes a problem easy or hard for a genetic algorithm (GA)? This question has become increasingly important as people have tried to apply the GA to ever more diverse types of problems. Much previous work on this question has studied the relationship between GA performance and the structure of a given fitness function when it is is expressed as a *Walsh polynomial*. The work of Bethke, Goldberg, and others has produced certain theoretical results about this relationship. In this paper we review these theoretical results, and then discuss a number of seemingly anomalous experimental results reported by Tanese concerning the performance of the GA on a subclass of Walsh polynomials, some members of which were expected to be easy for the GA to optimize. Tanese found that the GA was poor at optimizing all functions in this subclass, that a partitioning of a single large population into a number of smaller independent populations seemed to improve performance, and that hillclimbing outperformed both the original and partitioned forms of the GA on these functions. These results seemed to contradict several commonly held expectations about GAs.

We begin by reviewing *schema processing* in GAs. We then give an informal description of how Walsh analysis and Bethke's Walsh-Schema transform relate to GA performance, and we discuss the relevance of this analysis for GA applications in optimization and machine learning. We then describe Tanese's surprising results, examine them experimentally and theoretically, and propose and evaluate some explanations. These explanations lead to a more fundamental question about GAs: what are the features of problems that determine the likelihood of successful GA performance?

1. Introduction

The genetic algorithm (GA) is a machine learning technique, originated by Holland (1975), loosely based on the principles of genetic variation and natural selection. GAs have become increasingly popular in recent years as a method for solving complex search problems in a large number of different disciplines. The appeal of GAs comes from their simplicity and elegance as algorithms as well as from their power to discover good solutions rapidly for difficult high-dimensional problems. In addition, GAs are idealized computational models of evolution that are being used to study questions in evolutionary biology and population genetics (Bergman and Feldman, 1990).

In the simplest form of the GA, bit strings play the role of chromosomes, with individual bits playing the role of genes. An initial population of individuals (bit strings) is generated randomly, and each individual receives a numerical evaluation which is then used to make multiple copies of higher-fitness individuals and to eliminate lower-fitness individuals. Genetic operators such as mutation (flipping individual bits) and crossover (exchanging substrings of two parents to obtain two offspring) are then applied probabilistically to the population to produce a new population, or *generation*, of individuals. The GA is considered to be successful if a population of highly fit individuals evolves as a result of iterating this procedure.

The GA has been used in many machine-learning contexts, such as evolving classification rules (e.g., Packard, 1990; De Jong and Spears, 1991), evolving neural networks (e.g., Miller, Todd, & Hegde, 1989; Whitley, Dominic, & Das, 1991), classifier systems (e.g., Holland, 1986; Smith, 1980), and automatic programming (e.g., Koza, 1990). In many of these cases there is no closed-form "fitness function"; the evaluation of each individual (or collection of individuals) is obtained by "running" it on the particular task being learned. The GA is considered to be successful if an individual, or collection of individuals, evolves that has satisfactorily learned the given task. The lack of a closed-form fitness function in these problems makes it difficult to study GA performance rigorously. Thus, much of the existing GA theory has been developed in the context of "function optimization" in which the GA is considered to be successful if it discovers a single bit string that represents a value yielding an optimum (or near optimum) of the given function. An introduction to GAs and GA theory is given by Goldberg (1989c), and many of the results concerning GA theory and applications can be found in the various ICGA proceedings (Grefenstette, 1985; Grefenstette, 1987; Schaffer, 1989; and Belew & Booker, 1991), in Davis (1987) and Davis (1991), and in the two previous special issues of *Machine Learning* (Goldberg & Holland, 1988a; De Jong, 1990a).

GAs have been successfully applied to many difficult problems but there have been some disappointing results as well. In cases both of success and failure there is often little detailed understanding of why the GA succeeded or failed. Given the increasing interest in applying the GA to an ever wider range of problems, it is essential for the theory of GAs to be more completely developed so that we can better understand how the GA works and when it will be most likely to succeed.

In this paper, we focus on a specific, initially surprising instance of GA failure that brings up several general issues related to this larger goal. We describe and explain a number of seemingly anomalous results obtained in a set of experiments performed by Tanese (1989a) using the GA to optimize a particular subset of Walsh polynomials. These polynomials were chosen because of their simplicity and the ease with which they can be varied, and because of the relationship between GAs and Walsh functions (see Section 3). They were expected to present a spectrum of difficulty for the GA. However, the results of the GA on these polynomials were not at all as expected: for example, the GA's performance was strikingly poor on all of the functions included in Tanese's experiments, and was significantly worse than the performance of a simple iterated hillclimbing algorithm. Our analysis of these apparently surprising results explains the failure of the GA on these specific functions, and it makes the following more general contributions: (1) We identify some previously ignored features of search spaces that can lead to GA failure (2) We demonstrate that there are a number of different, independent factors that contribute to the difficulty of search for a GA, and (3) concluding that any successful research effort into the theory of GA performance must take into account this multiplicity rather than concentrating on only one factor (e.g., deception; see Goldberg, 1989b; Liepins & Vose, 1990; Whitley, 1991; and Das & Whitley, 1991).

While this paper deals with learning real-valued functions on bit strings, the discussion and results presented here are relevant to researchers using GAs in other contexts as well, for a number of reasons. First, it is important for researchers using a particular machine-learning method (here, GAs) to know what analysis tools exist and what use they have. Schemas and Walsh analysis are such tools, and this paper provides a review of these aimed at readers with little or no previous knowledge of GA theory. Second, Tanese's results had seemingly surprising implications for the relative effectiveness of partitioned versus traditional GAs and for the relative effectiveness of hillclimbing versus either form of the GA. This paper explains these results and in so doing, illuminates the particular aspects of Tanese's applications problems that allowed these implications to hold. Finally, this paper makes a number of more general points about the GA related to De Jong's advice for using GAs in the context of machine learning:

> The key point in deciding whether or not to use genetic algorithms for a particular problem centers around the question: what is the space to be searched? If that space is well-understood and contains structure that can be exploited by special-purpose search techniques, the use of genetic algorithms is generally computationally less efficient. If the space to be searched is not so well understood and relatively unstructured, and *if an effective GA representation of that space can be developed*, then GAs provide a surprisingly powerful search heuristic for large, complex spaces (De Jong, 1990b, p. 351, italics added).

It is of central importance to understand what constitutes "an effective GA representation". We address this question by discussing some more general implications of Tanese's results for studying how the structure of a search space is related to the expected performance of the GA.

In the following sections we review schema processing in GAs, give an overview of the relationship between schema processing and Walsh polynomials, and describe how Walsh analysis has been used to characterize the difficulty of functions for the GA. We then describe the particular functions Tanese used in her experiments (the *Tanese functions*), and discuss and explain some of her anomalous results. Finally, we discuss the more general question of how to characterize problems in terms of the likelihood of successful GA performance (and the role of *GA-deception* in such characterizations). This paper presents details from experiments that were summarized in Forrest and Mitchell (1991).

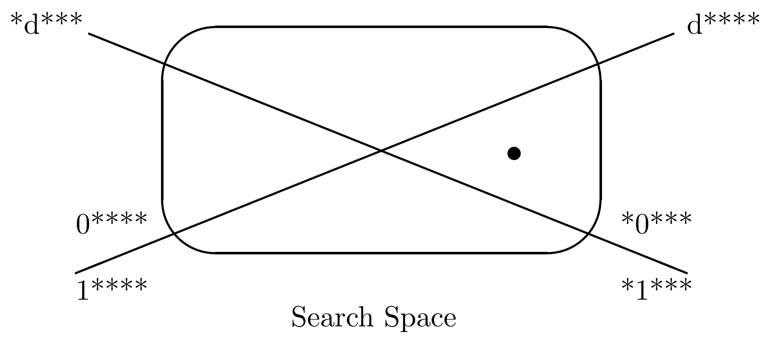


Figure 1: Schemas define hyperplanes in the search space.

2. Genetic Algorithms and Schema Processing

The notion of a *schema* is central to understanding how GAs work. Schemas are sets of individuals in the search space, and the GA is thought to work by directing the search towards schemas containing highly fit regions of the search space. The notion of schema processing is important in any GA application, but for the purpose of this paper, we will restrict our discussion to searches over bit strings. Much of the discussion given here also applies to more general representations.

In the case of bit strings, a schema can be expressed as a template, defined over the alphabet $\{0, 1, *\}$, that describes a pattern of bit strings in the search space $\{0, 1\}^l$, (the set of bit strings of length l). For each of the l bit-positions, the template either specifies the value at that position (1 or 0), or indicates by the symbol $*$ (referred to as *don't care*) that either value is allowed.

For example, consider the two strings A and B:

```
A = 100111
B = 010011.
```

There are eight schemas that describe the patterns these two strings have in common, including: ****11, **0***, **0**1, **0*11.

A bit string x that matches a schema s's pattern is said to be an *instance* of s (sometimes written as $x \epsilon s$); for example, 00 and 10 are both instances of $*0$. In schemas, 1's and 0's are referred to as *defined bits*; the *order* of a schema is simply the number of defined bits in that schema. The *defining length* of a schema is the distance between the leftmost and rightmost defined bits in the string. For example, the defining length of **0*11 is 3, and the defining length of **0*** is 0.

Schemas can be viewed as defining hyperplanes in the search space $\{0, 1\}^l$, as shown in Figure 1. Figure 1 shows four hyperplanes (corresponding to the schemas 0****, 1****, *0***, and *1***). Any point in the space is simultaneously an instance of two of these schemas. For example, the point in the figure is a member of both 1**** and *0*** (and also of 10***).

The fitness of any bit string in the population gives some information about the average fitness of the 2^l different schemas of which it is an instance (where l is the length of the string), so an explicit evaluation of a population of M individual strings is also an implicit evaluation of a much larger number of schemas. That is, at the explicit level the GA searches through populations of bit strings, but we can also view the GA's search as an *implicit* schema sampling process. At the implicit level, feedback from the fitness function, combined with selection and recombination, biases the sampling procedure over time away from those hyperplanes that give negative feedback (low average fitness) and towards those that give positive feedback (high average fitness).

According to the *building blocks* hypothesis (Holland, 1975; Goldberg, 1989c), the GA initially detects biases in low-order schemas (those with a small number of defined bits), and converges on this part of the search space. Over time, it detects biases in higher-order schemas by combining information from low-order schemas via crossover, and (if this process succeeds) it eventually converges on the part of the search space that is most fit. The building blocks hypothesis states that this process is the source of the GA's power as a search and optimization method. An important theoretical result about GAs is the Schema Theorem (Holland, 1975; Goldberg, 1989c), which guarantees that over time the observed best schemas will receive an exponentially increasing number of samples.

The GA therefore *exploits* biases that it finds by focusing its search more and more narrowly on instances of fit schemas, and it *explores* the search space using the heuristic that higher-order schemas with high average fitness are likely to be built out of lower-order schemas with high average fitness. Reproduction is the exploitation mechanism, and genetic operators—usually crossover and mutation—are the exploration mechanisms. (Holland proposes that mutation serves as a much less powerful exploration mechanism than crossover, effectively preventing information from being permanently lost from the population (Holland, 1975). According to the Schema Theorem, exploration predominates early on in a run of the GA, but over time, the GA converges more and more rapidly on what it has detected as the most fit schemas, and exploitation becomes the dominant mechanism.

This strong convergence property of the GA is a two-edged sword. On the one hand, the fact that the GA can identify the fittest part of the space very quickly is a powerful property; on the other hand, since the GA always operates on finite size populations, there is inherently some sampling error in the search, and in some cases the GA can magnify a small sampling error, causing *premature convergence* (Goldberg, 1989c). Also, in some cases strong convergence is inappropriate, for example, in classifier systems (Holland, 1986), in which the GA is trying to evolve a set of co-adapted rules, each one specialized for a specific but different task, rather than a population of similar rules.

As an example of the relevance of schemas to function optimization, consider the function shown in Figure 2. The function is defined over integers in the interval $[0, 31]$ (here, $l = 5$), so the x-axis represents the bit string argument (input to the function) and the y-axis shows the function's value, or fitness. In this example the x value will always be between 0 and 31. For example, the string 10000 would be interpreted as 16, and 01000 would be interpreted as 8.[1] Three schemas are displayed at the top of the plot: the schema 0**** (top dashed line) includes all points less than 16, the schema 1**** (bottom dashed line) includes all

[1]Most numerical optimization applications actually use a Gray-coded representation of numbers for the GA.

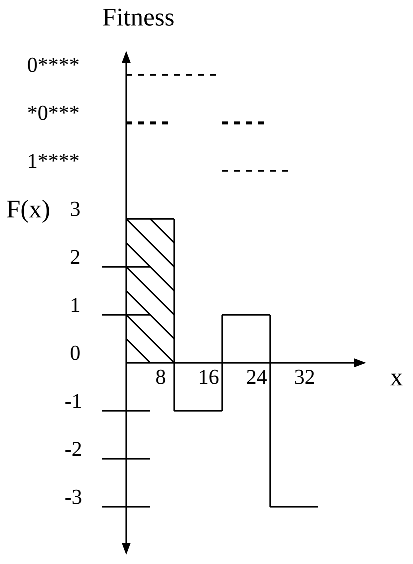


Figure 2: Example Function. The solid line indicates the function and the dashed lines indicate some schemas. The shaded region is the most fit region of the space.

points greater than or equal to 16, and the schema *0*** (middle dashed lines) specifies the intervals [0,8) and [16,24). Using this example it is easy to see how an individual that was an instance of the schema 0**** could be combined through crossover with an instance of the schema *0*** to yield an instance of 00***, which corresponds to the most fit region of the space (the shaded region in Figure 2). That is, 0**** and *0*** are partial solutions.

Schemas induce a partitioning of the search space (Holland, 1988). For example, as seen in Figure 1, the partition d**** (where "d" means "defined bit") divides the search space into two halves, corresponding to the schemas 1**** and 0****. That is, the notation d**** represents the partitioning that divides the space into two halves consisting of schemas with a single defined bit in the leftmost position. Similarly, the partition *d*** divides the search space into a different two halves, corresponding to the schemas *1*** and *0***. The partition dd*** represents a division of the space into four quarters, each of which corresponds to a schema with the leftmost two bits defined. Any partitioning of the search space can be written as a string in $\{d, *\}^l$, where the *order* of the partition is the number of defined bits (number of d's). Each partitioning of n defined bits contains 2^n *partition elements*; each partition element corresponds to a schema. Each different partitioning of the search space can be indexed by a unique bit string in which 1's correspond to the partition's defined bits and 0's correspond to the non-defined bits. For example, under this enumeration, the partition d***... * has index $j = 1000 \ldots 0$, and the partition dd***... * has index $j = 11000 \ldots 0$.

3. Walsh-Schema Analysis

Two goals for a theory of genetic algorithms are (1) to describe in detail how schemas are processed, and (2) to predict the degree to which a given problem will be easy or difficult

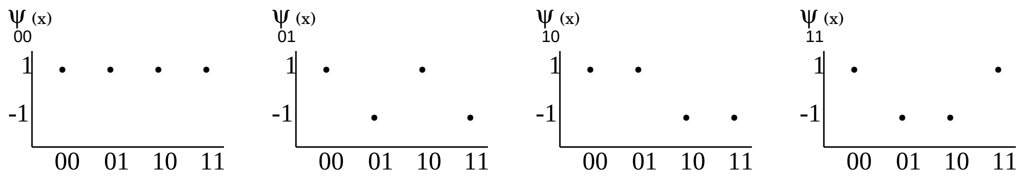


Figure 3: Plots of the four Walsh functions defined on two bits.

for the GA. Bethke's dissertation (1980) addressed these issues by applying Walsh functions (Walsh, 1923) to the study of schema processing in GAs. In particular, Bethke developed the *Walsh-Schema* transform, in which discrete versions of Walsh functions are used to calculate schema average fitnesses efficiently. He then used this transform to characterize functions as easy or hard for the GA to optimize. Bethke's work was further developed and explicated by Goldberg (1989a, 1989b). In this section we introduce Walsh functions and Walsh polynomials, review how the Walsh schema transform can be used to understand GAs, and sketch Bethke's use of this transform for characterizing different functions. Our discussion is similar to that given by Goldberg (1989a).

3.1 Walsh Functions, Walsh Decompositions, and Walsh Polynomials

Walsh functions are a complete orthogonal set of basis functions that induce transforms similar to Fourier transforms. However, Walsh functions differ from other bases (e.g., trigonometric functions or complex exponentials) in that they have only two values, $+1$ and -1. Bethke demonstrated how to use these basis functions to construct functions with varying degrees of difficulty for the GA. In order to do this, Bethke used a discrete version of Walsh's original continuous functions. These functions form an orthogonal basis for real-valued functions defined on $\{0,1\}^l$.

The discrete Walsh functions map bit strings x into {1, -1}. Each Walsh function is associated with a particular partitioning of the search space. The Walsh function corresponding to the j^{th} partition (where, as described above, the index j is a bit string) is defined as follows (Bethke, 1980; Tanese, 1989a):

$$\psi_j(x) = \begin{cases} 1 & \text{if } x \wedge j \text{ has even parity (i.e., an even number of 1's)} \\ -1 & \text{otherwise.} \end{cases}$$

Here, $\wedge$ stands for bitwise AND. Notice that $\psi_j(x)$ has the property that the only bits in x that contribute to its value are those that correspond to 1's in j.

Plots of the four Walsh functions defined on two bits are given in Figure 3.

Since the Walsh functions form a basis set, any function $F(x)$ defined on $\{0,1\}^l$ can be written as a linear combination of Walsh functions:

$$F(x) = \sum_{j=0}^{2^l-1} \omega_j \psi_j(x)$$

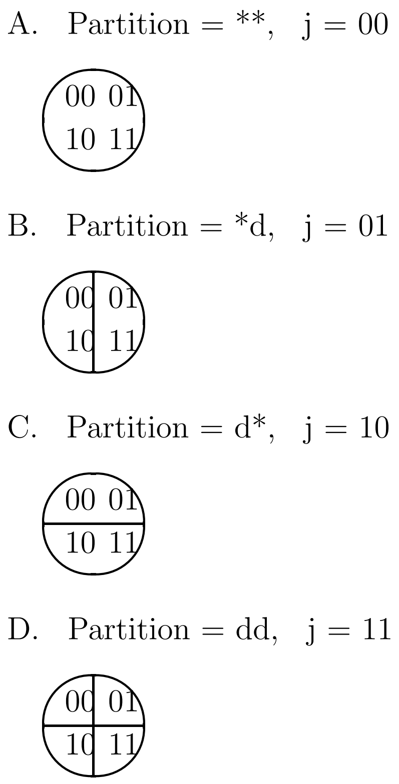

A. Partition = **, j = 00
00 01
10 11
B. Partition = *d, j = 01
00 01
10 11
C. Partition = d*, j = 10
00 01
10 11
D. Partition = dd, j = 11
00 01
10 11

Figure 4: Four different partitionings of the space of 2-bit strings.

where x is a bit string, l is its length, and each ω_j is a real-valued coefficient called a *Walsh coefficient.* For example, the function shown in Figure 2 can be written as

$$F(x) = 2\psi_{01000}(x) + \psi_{10000}(x).$$

The Walsh coefficients ω_j of a given function F can be obtained via the *Walsh transform*, which is similar to a Fourier transform. The representation of a function F in terms of a set of Walsh coefficients ω_j is called F's *Walsh decomposition* or *Walsh polynomial.* In this and the next subsection, we will explain how the Walsh transform works and discuss the close relationship between Walsh analysis and schemas.

As a simple example of the Walsh transform, consider the function $F(x) = x^2$, where x is a 2-bit string. The space of 2-bit strings can be partitioned into sets of schemas in four different ways, as illustrated in Figure 4.

The Walsh transform works by transforming $F(x)$ into the summed series of Walsh terms $F(x) = \sum_{j=0}^{2^l-1} \omega_j \psi_j(x)$, in which increasingly longer partial sums provide progressively better estimates of the value of $F(x)$. The terms in the sum are obtained from the average values of F in progressively smaller partition elements.

In this example we will use Walsh analysis to get better and better estimates for $F(11)$ $(= 9)$.

Consider first the average value of F on the entire space, which is the same as the average fitness $u(**)$ of the schema ** in the partition $j = 00$ (part A of Figure 4):

$$u(**) = \overline{F} = (F(00) + F(01) + F(10) + F(11))/4 = 14/4.$$

Let $\omega_{00} = u(**) = \overline{F}$. This could be said to be a "zeroth order" estimate of $F(11)$ (or of $F(x)$ for any x).

Now to get a better estimate for $F(11)$, some corrections need to be made to the zeroth order estimate. One way to do this is to look at the average value of F in a smaller partition element containing $F(11)$—say, *1 (the right-hand element shown in part B of Figure 4). The average value of the schema *1 is

$$u(*1) = \omega_{00} - \mathit{deviation}_{*1};$$

that is, it is equal to the average of the entire space minus the deviation of $u(*1)$ from the global average. Likewise, the average value of the complement schema *0 is

$$u(*0) = \omega_{00} + \mathit{deviation}_{*1},$$

since $u(*1) + u(*0) = 2u(**) = 2\omega_{00}$. (The assignment of $+$ or $-$ to $\mathit{deviation}_{*1}$ here is arbitrary; it could have been reversed.) The magnitude of the deviation is the same for both schemas (*1 and *0) in partition *d. Call this magnitude ω_{01}. A better estimate for F(11) is then $\omega_{00} - \omega_{01}$.

The same thing can be done for the other order-1 schema containing 11: 1*. Let ω_{10} be the deviation of the average value in d* from the global average. Then,

$$u(1*) = \omega_{00} - \omega_{10}.$$

An even better estimate for $F(11)$ is $\omega_{00} - \omega_{01} - \omega_{10}$. This is a first-order estimate (based on 1-bit schemas). The two deviation terms are independent of each other, since they correct for differences in average values of schemas defined on different bits, so we subtract them both. If the function were linear, this would give the exact value of $F(11)$. (In some sense, this is what it *means* for a function defined on bit strings to be linear.)

However, since $F(x) = x^2$ is nonlinear, one additional correction needs to be made to account for the difference between this estimate and the average of the order-2 schema, the string 11 itself:

$$F(11) = \omega_{00} - \omega_{01} - \omega_{10} + \mathit{correction}_{11}.$$

The magnitude of the order-2 correction term is the same for each $F(x)$. This can be shown as follows. We know that

$$F(11) = \omega_{00} - \omega_{01} - \omega_{10} + \mathit{correction}_{11},$$

and by a similar analysis,

$$F(10) = \omega_{00} + \omega_{01} - \omega_{10} + \mathit{correction}_{10}.$$

Adding both sides of these two equations, we get,

$$F(11) + F(10) = 2\omega_{00} - 2\omega_{10} + \mathit{correction}_{11} + \mathit{correction}_{10}.$$

$$\boxed{\begin{array}{l} F(00) = \omega_{00} + \omega_{01} + \omega_{10} + \omega_{11}. \\ F(01) = \omega_{00} - \omega_{01} + \omega_{10} - \omega_{11}. \\ F(10) = \omega_{00} + \omega_{01} - \omega_{10} - \omega_{11}. \\ F(11) = \omega_{00} - \omega_{01} - \omega_{10} + \omega_{11}. \end{array}}$$

Table 1: Expressions for $F(x)$ for each $x \epsilon \{0,1\}^2$

But $F(11) + F(10) = 2u(1*)$ (by definition of $u(1*)$), so we have:

$$F(11) + F(10) = 2u(1*) = 2\omega_{00} - 2\omega_{10},$$

since, as was discussed above, $u(1*) = \omega_{00} - \omega_{10}$. Thus, $\mathit{correction}_{11} = -\mathit{correction}_{10}$.

Similarly,

$$F(01) = \omega_{00} - \omega_{01} + \omega_{10} + \mathit{correction}_{01},$$

so,

$$F(11) + F(01) = 2\omega_{00} - 2\omega_{01} + \mathit{correction}_{11} + \mathit{correction}_{01},$$

and since

$$F(11) + F(01) = 2u(*1) = 2\omega_{00} - 2\omega_{01},$$

we have $\mathit{correction}_{11} = -\mathit{correction}_{01}$.

Finally,

$$F(00) = \omega_{00} + \omega_{01} + \omega_{10} + \mathit{correction}_{00},$$

so,

$$F(00) + F(01) = 2\omega_{00} + 2\omega_{10} + \mathit{correction}_{11} + \mathit{correction}_{01},$$

and since

$$F(00) + F(01) = 2u(0*) = 2\omega_{00} + 2\omega_{10},$$

we have $\mathit{correction}_{00} = -\mathit{correction}_{01}$. Thus the magnitudes of the second-order correction terms are all equal. Call this common magnitude ω_{11}.

We have shown that, for this simple function, each partition j' has a single $\omega_{j'}$ associated with it, representing the deviation of the real average fitness of each schema in partition j' from the estimates given by the combinations of lower-order ω_j's. The magnitude of this deviation is the same for all schemas in partition j'. This was easy to see for the first-order partitions, and we showed that it is also true for the second-order partitions (the highest order partitions in our simple function). In general, for any partition j, the average fitnesses of schemas are mutually constrained in ways similar to those shown above, and the uniqueness of ω_j can be similarly demonstrated for j's of any order.

Table 1 gives the exact Walsh decomposition for each $F(x)$.

We have now shown how function values can be calculated in terms of Walsh coefficients, which represent progressively finer correction terms to lower-order estimates in terms of schema averages. A converse analysis demonstrates how the ω_j's are calculated:

$$\begin{aligned}
\omega_{00} &= u(**) \\
&= (0+1+4+9)/4 \\
&= 14/4. \\
\omega_{01} &= \omega_{00} - u(*1) \\
&= (0+1+4+9)/4 - (1+9)/2 \\
&= (0-1+4-9)/4 \\
&= -6/4. \\
\omega_{10} &= \omega_{00} - u(1*) \\
&= (0+1+4+9)/4 - (4+9)/2 \\
&= (0+1-4-9)/4 \\
&= -12/4. \\
\omega_{11} &= F(11) - \text{first-order estimate} \\
&= F(11) - (\omega_{00} - \omega_{01} - \omega_{10}) \\
&= 9 - (14/4 + 6/4 + 12/4) \\
&= 4/4.
\end{aligned}$$

And to check:

$$F(11) = \omega_{00} - \omega_{01} - \omega_{10} + \omega_{11} = 14/4 + 6/4 + 12/4 + 4/4 = 9.$$

In general,

$$\omega_j = \frac{1}{2^l} \sum_{x=0}^{2^l-1} F(x)\psi_j(x).$$

This is the *Walsh transform* (it is derived more formally in Goldberg, 1989a). Once the ω_j's have been determined, F can be calculated as

$$F(x) = \sum_{j=0}^{2^l-1} \omega_j \psi_j(x).$$

This expression is called the *Walsh polynomial* representing $F(x)$.

How does one decide whether or not a deviation term ω_j is added or subtracted in this expression? The answer to this question depends on some conventions: e.g., whether $u(*1)$ is said to be $\omega_{00} - \omega_{01}$ or $\omega_{00} + \omega_{01}$. Once these conventions are decided, they impose constraints on whether higher-order Walsh coefficients will be added or subtracted in the expression for $F(x)$. If x happens to be a member of a schema s whose average deviates in a positive way from the lower-order estimate, then the positive value of the ω_j corresponding to s's partition goes into the sum. All that is needed is a consistent way of assigning these signs, depending on the partition j and what element of j a given bit string x is in. The purpose of the Walsh functions $\psi_j(x)$ is to provide such a consistent way of assigning signs to ω_j's, via bitwise AND and parity. This is not the only possible method; a slightly different method is given by Holland for his *hyperplane transform* (Holland, 1988)—an alternative formulation of the Walsh-schema transform, described in the next section.

3.2 The Walsh-Schema Transform

There is a close connection between the Walsh transform and schemas. The Walsh-Schema transform formalizes this connection. In the previous section, we showed that, using Walsh coefficients, we can calculate a function's value on a given argument x using the average fitnesses of schemas of which that x is an instance. An analogous method, proposed by Bethke (1980), can be used to calculate the average fitness $u(s)$ of a given schema s, e.g., *1. Bethke called this method the *Walsh-schema transform.* This transform gives some insight into how schema processing is thought to occur in the GA. It also allowed Bethke to state some conditions under which a function will be easy for the GA to optimize, and allowed him to construct functions that are difficult for the GA because low-order schemas lead the search in the wrong direction.

Formal derivations of the Walsh-schema transform are given by Bethke (1980), Goldberg (1989a), and Tanese (1989a). Here we present the transform informally.

Using the same example as before, the average fitness of the schema *1 is $u(*1) = \omega_{00} - \omega_{01}$; this comes from the definition of ω_{01}. The value of $u(*1)$ does not depend on, say, ω_{10}; it depends only on Walsh coefficients of partitions that either contain *1 or contain a superset of *1 (e.g., $** \supset *1$). In general, a partition j is said to *subsume* a schema s if it contains some schema s' such that $s' \supseteq s$. For example, the 3-bit schema 10* is subsumed by four partitions: dd*, d**, *d*, and ***, which correspond respectively to the j values 110, 100, 010, and 000. Notice that j subsumes s if and only if each defined bit in j (i.e., each 1) corresponds to a defined bit in s (i.e., a 0 or a 1, not a *).

The Walsh-schema transform expresses the fitness of a schema s as a sum of progressively higher-order Walsh coefficients ω_j, analogous to the expression of $F(x)$ as a sum of progressively higher-order ω_j's. Just as each ω_j in the expression for $F(x)$ is a correction term for the average fitness of some schema in partition j containing x, each ω_j in the expression for $u(s)$ is a correction term, correcting the estimate given by some lower-order schema that contains s. The difference is that for $F(x)$, all 2^l partition coefficients must be summed (although some of them may be zero). But to calculate $u(s)$, only coefficients of the subsuming partitions ("subsuming coefficients") need to be summed.

The example 2-bit function given above was too simple to illustrate these ideas, but an extension to three bits suffices. Let $F(x) = x^2$, but let x be defined over three bits instead of two. The average fitness of the schema *01 is a sum of the coefficients of partitions that contain the schemas ***, **1, *0*, and *01. An easy way to determine the sign of a subsuming coefficient ω_j is to take any instance of s, and to compute $\psi_j(x)$. This value will be the same for all $x \epsilon s$, as long as j is a subsuming partition, since all the ones in j are matched with the same bits in any instance of s. For example, the partition **d ($j = 001$) subsumes the schema *11, and $\psi_{001}(x) = -1$ for any $x \epsilon * 11$. Using a similar method to obtain the signs of the other coefficients, we get

$$u(*11) = \omega_{000} - \omega_{001} - \omega_{010} + \omega_{011}.$$

In general,

$$u(s) = \sum_{j:j \ subsumes \ s} \omega_j \Psi_j(s).$$

where $\Psi_j(s)$ is the value of $\psi_j(x)$ ($= +1$ or -1) for any $x \epsilon s$.

The sum

$$u(*11) = \omega_{000} - \omega_{001} - \omega_{010} + \omega_{011}$$

gives the flavor of how the GA actually goes about estimating $u(*11)$. To review, a population of strings in a GA can be thought of as a number of samples of various schemas, and the GA works by using the fitness of the strings in the population to estimate the fitness of schemas. It exploits fit schemas via reproduction by allocating more samples to them, and it explores new schemas via crossover by combining fit low-order schemas to sample higher-order schemas that will hopefully also be fit. In general there are many more instances of low-order schemas in a given population than high-order schemas (e.g., in a randomly generated population, about half the strings will be instances of 1**...*, but very few, if any will be instances of 111...1). Thus, accurate fitness estimates will be obtained much earlier for low-order schemas than for high-order schemas. The GA's estimate of a given schema s can be thought of as a process of gradual refinement, where the algorithm initially bases its estimate on information about the low-order schemas containing s, and gradually refines this estimate from information about higher and higher order schemas containing s. Likewise, the terms in the sum above represent increasing refinements to the estimate of how good the schema *11 is. The term ω_{000} gives the population average (corresponding to the average fitness of the schema ***) and the increasingly higher-order ω_j's in the sum represent higher-order refinements of the estimate of *11's fitness, where the refinements are obtained by summing ω_j's corresponding to higher and higher order partitions j containing *11.

Thus, one way of describing the GA's operation on a fitness function F is that it makes progressively deeper estimates of what F's Walsh coefficients are, and biases the search towards partitions j with high-magnitude ω_j's, and to the partition elements (schemas) for which these correction terms are positive.

3.3 The Walsh-Schema Transform and GA-Deceptive Functions

Bethke (1980) used Walsh analysis to partially characterize functions that will be easy for the GA to optimize. This characterization comes from two facts about the average fitness of a schema s. First, since $u(s)$ depends only on ω_j's for which j subsumes s, then if the order of j (i.e., the number of 1's in j) exceeds the order of s (i.e., the number of defined bits in s), then ω_j does not affect $u(s)$. For example, ω_{111} does not affect $u(*11)$: *11's two instances 011 and 111 receive opposite-sign contributions from ω_{111}. Second, if the defining length of j (i.e., the distance between the leftmost and rightmost 1's in j) is greater than the defining length of s (i.e., the distance between the leftmost and rightmost defined bits in

s), then $u(s)$ does not depend on ω_j. For example, ω_{101} does not affect $u(*11)$; again, since $u(*11)$'s two instances receive opposite-sign contributions from ω_{101}.

Bethke suggested that if the Walsh coefficients of a function decrease rapidly with increasing order and defining length of the j's—that is, the most important coefficients are associated with short, low-order partitions—then the function will be easy for the GA to optimize. In such cases, the location of the global optimum can be determined from the estimated average fitness of low-order, low-defining-length schemas. As we described above, such schemas receive many more samples than higher-order, longer-defining-length schemas: "low order" means that they define larger subsets of the search space and "short defining length" means that they tend to be kept intact under crossover. Thus the GA can estimate their average fitnesses more quickly than those of higher-order, longer-defining-length schemas.

Thus, all else being equal, a function whose Walsh decomposition involves high-order j's with significant coefficients should be harder for the GA to optimize than a function with only lower-order j's, since the GA will have a harder time constructing good estimates of the higher-order schemas belonging to the higher-order partitions j.

Bethke's analysis was not intended as a practical tool for use in deciding whether a given problem will be hard or easy for the GA. As was mentioned earlier, the fitness functions used in many GA applications are not of a form that can be easily expressed as a Walsh polynomial. And even if a function F can be so expressed, a Walsh transform of F requires evaluating F at every point in its argument space (this is also true for the "Fast Walsh Transform", Goldberg, 1989a), and is thus an infeasible operation for most fitness functions of interest. It is much more efficient to run the GA on a given function and measure its performance directly than to decompose the function into Walsh coefficients and then determine from those coefficients the likelihood of GA success. However, Walsh analysis can be used as a theoretical tool for understanding the types of properties that can make a problem hard for the GA. For example, Bethke used the Walsh-schema transform to construct functions that mislead the GA, by directly assigning the values of Walsh coefficients in such a way that the average values of low-order schemas give misleading information about the average values of higher-order refinements of those schemas (i.e., higher-order schemas contained by the lower-order schemas). Specifically, Bethke chose coefficients so that some short, low-order schemas had relatively low average fitness, and then chose other coefficients so as to make these low-fitness schemas actually contain the global optimum. Such functions were later termed "deceptive" by Goldberg (1987, 1989b, 1991), who carried out a number of theoretical studies of such functions. Deception has since been a central focus of theoretical work on GAs. Walsh analysis can be used to construct problems with different degrees and types of deception, and the GA's performance on these problems can be studied empirically. The goal of such research is to learn how deception affects GA performance (and thus why the GA might fail in certain cases), and to learn how to improve the GA or the problem's representation in order to improve performance.

In Section 9, we further discuss deception and its relation to the goal of characterizing functions as hard or easy for the GA.

4. Tanese Functions

Bethke's method of creating functions by directly assigning the values of Walsh coefficients permitted the straightforward construction of functions that present different degrees of difficulty to the GA. Tanese (1989a, 1989b) also used this method in her dissertation, in which she presented a comparative study of the performance of a number of variants of the GA. In particular, this study compared the variants' performance in optimizing particular classes of Walsh polynomials that were constructed to present various levels of difficulty to the GA[2].

Tanese compared the behavior of the "traditional" GA with a "partitioned" GA, in which the single large population of the traditional GA was subdivided into a number of smaller subpopulations that independently worked to optimize a given objective function. Tanese also studied a partitioned GA in which individuals in various subpopulations were allowed to "migrate" to other subpopulations during a run (she called this GA "distributed").

For her experiments comparing the performance of GA variants Tanese wanted to be able to generate automatically a number of classes of fitness functions, with different classes having different levels of difficulty, but with each function in a given class having similar levels of difficulty. She generated different classes of Walsh polynomials as follows. The functions were defined over bit strings of length 32. Each fitness function F was generated by randomly choosing 32 j's, *all of the same order* (e.g., 4). Tanese generated functions only of even order for her experiments. The coefficient ω_j for each of the 32 chosen j's was also chosen randomly from the interval $(0, 5]$. The fitness function consisted of the sum of these 32 terms. Once the 32 j's were chosen, a point x' was chosen randomly to be the global optimum, and the sign of each non-zero ω_j's was adjusted so that the fitness of x' would be $\sum |\omega_j|$. The *order* of F is defined as the common order of the j's in its terms.

For example,

$$F(x) = \psi_{11110}(x) + 2\psi_{11101}(x) - 3\psi_{11011}(x)$$

is an order-4 function on 5-bits (rather than 32) with three terms (rather than 32), with global optimum 6. This Walsh polynomial, like those used by Tanese, has two special properties: (1) all of the non-zero terms in the polynomial are of the same order (Tanese actually used only even-order functions); and (2) there exists a global optimum x' whose fitness receives a positive contribution from each term in the polynomial. Functions with these two properties will hereafter be called *Tanese functions*.

This method of constructing functions had several advantages: (1) it was easy to construct random functions of similar difficulty, since functions of the same order were thought to be of roughly the same difficulty for the GA; (2) functions of different degrees of difficulty could be constructed by varying the order, since low-order functions of this sort should be easier for the GA to optimize than high-order functions; and (3) the global optimum was known, which made it possible to to measure the GA's absolute performance.

The results of Tanese's experiments were surprising, and seem to contradict several common expectations about GAs. Specifically, Tanese's results show that for *every* Tanese

[2]In her dissertation, Tanese describes experiments involving several fitness functions. In this paper we consider only the results with respect to Walsh polynomials.

function she studied—including the low-order ones—the GA performed poorly, and that its performance was often improved when the total population was split up into very small subpopulations. Moreover, Tanese found that the performance of a simple iterated hillclimbing algorithm was significantly better on these functions than both the traditional and partitioned forms of the GA. These results apparently contradict Bethke's analysis which predicts that the low-order Tanese functions should be relatively easy for the GA to optimize. These results also apparently contradict some other beliefs about the GA—that it will routinely outperform hillclimbing and other gradient descent methods on hard problems such as those with nonlinear interactions (Holland, 1988); and that a population must be of a sufficient size to support effective schema processing (Goldberg, 1985; Goldberg, 1989d). In order to better understand the sources of Tanese's results, we performed a number of additional experiments, which are described in the following sections.

5. Experimental Setup

The experiments we report in this paper were performed with a similar GA and identical parameter values to those used by Tanese (1989a) (and also in previous work by Forrest, 1985). All of Tanese's experiments used strings of length 32 and populations of 256 individuals. The population was sometimes subdivided into a number of smaller subpopulations. Tanese's algorithm used a *sigma scaling* method, in which the number of expected offspring allocated to each individual is a function of the individual's fitness, the mean fitness of the population, and the standard deviation from the mean. The number of expected offspring given to individual x was: $\frac{F(x)-\overline{F}}{2\sigma}+1$, where $\overline{F}$ is the population mean and σ is the standard deviation. Thus an individual of fitness one standard deviation above the population mean was allocated one and a half expected offspring (there was a cutoff at a maximum of five expected offspring). Multipoint crossover was used, with a crossover rate of 0.022 per bit (e.g., for 32-bit strings, there were on average 0.7 crossovers per pair of parents). The crossover rate (per pair) was interpreted as the mean of a Poisson distribution from which the actual number of crosses was determined for each individual. The mutation probability was 0.005 per bit. With the exceptions of sigma scaling and multipoint crossover, Tanese's GA was conventional (proportional selection, complete replacement of the population on every time step, and no creative operators besides crossover and mutation). Tanese ran each of her experiments for 500 generations. For some of our experiments we altered certain parameter values; these exceptions will be noted explicitly.

Tanese conducted experiments on Walsh polynomials of orders 4, 8, 16, and 20. For each experiment she randomly generated 64 functions of a given order, and compared the performance of the *traditional* (single population) GA with a number of *partitioned* GAs, in which the population of 256 individuals was subdivided into various numbers of smaller subpopulations. In this paper, we discuss results only for functions of order 8, since these were the functions Tanese analyzed in the greatest depth. All of our experiments involved manipulations of parameters in the traditional GA; we did not do any experiments with partitioned GAs. For each of our experiments, we ran the GA once on each of 20 different randomly generated functions, for 200 generations each. Tanese carried each run out to 500 generations, but in each of our runs that used strings of length 32, the population had converged by about generation 100, and the results would not have changed significantly if the run had been extended. The shorter runs were sufficient for determining the comparative effects of the various manipulations we performed on the parameters.

Tanese also compared her GA results with the results of running an iterated hillclimbing algorithm on randomly generated Tanese functions. The iterated hillclimbing algorithm works as follows (Tanese, 1989a).

Repeat the following until a specified number of function evaluations have been performed.

1. Choose a random string x in the search space, and calculate its fitness.
2. Calculate the fitness of every one-bit mutation of x. If an improvement over x's fitness is found, set x equal to the string with the highest fitness.
3. Repeat step 2 until no one-bit mutation yields an improvement. Save this "hilltop" point.
4. Go to step 1.

Finally, return the highest hilltop.

In order to compare the performance of a run of iterated hillclimbing with that of a run of the GA, Tanese ran the above algorithm until the number of function evaluations was equal to that performed by the GA (the population size times the number of generations).

6. An Examination of Tanese's Results

One of Tanese's most striking results was the poor performance of the GA (in both its traditional and partitioned-population form) on the functions described above and the superior performance of hillclimbing to both forms of the GA.

The results of our replication of some of Tanese's original experiments comparing the traditional GA and hillclimbing are summarized in the first and sixth rows of Table 2, under "Original" and "Hill 32" (hillclimbing with 32-bit strings). Our replications confirm Tanese's findings that, on order-8 functions, neither the GA nor hillclimbing discovers the optimum, but that hillclimbing performs significantly better than the GA in terms of the average maximum fitness found. On average, hillclimbing was able to find a string whose fitness was within about 5% of the maximum, whereas the original GA was able to find a string whose fitness was only within about 12% of the maximum.

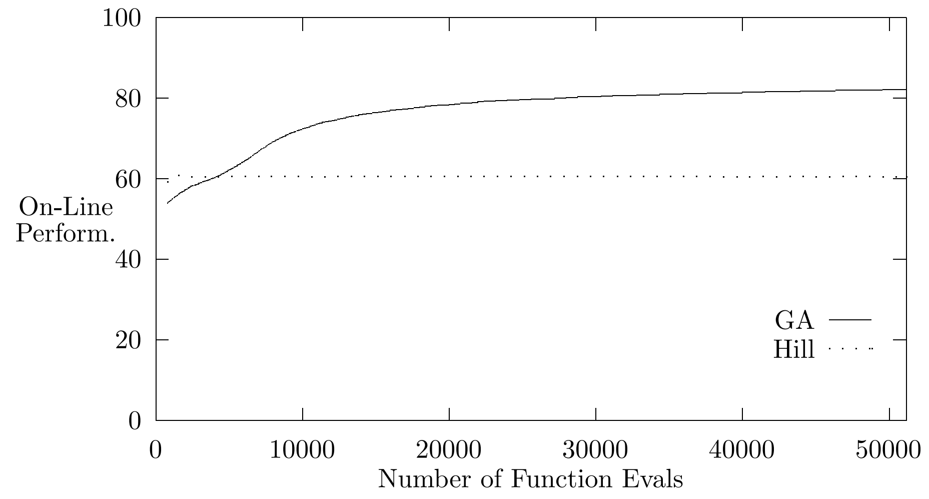
In the following subsections, four possible explanations for the GA's poor performance are discussed and evaluated: (1) the choice of performance criteria gave an unfair measure of the GA's performance, (2) crossover is ineffective on these functions because of the lack of lower-order building blocks; (3) the average defining-lengths of the j's are very long and thus good schemas tend to be broken up by crossover; and (4) the random generation of 32 j's over strings of length 32 results in a large number of correlated positions among the j's, effectively making the functions very difficult.

6.1 Performance Criteria

Tanese used three main performance measures in her experiments, each associated with an average over 64 "trials" (five runs each on 64 randomly generated functions). These measures were: (1) the best overall fitness (averaged over 64 trials); (2) the average fitness in the final generation (averaged over 64 trials); and (3) the best fitness in the final generation (averaged over 64 trials). She found that for the first and last measures, the performance of the partitioned GA was generally superior to that of the traditional GA; these results were generally reversed for the "average final fitness" measure. She also used a summary measure called "success rate": the number of trials on which the global optimum was found at least once out of five times. On the 64 trials (i.e., 320 runs total) on randomly generated order-4 Walsh polynomials, the success rate of the traditional GA was only 3. The most successful partitioned algorithm's success rate was only 15, and the success rate of hillclimbing was 23—almost eight times the success rate of the traditional GA and more than one and a half times the success rate of the best partitioned GA. On 320 runs on randomly generated order-8 Walsh polynomials, neither the traditional nor the various partitioned GAs (nor hillclimbing) ever found the optimum, and hillclimbing consistently reached higher fitness values than the GA did.

We examined the possibility that the seemingly poor performance of the GA was due to the strictness of the performance measures used by Tanese. To do this, we compared the GA and hillclimbing using two alternative performance measures that are well-known in the GA literature: De Jong's *on-line* and *off-line* performance measures (De Jong, 1975; Goldberg, 1989c). These measures are made in terms of the number of function evaluations an algorithm has performed up to a certain time. The on-line performance at time t is the average value of all fitnesses that have been evaluated by the algorithm up to time t. The off-line performance at time t is the average value, over t evaluation steps, of the best fitness that has been seen up to each evaluation step. For example, if, at $t = 5$, five strings have been evaluated yielding fitnesses 10, 20, 8, 4, and 15, the on-line performance would be $\frac{10+20+8+4+15}{5}$, and the off-line performance would be $\frac{10+20+20+20+20}{5}$.

In Figure 5, the on-line and off-line performance values are plotted over 51200 function evaluations (the equivalent of 200 generations of the GA with a population of 256) for a typical run of both the GA and hillclimbing running on the same order-8 Tanese function. The y-axes give the performance in terms of percent of the function optimum (0–100%). The on-line and off-line measures were computed every 800 function evaluations. As can be seen, the GA performs significantly better than hillclimbing on on-line performance, but worse on off-line performance (very similar results are obtained on other runs). The on-line result is to be expected: hillclimbing spends most of its time searching through low-fitness single-bit mutations of high-fitness strings, only moving when it finds a higher-fitness mutation. The offline result is closer to traditional optimization measures, and the better performance of hillclimbing on this measure is what is unexpected. In the next several sections we will propose and evaluate possible explanations for this result, along with Tanese's other results.



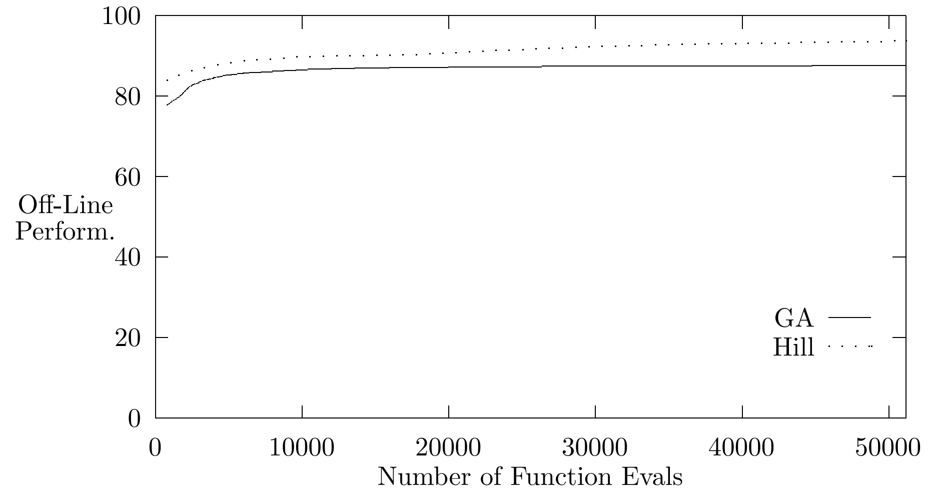


Figure 5: Plots displaying on-line and off-line performance on a typical run of the GA and of hillclimbing on an order-8 Tanese function. The y-axes give the performance in terms of percent of the function optimum (0–100%).

6.2 Is crossover effective on these functions?

As was discussed in Section 2, the GA's power is thought to be due to its *implicit* search of schemas in the search space, in which more and more samples are given to fitter schemas, and fit lower-order schemas are combined to produce hopefully fitter higher-order schemas. Thus, to understand the performance of the GA on a fitness function F, we must ask what types of schemas will yield useful information to the GA and to what extent these schemas can be usefully combined by crossover.

The fitness of a string x under a Tanese function F depends on the parity of $x \wedge j$ for each j in the terms of F. Recall that the only terms in F that contribute to a schema s's average fitness are those whose j's subsume s, where j subsumes s implies that each 1 in j corresponds to a defined bit in s. For example, $s = *011$ is subsumed by $j = 0111$ (the index for the partition *ddd) but not $j = 1011$ (the index for the partition d*dd).

To understand this better, consider a Tanese function with only one order-2 term (with $l = 4$):

$$F(x) = 2.0\psi_{1100}(x).$$

A schema s with only one defined bit (order 1) gives no information to the GA about this function, since half the instances of s will yield -2.0 and half will yield $+2.0$. This can be easily checked, for example, with the schema 1***. Likewise, the schema 0*00, which is also not subsumed by $j = 1100$, has the same property. In general, if a term $\omega_j\psi_j(x)$ does not subsume s (i.e., if some 1 in j corresponds to a * in s), then half of s's instances will yield $-\omega_j$ and half will yield $+\omega_j$ for that term, so if a schema s is not subsumed by any terms, then its average fitness is zero. An order-n j cannot subsume a schema of less than order n, so this means that for a Tanese function of order n, no schema of order less than n will give the GA any useful information about what parts of the search space are likely to be more fit. Thus crossover, one of the major strengths of the GA, is not a useful tool for recombining building blocks until schemas that are subsumed by terms in the fitness function have been discovered—such schemas must be of order at least as high as that of the function. This means that the power of crossover—as a means of recombining fit lower-order building blocks—is severely curtailed in these functions.

To verify this empirically, we ran the GA without crossover (i.e., crossover-rate = 0) on 20 randomly generated 32-bit order-8 functions. These results (along with results from all experiments described in this paper) are summarized in Table 2; they are summarized under the heading *No Xover 32*, and are to be compared with the values under *Original*, giving the results from our replication of Tanese's traditional GA runs on order-8 functions. The GA's performance was not impaired; the maximum fitness discovered and the mean population fitness are not significantly different from the runs in which crossover was used (*Original* in the table).

6.3 Is the GA's poor performance due to long defining lengths of schemas?

In the Tanese functions, an order-n partition index j was constructed by randomly placing n 1's in a string of 32 bits. The expected defining length for such a j can be calculated using Eq. (4) from Goldberg, Korb, and Deb (1990) (p. 5): $\frac{<\delta>}{l+1} = \frac{n-1}{n+1}$ where $< \delta >$ is the expected defining length for a string with n 1's in an l-bit string. Thus, the expected defining length for an order-4 Tanese function (with strings of length 32) is approximately 20, and for an order-8 function it is approximately 26, a substantial proportion of the entire string. This last estimate corresponds closely to the empirical data presented in Tanese's thesis (Tansese, 1989a, Table 5.1) for one example order-8 Walsh polynomial.

As we pointed out earlier, the only useful schemas are ones that are subsumed by one or more of the terms in the fitness function. The calculation given above implies that such schemas will, like the j's in the function's terms, tend to have long defining lengths. As was discussed in Section 3.3, long defining lengths of j's (and thus of useful schemas) can make a function hard for the GA because of the high probability of crossover disruption of useful schemas, since the longer the defining length of a schema, the more likely it is to be broken up under crossover (Holland, 1975; Goldberg, 1989c). To what degree was this problem responsible for the poor performance of the GA on these functions?

To answer this question, we ran the traditional GA with Tanese's parameters on 20 randomly generated order-8 functions, in which the j's were restricted to have a maximum defining length of 10. That is, for each of the 32 j's,, a randomly positioned window of 10 contiguous loci was chosen, and all eight 1's were placed randomly inside this window.

The results are summarized in Table 2 under *Def-Len 10*. Using the success-rate criterion described above, the performance was identical to Tanese's original results: the traditional GA never found the optimum. Other performance measures made it clear that limiting the defining length improved the GA's performance slightly: the GA was able to find strings with slightly higher fitnesses (in terms of percent of optimum) and slightly higher mean-population fitnesses. We conclude that the contribution of long defining lengths to the GA's overall poor performance on these functions is not significant.

6.4 Is the GA's poor performance due to overlap among significant loci in the partition indices?

Next, we considered the possibility that overlaps among defined positions in the j's (i.e., loci with 1's) were causing correlations among terms in a given fitness function, making the optimization problem effectively higher-order. As a simple example of this, suppose that 0011 and 0110 are two order-2 j's that have non-zero positive coefficients. These j's *overlap* at bit position 2 since they both have value 1 at that locus. There are eight strings x that will cause $\psi_{0011}(x)$ to be positive, corresponding to the schemas: **00 and **11. Likewise, there are eight strings that will cause $\psi_{0110}(x)$ to be positive, corresponding to the schemas: *00* and *11*. So, in order to find a point that gets a positive value from both $\psi_{0110}(x)$ and $\psi_{0011}(x)$, the GA must discover either the schema *000 or the schema *111. This overlap has the effect that three bits are effectively correlated instead of two, making the problem effectively order-3 instead of order-2. In the case of the Tanese functions, this is a likely source of difficulty; for example, with order-8 functions, where 32 order-8 terms were randomly generated, each 1 in any given j will on average be a member of 7 other different j's, and thus the effective linkage will be exceedingly high.

To assess the effect of overlaps, we ran the GA on functions in which the strings were

	Times Optimum Found	Average Max Fitness (% opt.)	Average Generation of Max Fitness	Average Max Mean Fitness (% opt.)
Original	0	88.1 (2.9)	31 (33)	85.1 (3.8)
No Xover 32	0	88.4 (2.7)	22 (28)	86.2 (2.6)
Def-Len 10	0	92.3 (2.9)	41 (48)	89.2 (3.0)
Str-Len 128	19	99.97 (0.13)	150 (30)	93.6 (1.3)
No Xover 128	17	99.85 (0.45)	72 (41)	93.9 (0.6)
Hill 32	0	95.4 (1.5)	-	-
Hill 128	20	100.0 (0.0)	-	-

Table 2: Summary of results of all experiments. The experiments were all performed running the traditional GA (and in two cases, hillclimbing) on randomly generated order-8 Walsh polynomials. Each result summarizes 20 runs of 200 generations each. The experiments were: (1) *Original* (replicating Tanese's experiments); (2) *No Xover 32* (same as *Original* but with no crossover); (3) *Def-Len 10* (limiting the defining length of partition indices to 10); (4) *Str-Len 128* (increasing the string length to 128 bits); (5) *No Xover 128* (same as *Str-Len 128* but with no crossover); (6) *Hill 32* (hillclimbing on 32 bits); and (7) *Hill 128* (hillclimbing on 128 bits) All runs except the 128-bit runs used strings of length 32. The values given are (1) the number of times the optimum was found; (2) the maximum fitness (percent of optimum) found (averaged over 20 runs); (3) the average generation at which the maximum fitness was first found; and (4) the maximum population mean (% of optimum) during a run (averaged over 20 runs). The numbers in parentheses are the standard deviations.

of length 128 rather than 32 (but still order 8), in order to reduce the number of overlaps. With a string length of 128, each defined locus with a 1 would participate on average in only 2 of the 32 j's. As shown in Table 2 (under *Str-Len 128*), the GA's performance was remarkably improved. The GA found the optimum 19/20 times (compared with 0/20 for the 32-bit case), and came very close to finding the optimum on the other run. In addition, the mean fitnesses of the population were substantially higher than they were on the 32-bit functions. Of all the experiments we tried, this caused the most dramatic improvement, leading us to conclude that the principle reason the Tanese functions are difficult is because the short strings (32 bits) and relatively high number of terms (32) causes nearly all 32 bits to be correlated, thus creating an order-32 problem. In the non-overlapping case, the order of the problem is much lower, and it is possible for the GA to optimize each term of the function almost independently.

To verify further the ineffectiveness of crossover on the Tanese functions, we ran the GA without crossover on 20 randomly generated 128-bit order-8 functions. The results are summarized under *No Xover 128* in Table 2: the performance of the GA was not significantly different from the 128-bit runs in which crossover was used (*Str Len 128*). With both the shorter and longer string-lengths (and thus with both large and small amounts of overlap), whether or not crossover is used does not make a significant difference in the GA's performance on these functions.

The fact that overlap is much higher with 32-bit functions than with 128-bit functions explains the strikingly different dynamics between runs of the GA on the former and latter functions, as shown in Figures 6–11. These figures are explained below.

A Walsh polynomial F can be thought of as a "constraint satisfaction" problem in which each term in F is a particular constraint. The goal is to find an individual x that satisfies (i.e., receives positive values from) as many terms as possible, and preferably from terms with high coefficients. In a typical run on 32-bit strings, the GA quickly finds its highest-fit individual, typically by generation 30 or so (see the "Average Gen. of Max. Fitness" column in Table 2). Figure 6 plots the percentage of the optimum of the maximum fitness in the population versus time for one typical run. As can be seen, the GA found its highest-fit individual by about generation 40, and this individual's fitness was about 90% of the optimum.

Given an individual x in the population, each term $\omega_j \psi_j(x)$ in the fitness function contributes either a positive or negative value to x's total fitness. If $\omega_j \psi_j(x)$ contributes a positive value, we call x a *positive instance* of that term; otherwise we call x a *negative instance.* Figure 7 shows, for this same run, how the percentages in the population of positive and negative instances of each term vary over time. The y-axis consists of the 32 partition coefficients ω_j of the fitness function, in order of ascending magnitude (recall that these coefficients were each chosen randomly in a range from 0.0 to 5.0, and then their signs were adjusted). The x-axis consists of generation numbers, from 0 to 200. At any point in the plot, the value of the gray-scale represents the percentage of individuals x in the population that are positive instances of the given term at the given time. The darker the gray value, the higher the percentage. As can be seen, very early on (after generation 10), a large subset of the terms have 80–100% as positive instances, and a smaller subset of terms have only a small number of positive instances.

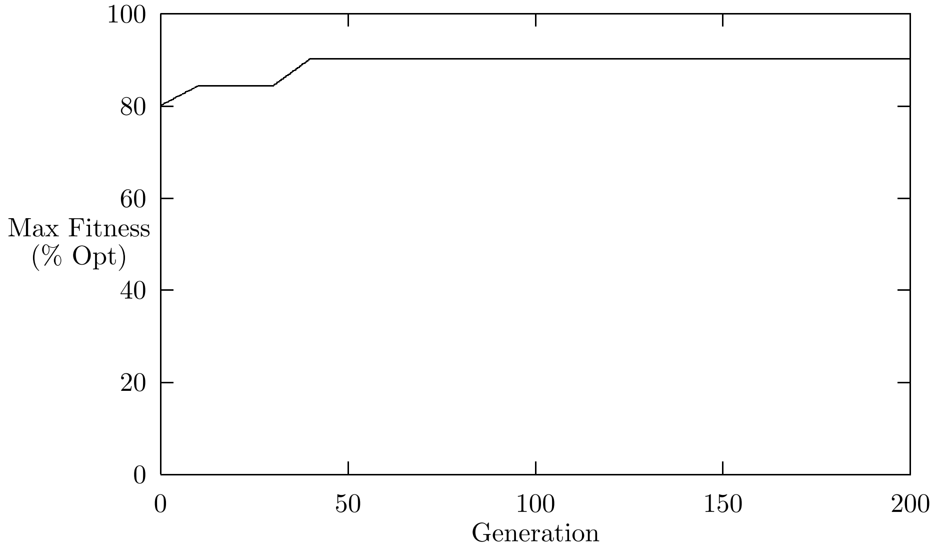


Figure 6: The percentage of the optimum of the maximum fitness plotted over time for a typical run with string length 32. For this plot, the maximum fitness was sampled every 10 generations.

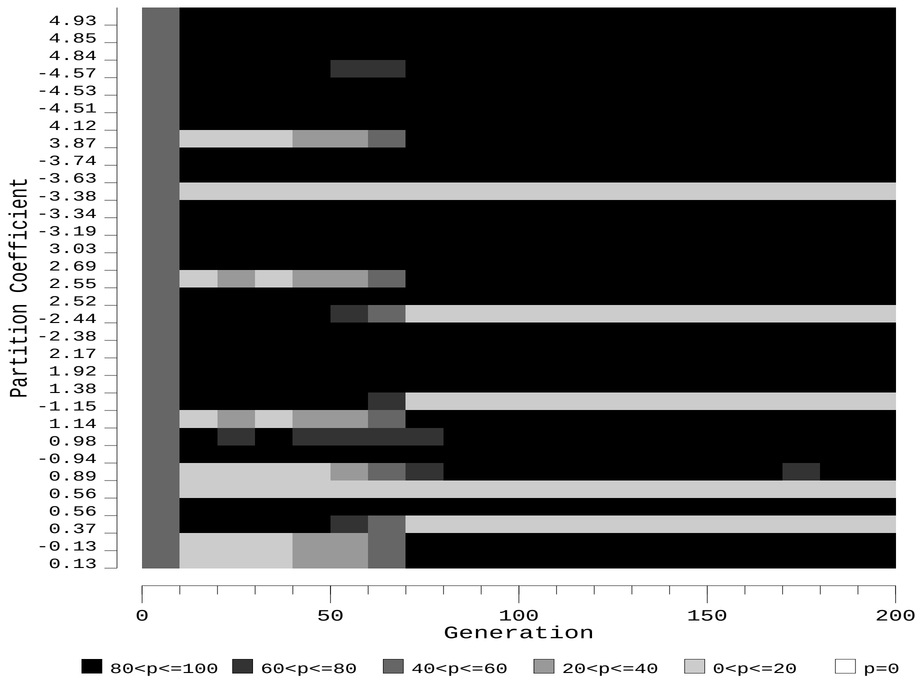


Figure 7: The percent of positive instances for each term in the fitness function plotted over time for the same run with string length 32. The coefficients on the y-axis are in order of increasing magnitude. The darker the gray-value, the higher the percentage of positive instances. The values plotted here were sampled every 10 generations.

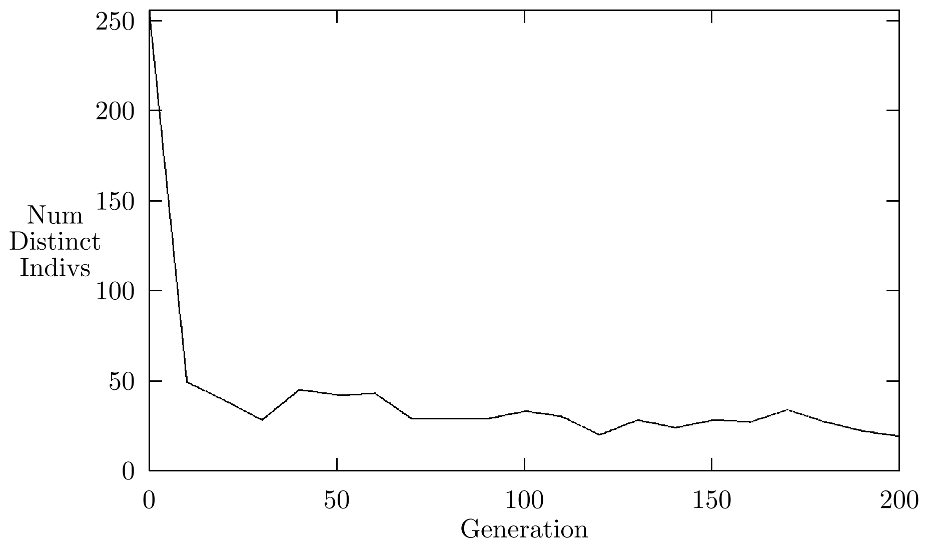


Figure 8: The number of distinct individuals in the population plotted over time for the same run with string length 32. For this plot, the number of distinct individuals was sampled every 10 generations.

This result can be explained in terms of the constraint-satisfaction problem. Any individual x with suboptimal fitness receives positive values from only a subset of the terms in the fitness function, leaving the rest of the terms "unsatisfied", that is, yielding negative values on that individual. However, because of the high degree of overlap, the constraint satisfaction problem here is very severe, and to discover an individual that receives additional positive values from other terms —without losing too many of the currently held positive values—is very difficult, especially since crossover does not help very much on these functions (see Section 6.2). This is particularly true since, once individuals of relatively high fitness are discovered, the diversity of the population falls very quickly. This can be seen in Figure 8, which plots the number of distinct individuals in the population over time[3]. As Figure 7 indicates, the population very quickly subdivides itself into a small number of mutually exclusive sets: one large set of individuals receiving positive values from one subset of the terms in the fitness function, and other, much smaller sets of individuals receiving positive values from the other terms in the fitness function.

To summarize, the GA stays stuck at a local optimum that is discovered early. It is possible that raising crossover and mutation rates (or using other techniques to prevent premature convergence; see Goldberg, 1989c) might improve the GA's performance on such functions, since it would increase the amount of diversity in the population.

On a typical run involving strings of length 128, the situation is very different. Figures 9–

[3]We counted two individuals as being distinct only if they were different at some significant locus — that is, at a locus in which one of the 32 j's had value 1 (otherwise the two individuals would be equivalent as far as the fitness function was concerned). As one would expect, with strings of length 32, every locus was significant, which is not generally the case for strings of length 128.

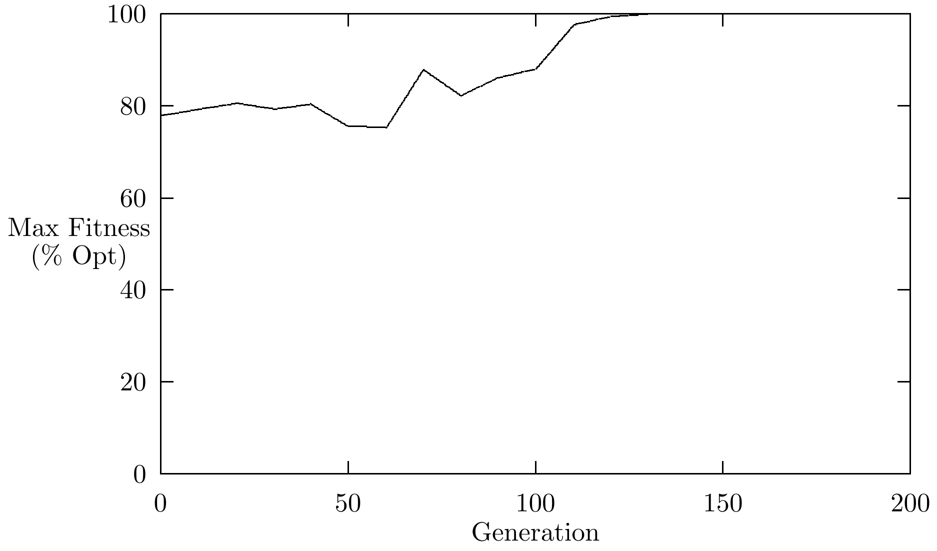


Figure 9: The percentage of the optimum of the maximum fitness plotted over time for a typical run with string length 128. For this plot, the maximum fitness was sampled every 10 generations.

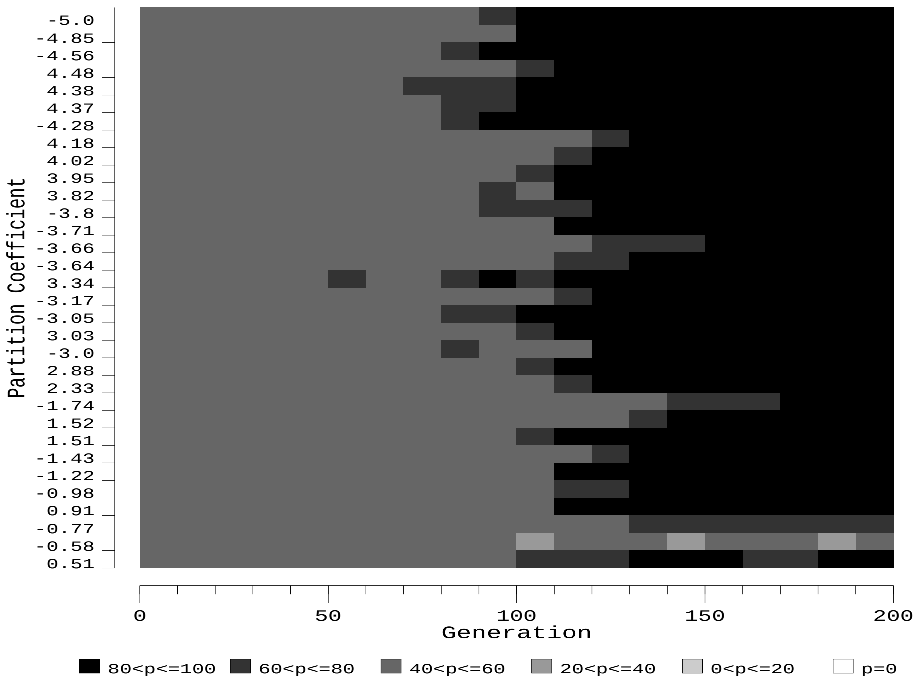


Figure 10: The percent of positive instances for each term in the fitness function plotted over time for the same run with string length 128. The coefficients on the y-axis are in order of increasing magnitude. The darker the gray-value, the higher the percentage of positive instances. The values plotted here were sampled every 10 generations.

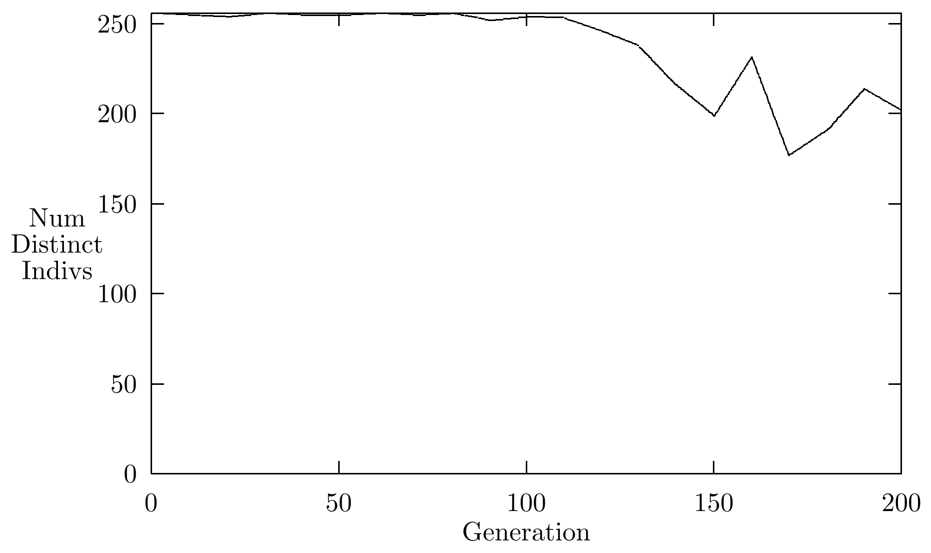


Figure 11: The number of distinct individuals in the population plotted over time for the same run with string length 128. For this plot, the number of distinct individuals were sampled every 10 generations.

11 plot the same quantities for one such run. As can be seen in Figure 9, the GA tends not to discover its highest-fit individual until late in the run (on average, around generation 150), and, as can be seen in Figure 11, the diversity of the population remains relatively high throughout the run. The continuing high diversity of the population is a result of several factors, including the following: since the problem is less constrained, there are many ways in which an individual can achieve relatively high fitness; and since the crossover rate was defined on a per-bit basis, there are on average a greater number of crossovers per chromosome here than in the 32-bit case. But the relative lack of constraints was the major factor, since when crossover was turned off in the 128-bit case, the population diversity remained considerably higher than for the 32-bit run described above, although it was less than in the 128-bit case with crossover turned on. As can be seen in Figure 10, in the 128-bit case the population does not segregate into mutually exclusive sets—throughout a typical run, almost all of the terms in the fitness function provide positive values to a significant fraction of the population.

Because of the relative lack of constraints in the 128-bit case, the population does not quickly become locked into a local optimum, and is able both to continue exploring longer than in the 32-bit case and to optimize each term of the fitness function more independently.

In short, these results indicate that the major cause of the GA's difficulty on Tanese's original functions is the high-degree of overlap among significant loci in the j's, creating problems of high effective order. A secondary cause of difficulty is the lack of information from lower-order schemas, making crossover relatively ineffective on these functions.

Note that these two factors—overlaps and lack of information from lower-order schemas—

do not imply that the Tanese functions are deceptive; a function can have these features and still not be deceptive. The Tanese functions thus provide an important counter-example to a prevailing belief that deception is the crucial feature in characterizing what types of functions are difficult for the GA to optimize (Das & Whitley, 1991). The notion of deception and its relation to the Tanese functions and to GA performance in general is discussed further in Section 9.

7. Why does subdividing the population seem to improve performance?

Recall that Tanese compared the behavior of the "traditional" GA with a "partitioned" GA, in which the single large population of the traditional GA was subdivided into a number of smaller subpopulations that independently worked to optimize the given function. Although both the traditional and partitioned GAs performed poorly on the Tanese functions, the results reported by Tanese showed that the performance of most instances of the partitioned GA was better than that of the traditional GA. This was the case even when the population was subdivided into extremely small subpopulations, such as 64 subpopulations of 4 individuals each, and, in some cases, even for 128 subpopulations of 2 individuals each. For functions of order higher than 4, neither the traditional nor partitioned GA's ever found the function optimum, so the difference between the two was measured only in terms of proximity to the optimum of the best fitness discovered.

As Tanese points out, these results run against conventional wisdom about GAs: it has been thought that on difficult problems a large population is needed for processing a sufficient number of schemas (Goldberg, 1985).

Tanese proposes three main reasons for this surprising result.

1. Tanese functions have a large number of local optima and the GA tends to converge on one. Each of the smaller subpopulations of the partitioned GA will converge earlier than a larger single population, but the subpopulations tend to converge on different optima, and so are able to explore a greater number.
2. After the populations converge, the major adaptive force is mutation. Mutation will be more effective in a smaller population than in a large population, since in a large population, the greater number of mutations will drive up the variance of fitnesses in the population, making it less likely that fit mutations will be able to produce enough offspring to spread in the population (recall that the expected number of offspring is a function of the fitness and the standard deviation). In the smaller population, fewer mutations will occur and the variance will be lower, allowing fit mutations to spread more effectively.
3. Since smaller populations tend to be more homogeneous (for the reasons given above), fit mutations are less likely to be destroyed through crossover.

Explanation 1 seems correct, especially in light of the dynamics that were discussed in Section 6.4. It seems that the traditional GA quickly finds a single local optimum that satisfies a subset of terms in the Walsh polynomial, and cannot go beyond that point. Since the number of different chromosomes in the population falls so quickly, the traditional GA does not use the potential for diversity that its large population offers. This explains why a number of small populations running in parallel would likely result in at least one way of satisfying the constraints that was superior to local optimum found by a single large-population GA.

Explanation 2 relies on the assumption that large populations tend to be more diverse and have higher standard deviations than smaller ones. But as was seen in Figure 8, the diversity of the large population with 32-bit strings falls very quickly, so this assumption seems incorrect. We believe that the main factor keeping successful mutations at bay is the degree to which the problem is constrained: once a local optimum is settled upon, it is very unlikely that a single mutation (or a small set of mutations) will yield a fitter individual.

A similar point could be made with respect to Explanation 3. Given the homogeneity that results even with a large population, it seems likely that this is not a significant factor in the relative poor performance of the traditional GA.

It is important to point out that the effects discussed here come about because of the special characteristics of the fitness functions being used—in particular, the fact that the single-order functions prevent the GA from exploiting information from lower-order building blocks through crossover. For functions in which lower-order building blocks do yield useful information, these effects might not be seen. The results and analysis for these functions cannot be assumed to be applicable to all instances of Walsh polynomials.

8. Why does hillclimbing outperform the GA on these functions?

The discussions in the previous sections allow us to understand why hillclimbing outperforms the GA on the Tanese functions. Since the nature of the Tanese functions precluded the GA from using information from lower-order building blocks via crossover, the genetic operators of crossover and mutation served mainly as a means of generating random variation—in effect, an inefficient form of hillclimbing. This is the reason for the greater success of hill-climbing on these functions. This is shown further by the results of running hillclimbing on functions with 128-bit strings, summarized under *Hill 128* in Table 2. On these easier functions (with less overlap than the 32-bit string functions), hillclimbing still beat the GA, finding the optimum 20 out of 20 times versus 17 out of 20 times for the GA.

9. Are the Tanese Functions Deceptive?

None of the explanations given so far for Tanese's anomalous results relies on the Tanese functions being deceptive. However, the poor performance of the GA led Goldberg to propose that "deception or partial deception" was lurking in the Tanese functions (Goldberg, 1991). In this section we examine the question of whether or not it is possible for a Tanese functions to be deceptive. This examination has led to some general concerns about what the term "deception" actually means or should mean, and what its role is in understanding GAs.

9.1 Definitions of Deception

A number of different definitions of deception as well as types of deception have been proposed in the GA literature (e.g., see Bethke, 1980; Goldberg, 1987; Liepins & Vose, 1990; and Whitley, 1991). Although there is no generally accepted definition of the term "deception", it is generally agreed that the notion of deception involves situations in which lower-order schemas give misleading information to the GA about the probable average fitness of higher-order refinements (higher-order schemas contained by the lower-order schemas). For the purpose of this section we will use the definitions of different types of deception proposed by Whitley (1991). These definitions are summarized below. These definitions all use the notion of competition among schemas in a partition. For example, in a 4-bit problem, the order-2 partition *dd* contains four schemas: *00*, *01*, *10*, and *11*. The *winner* of this "hyperplane competition at order-2" (or, equivalently, the *winner of this partition*) is the schema with the highest average fitness. The partition *dd* is said to *subsume* the partition *ddd, which contains eight schemas, each a subset of one of the schemas in *dd*. This notion of a partition subsuming another partition is analogous to the notion of a partition subsuming a schema.

The following definitions are adopted from Whitley (1991) (which, as is stated in that paper, are consistent with most other proposed definitions of deception in the literature).

Definition: A problem *contains deception* if the winner of some partition has a bit pattern that is different from the bit pattern of the winner of a higher-order subsumed partition.

Definition: A problem is *fully deceptive* at order N if, given an order-N partition P, the winners of *all* subsuming lower-order partitions lead toward a *deceptive attractor*, a hyperplane in P that is different from the winner of P.

Definition: A problem is *consistently deceptive* at order N if, given a order-N partition P, the various winners of all subsuming lower-order partitions lead toward hyperplanes that are different from the winner of P (though not necessarily leading towards the same deceptive attractor).

It should be noted that, strictly speaking, deception is a property of a particular *representation* of a problem rather than of the problem itself. In principle, a deceptive representation could be transformed into a non-deceptive one, but in practice it is usually an intractable problem to find the appropriate transformation.

9.2 Tanese Functions and Deception

The first question we want to address is, could the functions that Tanese used possibly be deceptive? As it turns out, the answer is not clear, since the definitions given above neglect a key aspect of these functions: every partition of the search space contains at least two winning (maximal) schemas. This is because for j's of even order, $\psi_j(x) = \psi_j(\overline{x})$. But the definitions given above all specify that there is a single winner ("the winner") of each partition.

The definition of *contains deception* could be modified as follows:

Definition: A problem *contains deception* if *one of the winners* of some hyperplane competition has a bit pattern that is different from the bit pattern of *each of the winners* of a higher-order subsumed partition.

If this definition were adopted, then it would be possible for a even-order Tanese function to contain deception. The following is a simple example of such a function (where the bit strings are of length 5 and the order of the j's is 4):

$$F(x) = \psi_{11110}(x) + 2\psi_{11101}(x) - 3\psi_{11011}(x).$$

$F(x)$ meets the two requirements of a Tanese function: all the non-zero terms are of the same order (4), and there is a point x' whose fitness receives a positive contribution from each term in $F(x)$. One such point is $x' = 10100$, with $F(x') = 6$.

To show that this function contains deception according to the modified definition, consider the partition $P =$ dddd*. One winner of P is the schema $s_0 =$ 1111*. This can be seen as follows. As was discussed in Section 6.2, the only terms in F that contribute to a schema s's average fitness are those whose j's *subsume* s, where j *subsumes* s implies that each 1 in j corresponds to a defined bit in s. For example, s_0 is subsumed only by the first term of F. A term $\omega_j\psi_j(x)$ that does not subsume s will contribute 0 to s's average fitness, since, because a Walsh function's value depends on parity, half the instances of s will make this term positive and half will make it negative. This can be easily checked for s_0 with respect to the second two terms of F. The average fitness of s_0 is 1, which is the highest possible average fitness for schemas in P, since schemas in P are subsumed only by the first term of F. Thus, s_0 is a winner in P.

However, consider the subsumed partition $P' =$ ddddd. According to the modified definition, if the problem does not contain deception, then either 11111 or 11110 should be a winner in P'. But the fitness of 11111 is 0, and the fitness of 11110 is 2, whereas the fitness of $x' = 10100$ (a winner in P') is 6. Thus, this function contains deception according to the modified definition.

It can be proved, however, that for any Tanese function, every partition contains *some* winner that does not lead the GA astray. This is stated formally by the following theorem:

Theorem 1: Let P be any partition of the search space defined by a Tanese function F. Then there exists a schema s in P such that s is a winner in P and, for any P' subsumed by P, there exists a winning schema s' in P' such that $s' \subset s$.

Proof:

Let x' be a global optimum of F. Let s be the schema in P whose defined bits are the same as the corresponding bits in x'. The average fitness of s depends only on the terms in

F that subsume s. Call the set of such terms SUBSUMES(s). Since x' is a global optimum of a Tanese function, it receives positive values from all terms in F, so it receives positive values from all terms in SUBSUMES(s). But the value of a term $\omega_j\psi_j(x)$ in SUBSUMES(s) on x' depends only on the bits in x' corresponding to 1's in j. Since SUBSUMES(s) is the set of terms that subsume s, these 1's appear only in positions corresponding to defined bits in s (which are the same as x's defined bits), so every instance of s also receives positive values from all the terms in SUBSUMES(s). Thus s is a winner in P. (Note that if SUBSUMES(s) is empty, then all schemas in P are winners, since they all have average fitness 0.)

Now let P' be a higher-order subsumed partition of P. Let s' be the schema in P' for which $s' \subset s$, and for which the additional defined bits in s' are set so that $x' \subseteq s'$. Then, for the same reason as above, s' is a winner in P'.

We have shown that (1) there can be a winning schema in a given partition that does not contain any winning schema in a higher-order subsumed partition, but (2) there is always at least one winning schema in every partition that contains a winning schema in every higher-order subsumed partition (and thus contains a global optimum).

It is unclear from Whitley's definitions whether or not this situation should be called "deception". Theorem 1 implies that the Tanese functions cannot be fully or consistently deceptive, but it seems to be a matter of definition whether or not they can contain deception. Certainly the property illustrated by the example "deceptive" function given above might cause difficulty for the GA, but the degree of difficulty might be different from a function containing deception that strictly conforms to Whitley's definition. It is interesting to note that Wilson (1991) defines a problem to be "GA-easy" if "the highest fitness schema in *every* complete set of competing schemata contains the optimum." Theorem 1 could be interpreted to imply that the Tanese functions are GA-easy, since there is *a* highest-fitness schema in each partition that contains the optimum. Of course, as Tanese demonstrated, many Tanese functions were far from easy for the GA.

As with the notion of "GA-easy", most definitions of deception assume that there will always be a single winner in every partition. However, it cannot be expected that for every problem posed to the GA, every partition in the search space will contain a unique winning schema. Thus it is of considerable importance, in formulating a definition of deception, to take into account situations such as the multiple partition-winners in the Tanese functions. While the Tanese functions are a particularly extreme example of this, multi-modality of this kind certainly exists in other more natural functions, and we need to be clear about what deception means in these circumstances. Definitions of deception should at least take into account the difference between the kind of deception found in the example function given above, and the kind specified by Whitley's original definition. If they are both to be called "deception," then it is clear that deception is a broad concept with a number of dimensions, and these various dimensions should be identified.

Note that the example of a "deceptive" Tanese function given above is quite different from the functions that Tanese actually used. It may be that the differences (e.g., 32 terms instead of 3 terms, 32-bit strings instead of 5-bit strings, higher-order j's) place additional constraints on the functions that affect whether or not they can be deceptive. We have shown only that it is *possible* to construct a Tanese function (obeying the two criteria stated earlier) that contains "deception." It is still an open question whether or not the original Tanese functions actually contain the type of deception illustrated in the example. And again it is important to note that the two major factors we have identified as responsible for the GA's difficulty on the Tanese functions—overlaps and lack of information from lower-order schemas—are distinct from the notion of deception. We believe it is these features, and not deceptiveness, that makes Tanese functions hard for the GA to optimize.

9.3 Deception and GA Performance

The previous subsection raised some concerns suggested by the Tanese functions about how deception should be defined. We also have some concerns about the role played by the notion of deception in understanding GAs. It has been common in the GA literature to characterize problems containing deception as "GA-hard" and problems lacking deception as "GA-easy" (e.g., see Goldberg & Bridges, 1988; Liepins & Vose, 1990; and Wilson, 1991), but these are unfortunate identifications, since they imply that every problem containing deception will be hard for the GA and every problem lacking deception will be easy for the GA.

However, the GA can easily succeed on many deceptive problems—for example, when the degree of deception is very slight. In addition, Grefenstette and Baker (1989) have questioned some of the assumptions made in applying Walsh analysis to GAs; their results imply that that certain types of deception might not cause the difficulty that Walsh analysis would predict. Conversely, non-deceptive problems can be difficult for the GA: for example, Mitchell, Forrest, and Holland's non-deceptive "Royal Road" functions (Mitchell, Forrest, & Holland, 1992; Forrest & Mitchell, 1992) can be constructed to be quite difficult for the GA. Thus, as the terms are currently used, "GA-hard" does not necessarily imply hard for the GA, and "GA-easy" does not necessarily imply easy for the GA. This is clearly a bad use of terminology[4].

Thus, deception is not the only factor determining how hard a problem will be for a GA, and it is not even clear what the relation is between the degree of deception in a problem and the problem's difficulty for the GA. There are a number of factors that can make a problem easy or hard, including those we have described in this paper: the presence of multiple conflicting solutions or partial solutions, the degree of overlap, and the *amount* of information from lower-order building blocks (as opposed to whether or not this information is misleading). Other such factors include sampling error (Grefenstette & Baker, 1989), the number of local optima in the landscape (Schaffer et al., 1989), and the relative differences in fitness between disjoint desirable schemas (Mitchell, Forrest, & Holland, 1992). Most efforts at characterizing the ease or difficulty of problems for the GA have concentrated on deception, but these other factors have to be taken into account as well.

At present, the GA community lacks a coherent theory of the effects on the GA of the various sources of facilitation or difficulty in a given problem; we also lack a complete list of such sources as well as an understanding of how to define and measure them. Until such a theory is developed, the terms "GA-hard" and "GA-easy" should be defined in terms of

[4]Stewart Wilson (personal communication) has proposed the term "veridical" to replace "GA-easy" in describing problems lacking deception.

the GA's *performance* on a given problem rather than in terms of the *a priori structure* of the problem. A general relation between a given problem's structure and the expected GA performance on that problem is still a very open area of research.

Finally, we are concerned about the focus on finding the global optimum that is implicit in many definitions of deception (and thus of "GA-hard" and "GA-easy"). In many applications it is infeasible to expect the GA to find the global optimum (and in many applications the global optimum is not known). Moreover, it could be argued that in general the GA is more suited to finding good solutions quickly than to finding the absolute best solution to a problem. But some researchers define deception solely in terms of hyperplanes leading toward or away from the global optimum (e.g., see Liepins & Vose, 1990). This narrows the concept of deception and makes it less generally useful for characterizing problems according to probable GA success or failure, especially when "success" does not require that the exact optimum point be found. Also, several so-called deceptive problems are essentially multi-peaked functions with a negligible difference between the global optimum and the runner-up; for many applications it would not matter which one the GA found. Should such cases be classified with the stigma of deception? Or even worse, should such cases be relegated to the foreboding class of "GA-hard"? We need to clarify what it *means* for the GA to succeed or fail as well as what causes it to succeed or fail before we can define "GA-hard" and "GA-easy."

10. Conclusions

After examining several possible causes for the GA's poor performance on the Tanese functions, we reach several conclusions:

- Overlaps in the defined loci between different j's in a given Tanese function created functions of much higher effective order than the actual order of their Walsh terms. This is the principle reason for the difficulty of the functions.
- The lack of information from lower-order schemas hampered crossover from contributing to the search. This was a secondary cause of the GA's poor performance, and largely accounts for the superior performance of hillclimbing.
- Long defining lengths in the non-zero partitions contributed slightly to the GA's poor performance but were not a major factor. This is related to the ineffectiveness of crossover on these functions. On other problems for which crossover *is* effective, useful schemas with long defining lengths would be more likely to be destroyed by crossover and would thus cause difficulty for the GA.

Tanese's results could be erroneously interpreted to imply that *all* Walsh polynomials are difficult for GAs, and that hillclimbing will generally outperform the GA on such functions. Such a result would be very negative for the GA, since any real-valued function defined on bit strings can be expressed as a Walsh polynomial. However, the experiments and analyses reported in this paper suggest that Tanese's results should not be interpreted as a general negative verdict on GAs. The functions Tanese studied are a highly restricted subset of the class of Walsh polynomials and have several peculiar properties that make them difficult to search. Her results could also mistakenly be taken to imply that partitioned GAs with smaller subpopulations will always outperform traditional GAs. This may not be true for functions in which recombination of lower-order building blocks plays a major role in the search.

These results raise the important question of what problems are well-suited for the GA, and more importantly, what features of these problems distinguish the GA from other methods. We have shown in this paper that the GA's difficulty with the Tanese functions was due to factors other than deception, which has historically been the focus of theoretical work on GAs. The results discussed in this paper underscore the fact that there are a number of different, independent factors that can be sources of difficulty for a GA, and we believe that these various factors must be identified and studied as part of the effort to characterize different problems in terms of the likelihood of GA success. The promise of GAs is based on the belief that "discovery and recombination of building blocks, allied with speedup provided by implicit parallelism, provides a powerful tool for learning in complex domains" (Goldberg & Holland, 1988b), but a better understanding of the details of building-block processing and implicit parallelism, and how they are affected by the structure of a given problem, is needed in order to use GAs most effectively.

One clearly important factor lacking in the Tanese functions is the availability of lower-order building blocks which can combine to produce fit higher-order schemas; such hierarchical schema-fitness structure is what makes crossover an effective operator. The "building blocks hypothesis" (Holland, 1975; Goldberg, 1989c) implies that the degree to which a fitness landscape contains such structure will in part determine the likelihood of success for the GA. Another hypothesized contributing factor is the degree to which the fitness landscape contains different "mountainous" regions of high-fitness points that are separated by "deserts" of low-fitness points (Holland, 1989). In order to travel from a low mountainous region to a higher one, a low-fitness desert must be crossed. Point-mutation methods such as hillclimbing can have very high expected waiting times to cross such deserts (if they can cross them at all); the hypothesis is that the GA will be much more likely to quickly cross such deserts via the crossover operation.

The degree to which these factors are present in a given problem may depend to a large extent on the representation chosen for the problem; the role of representation in GA function optimization has been discussed by Liepins and Vose (1990). We are currently studying the behavior of the GA and other search methods on a class of functions in which the degree to which these and other factors are present can be directly varied (Mitchell, Forrest, & Holland, 1992; Forrest & Mitchell, 1992), and we are investigating ways in which the presence of such features can be detected in a given function with a given representation. We believe that this investigation will lead to a better understanding of the classes of problems for which the GA is likely to be a successful search or learning method.

Acknowledgments

This work is a natural follow-up to Reiko Tanese's thesis work. We are grateful for the interesting results that she uncovered and the clarity with which she reported them in her dissertation. We thank John Holland and Quentin Stout for providing many valuable suggestions for this research, Robert Smith for a number of helpful discussions about the notion of GA deception, and Lashon

Booker, Greg Huber, Gunar Liepins, and Rick Riolo for their comments on earlier drafts of this paper. We also thank Emily Dickinson, Julie Rehmeyer, and Yuval Roth-Tabak for help in creating the figures and graphs in this paper. Support for this project was provided by the Santa Fe Institute, Santa Fe, NM (support to both authors); the Center for Nonlinear Studies, Los Alamos National Laboratory, Los Alamos, NM, Associated Western Universities, and National Science Foundation grant IRI-9157644 (support to S. Forrest); and the Michigan Society of Fellows, University of Michigan, Ann Arbor, MI (support to M. Mitchell).

References

Belew, R. K. and Booker, L. B. (Eds.) (1991). *Proceedings of the Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Bergman, A. and Feldman, M. W. (1990). More on selection for and against recombination. *Theoretical Population Biology 38 (1)*, 68–92.

Bethke, A. D. (1980). Genetic Algorithms as Function Optimizers. (Doctoral dissertation, University of Michigan.) *Dissertation Abstracts International, 41(9)*, 3503B.

Das, R. and Whitley, L. D. (1991). The only challenging problems are deceptive: Global search by solving order-1 hyperplanes. In R. K. Belew and L. B Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Davis, L. D. (Ed.) (1987). *Genetic Algorithms and Simulated Annealing.* Research Notes in Artificial Intelligence. Los Altos, CA: Morgan Kauffmann.

Davis, L. D. (Ed.) (1991). *The Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold.

De Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor, MI.

De Jong, K. A. (Ed.) (1990a). Special issue on genetic algorithms. *Machine Learning 5(4).*

De Jong, K. A. (1990b). Introduction to the second special issue on genetic algorithms. *Machine Learning, 5(4)*, 351–353.

De Jong, K. A. and Spears, W. (1991). Learning concept classification rules using genetic algorithms. In *Proceedings of the Twelfth International Conference on Artificial Intelligence*, 651–656. San Mateo, CA: Morgan Kaufmann.

Forrest, S. (1985). *Documentation for prisoner's dilemma and norms programs that use the genetic algorithm.* Unpublished report, University of Michigan, Ann Arbor, MI.

Forrest, S. and Mitchell, M. (1991). The performance of genetic algorithms on Walsh polynomials: Some anomalous results and their explanation. In R. K. Belew and L. B. Booker (Eds). *Proceedings of the Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Forrest, S. and Mitchell, M. (1992). Towards a stronger building-blocks hypothesis: Effects of relative building-block fitness on GA performance. To appear in L. D. Whitley (ed.), *Foundations of Genetic Algorithms 2.* San Mateo, CA: Morgan Kaufmann.

Goldberg, D. E. (1985). *Optimal initial population size for binary-coded genetic algorithms.* Technical Report TCGA Report No. 85001, University of Alabama, Tuscaloosa, AL.

Goldberg, D. E. (1987). Simple genetic algorithms and the minimal deceptive problem. In L. D. Davis (Ed.), *Genetic Algorithms and Simulated Annealing.* Research Notes in Artificial Intelligence. Los Altos, CA: Morgan Kaufmann.

Goldberg, D. E. (1989d). Sizing populations for serial and parallel genetic algorithms. In J. D. Schaffer (Ed.) (1989). *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

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

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

Goldberg, D. E. (1989c). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison Wesley.

Goldberg, D. E. (1991). *Construction of high-order deceptive functions using low-order Walsh coefficients.* Technical Report 90002, Illinois Genetic Algorithms Laboratory, Dept. of General Engineering, University of Illinois, Urbana, IL.

Goldberg, D. E. and Bridges, C. L. (1988). *An analysis of a reordering operator on a GA-hard problem.* Technical Report 88005, The Clearinghouse for Genetic Algorithms, Dept. of Engineering Mechanics, University of Alabama, Tuscaloosa, AL.

Goldberg, D. E. and Holland, J. H. (Eds.) (1988a). Special issue on genetic algorithms. *Machine Learning 3(2-3).*

Goldberg, D. E. and Holland, J. H. (1988b). Introduction to the first special issue on genetic algorithms. *Machine Learning 3(2-3)*, 95–99.

Goldberg, D. E., Korb, B., and Deb, K. (1990). Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems 3*, 493–530.

Grefenstette, J. J. (Ed.) (1985). *Proceedings of the First International Conference on Genetic Algorithms and Their Applications.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Grefenstette, J. J. (Ed.) (1987). *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Grefenstette, J. J. and Baker, J. E. (1989). How genetic algorithms work: A critical look at implicit parallelism. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell (Eds.), *Machine Learning II*, 593–623. San Mateo, CA: Morgan Kaufmann.

Holland, J. H. (1988). The dynamics of searches directed by genetic algorithms. In Y. C. Lee (Ed.), *Evolution, Learning, and Cognition*, 111–128. Teaneck, NJ: World Scientific.

Holland, J. H. (1989). Using classifier systems to study adaptive nonlinear networks. In D. L. Stein (Ed.), *Lectures in the Sciences of Complexity* (Vol. 1, pp. 463–499). Reading, MA. Addison-Wesley.

Koza, J. R. (1990). *Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems.* Technical Report STAN-CS-90-1314, Department of Computer Science, Stanford University, Stanford, CA.

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

Miller, G. F., Todd, P. M., and Hegde, S. U. (1989). Designing neural networks using genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, 379–384. San Mateo, CA: Morgan Kaufmann.

Mitchell, M., Forrest, S., and Holland, J. H. (1992). The royal road for genetic algorithms: Fitness landscapes and GA performance. In *Proceedings of the First European Conference on Artificial Life.* Cambridge, MA: MIT Press/Bradford Books.

Packard, N. H. (1990). A genetic learning algorithm for the analysis of complex data. *Complex Systems 4(5).*

Schaffer, J. D. (Ed.) (1989). *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Smith, S. F. (1980). *A learning system based on genetic adaptive algorithms.* Unpublished Ph.D. dissertation, Computer Science Department, University of Pittsburgh, Pittsburgh, PA.

Tanese, R. (1989a). *Distributed Genetic Algorithms for Function Optimization.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor, MI.

Tanese, R. (1989b). Distributed genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA. Morgan Kaufmann.

Walsh, J. L. (1923). A closed set of orthogonal functions. *American Journal of Mathematics, 55*, 5–24.

Whitley, L. D. (1991). Fundamental principles of deception in genetic search. In G. Rawlins (Ed.), *Foundations of Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

Whitley, L. D., Dominic, S., and Das, R. (1991). Genetic reinforcement learning with multilayer neural networks. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, 562–569. San Mateo, CA: Morgan Kaufmann.

Wilson, S. W. (1991). GA-easy does not imply steepest-ascent optimizable. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of The Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.