Hyperplane Ranking, Nonlinearity and the Simple Genetic Algorithm

Darrell Whitley
Computer Science Department
Colorado State University
Fort Collins, CO 80523
whitley@cs.colostate.edu

Robert B. Heckendorn
Computer Science Department
University of Idaho
Moscow, ID 83844
heckendo@cs.uidaho.edu

Soraya Stevens
BBN Technologies
10 Moulton Street
Cambridge, MA 02138
sstevens@bbn.com

May 15, 2003

Abstract

We examine the role of hyperplane ranking during genetic search by developing a metric for measuring the degree of ranking that exists with respect to static hyperplane averages taken directly from the function, as well as the dynamic ranking of hyperplanes during genetic search. We show that the degree of dynamic ranking induced by a simple genetic algorithm is highly correlated with the degree of static ranking that is inherent in the function, especially during the initial generations of search. The ϕ metric is designed to measure the consistency of an arbitrary ranking of hyperplanes in a partition with respect to a target string. Walsh coefficients can be calculated for small functions in order to characterize sources of linear and nonlinear interactions. Correlations between the ϕ metric and convergence behavior of a simple genetic algorithm are studied over large sets of functions with varying degrees of nonlinearity.

1 Introduction

A recurring theme in Holland's 1975 *Adaptation in Natural and Artificial Systems* [10] is that genetic algorithms process schemata. One of the ways they are supposed to do this is by ranking schemata in the population according to their usefulness. Yet this idea has become extremely controversial in recent years, with some researchers questioning schema processing (e.g. [1] [15] [5]), or dismissing the notion of schema processing altogether. Other researchers continue to make some form of schema processing a key component of their models of genetic algorithms (c.f. [9, 11, 12]). A critical question, then, is to what degree are genetic algorithms capable of schema processing?

Whitley et al. [17] developed the metric ϕ which can be used to measure the consistency of an ordered set of hyperplanes in a partition with respect to a specified target string. This metric makes it possible to measure the consistency of the ranking of hyperplanes in all partitions of the search space with respect to a target string when the hyperplanes are 1) sorted with respect to their static average fitness or, alternatively, 2) sorted by proportional representation in the population of an actual genetic algorithm.

It can be shown empirically that during the first few generations of a genetic algorithm the dynamic ranking of the hyperplanes in any partition, measured in terms of their proportional representation in the population, is highly correlated with the average hyperplane fitness measured

over all strings in each hyperplane. As the number of generations of the genetic algorithm increases, schemata (representing hyperplanes) in a population are not dynamically ranked according to their observed fitness. Instead, empirical results suggest that the proportional representation of schemata in the population is related to their consistency with the string or set of strings that come to dominate the population. This work provides indirect evidence that genetic algorithms are sensitive to information found in numerous hyperplane partitions. However, the degree to which a genetic algorithm is able to ultimately exploit this information is limited by the degree of consistency in the ranking that inherently exists in the function.

This paper also presents evidence to support the hypothesis that the degree of nonlinearity of a function has a direct impact on the degree to which a function displays a consistent static ranking. To study the effects of nonlinearity we introduce the ϕ *spectrum* metric. This is a measure of ϕ with respect to all points in the search space. We look at functions with differing degrees of nonlinearity created using Walsh coefficients [2, 13] and measure the distribution of the nonlinearity with a measure called the *Walsh sum*.

With these analysis tools we show that the static ϕ metric is a strong indicator of dynamical convergence behavior. In particular we show how the ϕ *spectrum* indicates which strings are most likely to survive in the population of a simple genetic algorithm.

1.1 Basic Notation

Let L be the length of the binary strings which we are processing. A string of all 1's will be denoted by $\vec{1}$. A **schema** is one of the 3^L strings of 0's, 1's and $*$'s where a 0 or 1 occupy each **fixed bit position** and the $*$'s represent either a 0 or a 1 in the **variable bit positions**.

A schema represents a **hyperplane** which is a set of strings with the same set of fixed bit positions. A schema with C fixed bit positions represents a hyperplane of **order** C containing 2^{L-C} strings. A **partition** is a set of competing hyperplanes specified by a string with bits b in positions of competition and $*$'s in all other positions. A partition with C b's and $(L-C)$ $*$'s is of **order** C and defines a set of 2^C hyperplanes. In this paper bit positions in strings and schemata are numbered from *right to left*, starting with 1. This has the representational advantage that a string with a single bit at position i has a numeric value of 2^{i-1}. For example, the set of schemata {00**, 01**, 10**, 11**} constitutes the order-2 partition denoted by "bb**" with bit positions 3 and 4 selected.

2 Background

Holland [10] originally explained the computational power of genetic algorithms by developing a theory of how genetic algorithms process hyperplanes represented by schemata. Certain aspects of the theory regarding how hyperplanes are processed by genetic algorithms have been under attack for the last few years [15].

The criticisms of the various theories related to hyperplane sampling take two general forms. First, counterexamples to some of the basic schema processing hypotheses have been developed. Specifically, the 2-armed bandit analogy suggests that genetic algorithms allocate exponential trials to the observed best hyperplane subpartitions; there are certainly counterexamples to show this is not always true [4, 5, 6]. Another conjecture is that a simple genetic algorithm *ranks* hyperplanes

according to fitness. This implies that hyperplanes with a higher average static fitness are allocated a higher proportional representation in the population over time. Counterexamples presented in this paper also prove that the genetic algorithm does not always rank hyperplanes according to fitness.

The second form of criticism of the schema processing hypothesis is that very little has actually been proven aside from the schema theorem itself. New theoretical models of how the genetic algorithm behaves as a dynamical system, such as those based on Markov models [16], shed little or no light on schema processing.

This paper takes an empirical approach to studying some of the principles that motivated Holland's original theory about schema processing and hyperplane ranking. This paper also attempts to determine where that theory fails and if some restricted form of that theory might hold under more specific conditions. The goal of this paper is not to solve many of the outstanding theoretical problems associated with the schema theorem and schema processing, but rather to explore directions that could provide a foundation for such a theory. One of the goals of this paper is to provide a better understanding of when and to what degree a simple genetic algorithm ranks hyperplanes according to fitness.

A metric is developed that measures the degree to which the proportional representation of hyperplanes in a population corresponds to the static average fitness of strings contained in those hyperplane subpartitions. The results suggest that the degree to which the genetic algorithm ranks hyperplanes depends very much on the degree to which those hyperplanes are consistently ranked statically when all points in the search space are evaluated. Thus, there appear to be special conditions under which a simple genetic algorithm does rank many competing hyperplanes in an appropriate fashion, although it does not and cannot consistently rank all hyperplanes for most functions, as is illustrated in the next section.

2.1 Does the Genetic Algorithm Rank All Schemata?

Let μ_ξ be the average evaluation of all strings in hyperplane ξ. Let $P(\xi, t)$ represent the proportion of a population that samples hyperplane ξ at time t during the execution of a simple genetic algorithm. Also, $f(\xi, t)$ represents the average fitness of the strings in a population that sample ξ at time t during genetic search. Holland suggests that the average fitness of strings sampled from hyperplane ξ at time t will come to estimate μ_ξ and that each hyperplane ξ will come to be represented in the population in accordance with the *observed* fitness of that hyperplane. More precisely, Holland states [10] (page 88):

> The discussion of "intrinsic parallelism" in chapter 4 would imply here that *each* [hyperplane] ξ represented in [the population] $\beta(t)$ should increase (or decrease) at a rate proportional to *its* observed "usefulness" .... If this could be done consistently then each ξ would be automatically and properly ranked within $\beta(t)$ as t increases.

If we equate "usefulness" with $f(\xi, t)$ and the increase (or decrease) of ξ with $P(\xi, t)$, then the preceding discussion of intrinsic parallelism might be rephrased in the form of the following conjecture.

$$\forall(\xi_x, \xi_y) \quad \Big\{ P(\xi_x, t) > P(\xi_y, t) \quad \text{iff} \quad \mu_{\xi_x} > \mu_{\xi_y} \Big\}.$$

Such a statement is clearly not true in general for any arbitrary pair of hyperplanes.

Observation: If $\xi_x \subseteq \xi_y$ then $P(\xi_y, t) \geq P(\xi_x, t)$ even if $\mu_{\xi_y} < \mu_{\xi_x}$.

Proof: If $\xi_x = \xi_y$ then $P(\xi_y, t) = P(\xi_x, t)$. If ξ_x is a proper subset of ξ_y then there must exist a set of hyperplanes which we will denote by $\{\xi_y - \xi_x\}$ which together with ξ_x partitions ξ_y. We will extend our definition of P(H,t) to include sets of hyperplanes, such that

$$P(\{S\}, t) = \sum_{\xi \in S} P(\xi, t).$$

Then by definition $P(\xi_y, t) = P(\xi_x, t) + P(\{\xi_y - \xi_x\}, t)$. If ξ_x is a proper subset of ξ_y and $P(\xi_y, t) = P(\xi_x, t) + P(\{\xi_y - \xi_x\}, t) \neq 0$ then it follows $P(\xi_y) > P(\xi_x, t)$, even if $\mu_{\xi_y} < \mu_{\xi_x}$. □

In simple terms, the proportional representation of a lower order hyperplane must be greater than or equal to the proportional representation of any higher order hyperplane that is fully contained within the lower order hyperplane, regardless of their average fitness values.

Therefore, we explore a more restricted notion of ranking. We specifically look at ranking among the schemata in a hyperplane *partition* [18]. From a dynamic point of view, one might argue that a partition denoted by H, is ranked if

$$\forall(\xi_x, \xi_y)\epsilon H \quad \left\{P(\xi_x, t) > P(\xi_y, t) \quad \text{iff} \quad \mu_{\xi_x} > \mu_{\xi_y}\right\}.$$

This definition of hyperplane ranking also fails to characterize what actually happens during genetic search. For example, assume the schemata 000** and *111* have the highest static average fitness within their respective hyperplane competitions. Clearly, both 000** and *111* cannot continually increase their representation in the population as measured by $P(\xi, t)$ since the decision whether to select for the 0 or 1 bit in positions 3 and 4 must eventually be resolved. If $*111*$ increases its representation in the population, hyperplane 000** will be represented less than other hyperplanes such as 011** even though $\mu_{\xi_{000**}} > \mu_{\xi_{011**}}$. Thus, one cannot predict in general how a genetic algorithm will dynamically rank a partition of competing schemata without considering what is happening in other overlapping partitions. A major criticism of the idea of "schema processing" is that it ignores the interactions between schema, which are clearly important to the dynamical behavior of a genetic algorithm.

A key problem that arises when trying to relate measurements based on dynamical population behavior (e.g. $P(\xi, t)$) to static hyperplane averages (e.g. μ_ξ) is that *inconsistent* bit patterns may exist over different sets of hyperplane partitions when ranked according to static hyperplane fitness averages. On the other hand, the degree to which inconsistency can exist in the population is very much constrained. This implies that there can be relationships between hyperplanes with respect to measurements based on μ that cannot be captured in the relative population representation of hyperplanes as measured by $P(\xi, t)$.

3 Measures of Consistency and Complexity

We will use the notation $R(\xi) = \mu_\xi$ to denote a ranking function based on the average fitness of the strings in hyperplane ξ. All hyperplanes within a particular partition can be ranked according to their average fitness. This partition is said to have a **consistent static ranking** if there exists a **target string**, τ, such that when all the hyperplanes in the partition are sorted with respect to their average fitness, μ, each schema (hyperplane) closer in Hamming distance to the target string will have a greater fitness than any schema (hyperplane) farther away. Note that the target string can be any string in the space. Assume the target string is $\vec{1}$. Consider the following fitness relationships for the partitions $bb**$ and $*bb*$ which have a consistent static ranking:

$$\mu_{11**} > \{\mu_{10**}, \mu_{01**}\} > \mu_{00**} \quad \text{and} \quad \mu_{*11*} > \{\mu_{*10*}, \mu_{*01*}\} > \mu_{*00*}$$

Notice that a given partition may support several choices for the target string. For example, the second partition could fully support 0110 as the optimal string as well as 1111. However, the first partition can't fully support 0110 since the fitness of $11**$ is better than $01**$.

Given the definition of consistency for a partition, we can use that definition to define consistency for a function. We say that a **function** has a **consistent static ranking** if all partitions can be consistently statically ranked with respect to the *same* target string. For example, consider the ONEMAX function which counts the number of 1 bits in the string. The maximum evaluation for this function is at the string $\vec{1}$. Regardless of the partition, schema with more 1 bits will correspond to higher ranked hyperplanes. Therefore the entire function is consistently statically ranked.

When functions are consistent, however, it is easily proved that they can only be consistent about the global optimum.

Theorem 1 *Functions can only be consistently ranked when the target string is the global optimum.*

Proof:

Let the global optimum be denoted by τ_{opt}. Choose any string other than a global optimum as the target string, and denote this by τ. If the function is consistent, then all partitions must be consistent with respect to τ. That is to say that all hyperplanes within all partitions are ranked such that the hyperplane containing τ is always ranked highest. But the partition composed of the set of all strings, must also be ranked according to the fitness of that partition, which implies that $\tau_{opt} < \tau$. This results in a contradiction since $\tau < \tau_{opt}$. $\square$

It can also be shown that similar restrictions on other partitions also reinforce the notion that a function can only be consistent with the global optimum. More often than not, nonlinear functions are not consistent. A common conflict occurs when two partitions share overlapping fixed bit-values in their schema definitions, but the ranking are such that hyperplanes with different and conflicting bit values may dominate in the two partitions. The following is an example of two partitions having an inconsistent static ranking no matter what target string is chosen:

$$\mu_{11**} > \{\mu_{10**}, \mu_{01**}\} > \mu_{00**} \quad \text{and} \quad \mu_{*00*} > \{\mu_{*10*}, \mu_{*01*}\} > \mu_{*11*}$$

If genetic algorithms are sampling hyperplanes according to their relative fitness then this inconsistent static ranking should have an impact on the performance of the genetic algorithm and its dynamical behavior. Note that if a simple genetic algorithm is applied to the example function

above, as the population converges, the second bit position cannot be both a 1 and a 0; one or the other must come to dominate the population.

Let $P(\xi, t)$ denote the proportional representation of hyperplane ξ at generation t in a genetic algorithm. If a function has a consistent static ranking then this should have implications for a dynamic measurements based on $P(\xi, t)$. The orderings

$$P(11**, t) > P(00**, t) \quad \text{and} \quad P(*00*, t) > P(*11*, t)$$

cannot both be true as the population reaches convergence. However, the orderings

$$P(11**, t) > P(00**, t) \quad \text{and} \quad P(*11*, t) > P(*00*, t)$$

can both be true as the population reaches convergence. If static hyperplane relationships impact dynamical sampling of the hyperplanes, then a consistency metric should be a useful tool for characterizing functions and how they are processed by a simple genetic algorithm.

To support this comparison we say that a **partition** has a **consistent dynamic ranking** at time t if there exists a target string, such that when all the hyperplanes in the partition are sorted with respect to their proportional representation, $R(\xi) = P(\xi, t)$, each hyperplane closer in Hamming distance to the target string will have a greater representation than any hyperplane farther away. It is clear that a consistent dynamic ranking of a function is defined similar to its static analog but with R being a dynamic measure.

3.1 Degree of Consistency for a Partition and the ϕ Metric

It is useful to quantify the amount of consistency inherent in a partition or function. This section describes the ϕ metric as proposed by Whitley et. al. [17] which can be used to measure the *degree of consistency* of a partition. The next section will expand the definition of ϕ to define the *degree of consistency* for a function.

The degree of ranking for an order k partition π containing hyperplanes $\xi_1, \xi_2, ...\xi_{2^k}$ is denoted by $\phi(R, \pi, \tau)$ and defined as follows[1]

$$\phi(R, \pi, \tau) = \sum_{i=1}^{2^k} \sum_{j=1}^{2^k} [\text{Pred}(R(\xi_i) > R(\xi_j))\ \text{Pred}(M(\xi_i, \tau) > M(\xi_j, \tau))]\ (M(\xi_i, \tau) - M(\xi_j, \tau))$$

where R is a ranking function of a hyperplane, π is a k order partition, τ is a target string and Pred(*expression*) returns a 1 if the expression is true and 0 if it is false. The function to which this is applied is left out of the argument list to reduce the notational load. The expression in brackets [...] also returns 0 or 1, and when multiplied by the remaining expression, $(M(\xi_i, \tau) - M(\xi_j, \tau))$, acts as a selector to indicate when the expression should be added to the sum. $M(\xi, \tau)$ is a **match count function** that measures the number of bits that match between the target string τ and hyperplane ξ in the fixed bit positions. For example, M(*1100*, 110100) = 2. M can be thought of as an inverse Hamming distance function. In using ϕ we often assume $\tau = \vec{1}$. In these cases we denote $M(\xi, \vec{1})$ as $M(\xi)$ and $\phi(R, \pi, \vec{1})$ as $\phi(R, \pi)$.

The metric ϕ is a summation of the difference between the match counts of all ordered pairs of hyperplanes in which the ordering of the pair by match count and by ranking agree in one direction.

[1] This definition is functionally the same as in Whitley et al. [17] but expressed differently to more clearly show the conditions under which the value of ϕ is increased.

ξ_i	$R_1(\xi_i)$	$M(\xi_i)$	ξ_i	$R_2(\xi_i)$	$M(\xi_i)$
111***	19.0	3	111***	21.0	3
110***	17.0	2	100***	13.0	1
101***	13.0	2	010***	8.0	1
011***	11.0	2	101***	5.0	2
001***	7.0	1	001***	3.0	1
010***	5.0	1	011***	2.0	2
100***	3.0	1	000***	1.0	0
000***	2.0	0	110***	1.0	2
$\phi(R_1, \pi) = 30$			$\phi(R_2, \pi) = 20$		
$\widehat{\phi}(R_1, \pi) = 1.0$			$\widehat{\phi}(R_2, \pi) = 0.66\overline{6}$		

Table 1: Computations of ϕ for two Example Rankings.

The ϕ function has its largest value for a partition when a sort of all of the hyperplanes in the partition by increasing R results in the hyperplanes being sorted by increasing M. Thus the ϕ function has a larger value when the hyperplanes in π are more consistently ranked. ϕ is zero when a set of hyperplanes are sorted by decreasing M by any given ranking function.

For the case where $R = \mu$, we say ϕ measures the **degree of static ranking** for a partition. When $R = P(\xi, t)$ as measured by the proportional representation in some genetic algorithm population we say ϕ measures the **degree of dynamic ranking** for a partition at time t. We will also refer to this static ranking as R_{stat} and the dynamic ranking at time t as $R_{dyn}(t)$.

The maximum possible ϕ value for a partition π of order k can be computed [17] as:

$$\phi^{max}(\pi) = \sum_{q=1}^{k} \binom{k}{q} \sum_{i=0}^{q-1} \binom{k}{i} (q - i) \quad \text{where} \quad k = order(\pi)$$

Notice that ϕ^{max} does not take τ or R as arguments since the target string and ranking function are irrelevant to computation of the maximum possible ϕ. ϕ^{max} allows ϕ to be normalized by dividing each count so that the resulting measure for each partition is between 0 and 1.

$$\widehat{\phi}(R, \pi, \tau) = \phi(R, \pi, \tau)/\phi^{max}(\pi) \qquad \in [0, 1]$$

Table 1 shows an example computation of $\phi(R, \pi)$ for two different ranking functions R_1 and R_2 and the partition $\pi = bbb***$. The definition of the ranking functions are not specified, but their values are given in the table. The table is sorted by R from greatest to least to aid hand verification of the results at the bottom of the table. Whereas $\phi(R_1, \pi) = \phi^{max}(\pi)$, R_2 provides a more instructive example of the computation of ϕ. The two sums in the formula for ϕ examine all possible ordered pairs. If we take the first two hyperplanes for example, we evaluate the product of the two predicates as:

$$\text{Pred}(R(111***) > R(100***))\ \text{Pred}(M(111***, \vec{1}) > M(100***, \vec{1}))$$

which becomes the product Pred(21.0 > 13.0) Pred(3 > 1). Since both predicates are true their product returns 1. This is multiplied by

$$(M(111***, \vec{1}) - M(100***, \vec{1})) = 3 - 1$$

giving a value of 2 which is added to the sum. The other terms are similarly computed to produce the final calculation, $\phi(R_2, \pi) = 20$, which is normalized using $\phi_{max} = 30$ to yield $\widehat{\phi}(R_2, \pi) = 0.66\overline{6}$.

$\phi(R, \pi, \tau)$ exhibits an interesting property for $R = R_{dyn}(t)$ in a simple genetic algorithm with no mutation. As the population converges, a single individual eventually becomes the only member of the population. This means that $Pred(R(\xi_i) > R(\xi_j))$ is true for only one value of i in each partition while the remaining hyperplanes are *tied*. Ties contribute nothing to the value of ϕ. As a population converges, the increasing number of ties for the dynamic ranking results in a decrease in the attainable value of ϕ. For this reason our experiments limited the observation of ϕ to the first 20 generations.

3.2 Degree of Consistency for a Function and the ϕ_{sum} Metric

As stated earlier, functions are usually not fully consistent; consistency is usually a matter of degree. We define the **degree of consistency** for a function with respect to a target string τ as:

$$\phi_{sum}(R, \tau) = \sum_{\pi \in P} \widehat{\phi}(R, \pi, \tau), \qquad R = R_{stat}$$

where P is the set of all partitions. Thus, ϕ_{sum} is measured over 2^L partitions and ranges from 0 to $2^L - 1$. The maximum degree of consistency, $2^L - 1$, occurs only when a function is consistent.

4 Experimental Design

The goal of these experiments was to compare ϕ measurements using static and dynamic ranking functions. The dynamic analysis was done using an executable infinite population model of a simple genetic algorithm [14, 19]. An infinite population model was chosen because it is less susceptible to truncation effects associated with finite populations. Since no sampling is taking place, the infinite population model also offers a benefit in eliminating the need for multiple trials as would be required if a finite population were used. Our model also does not include mutation. The resulting model will converge to a population that is dominated by a single string. Since we compute consistency as compared to a single string, it seems reasonable to consider an infinite population model without mutation. Initially each string had an equal representation in the population. At the end of each generation t the proportional representation of each individual was collected, allowing us to compute $\phi_{sum}(R_{dyn}(t))$ for each generation using the dynamic ranking function.

4.1 Static and Dynamic Ranking on Random Functions

In preliminary experiments we used 114 random functions to compare the static ranking to the dynamic ranking when a model of the genetic algorithm was executed on each individual function. We generated 100 random 6 bit functions in order to evaluate the relationship between the static ranking of hyperplanes for each function and the dynamic ranking of the hyperplanes during genetic search for that same function. Functions were generated by randomly assigning an integer value between 0 and 255 to each of the 64 strings composing the search space for each function. We also wished to include a few fully consistent functions in the comparison of the static and dynamic rankings. We generated 100,000,000 random functions; only 14 of these functions were fully consistent. These were also included in our set of test functions. All functions were transformed in the

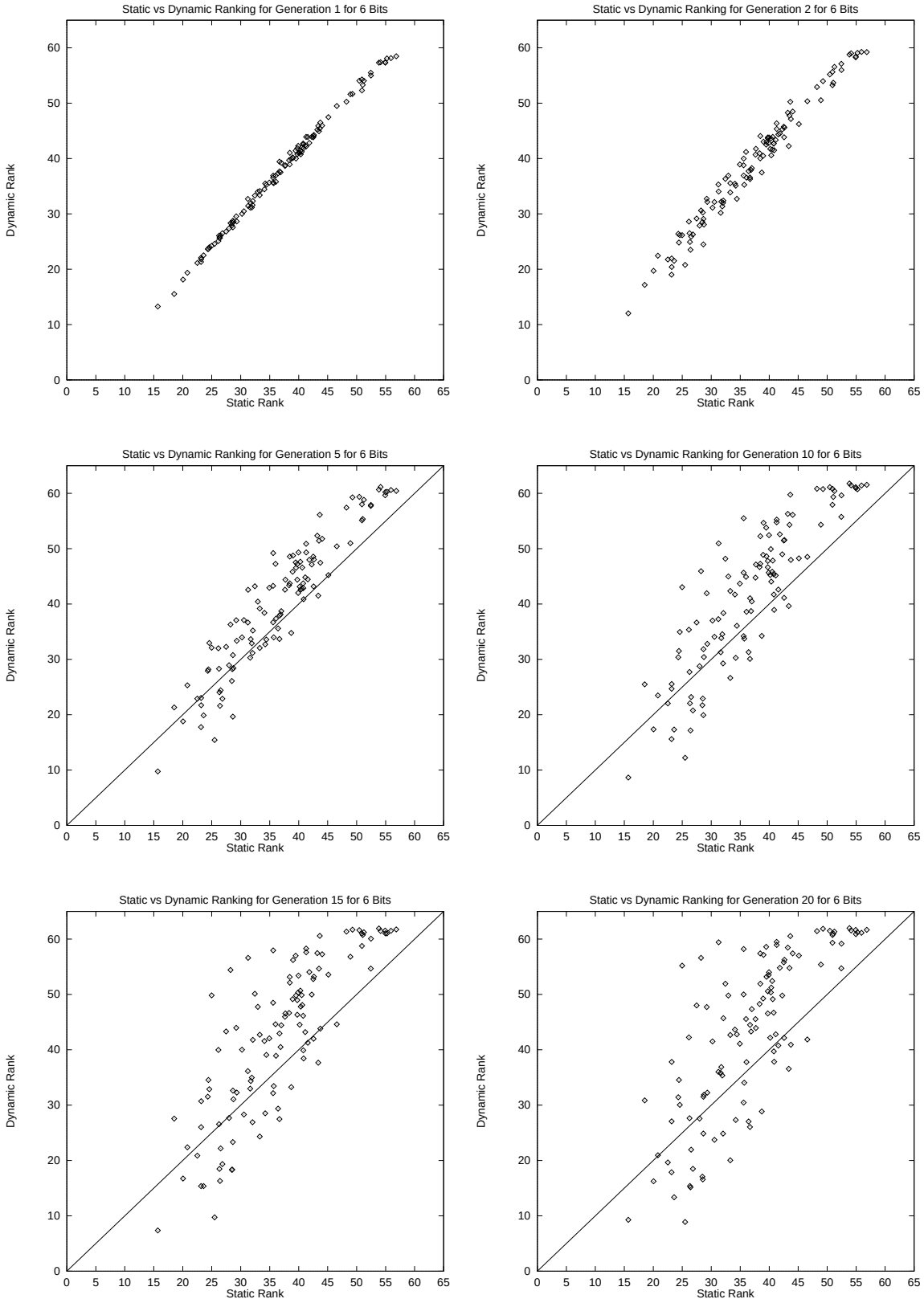

Figure 1: Static Ranking versus Dynamic Ranking for Generations 1, 2, 5, 10, 15 and 20, based on the dynamic metric $\phi_{sum}(R_{dyn}(t))$ and the static metric $\phi_{sum}(R_{stat})$. The target string is the global optimum.

domain space using exclusive-or so that the maximum was at $\vec{1}$. This has no effect on the value of ϕ_{sum}, since this affine transform does not disturb the Hamming distance between strings or the sets of hyperplane pairs that satisfy the predicates in the formula for ϕ.

We collected statistics based on a model of the genetic algorithm assuming an infinite population size, a crossover rate of 0.6 and a mutation rate of 0. The recombination operator was 1-point crossover. The results for generations 1, 2, 5, 10, 15 and 20 are shown in Figure 1.

For each function we computed $\phi_{sum}(R, \tau)$ over all hyperplane partitions; we denote this more briefly as ϕ_{sum}. Each point on the graph represents the dynamic ranking as measured by ϕ_{sum} plotted against the static ranking measured using ϕ_{sum} for one of the 114 functions.

Figure 1 shows that the degree of dynamic ranking during the first few generations is highly correlated with the degree of static ranking which is inherent in the functions. However, the correlation does not improve with time. Instead, the degree of dynamic ranking increases if the genetic algorithm converges toward the global optimum. (Since our model contained no mutation, it makes sense to talk about the simple genetic algorithm converging to a single point.) The degree of dynamic ranking with respect to the global optimum decreases if the genetic algorithm allocates increasing numbers of trials to hyperplanes with bit patterns that are inconsistent with the global optimum. The degree to which a point moves up or down indicates the number of bits shared with the global optimum found in the solution toward which the population is converging.

It should also be noted that this correlation is entirely emergent during the first generation of the simple genetic algorithm. It has nothing to do with the ranking of the population during generation 0 (before search begins). For generation 0, the distribution would not change for the static rankings (since these are fixed) and the dynamic ranking would be identical for all functions because the model has an identical representation for all hyperplanes over all functions. Thus, the distribution would appear as a straight horizontal line across the graph. (An example is given later in Figure 3). In the first generation this jumps from a flat line to the distribution shown for generation 1. This implies that the dynamic ranking as measured by ϕ_{sum} that emerges during generation 1 is very consistent with the static ranking of the function as measured by ϕ_{sum}. It is important that only during the first generation of the simple genetic algorithm is the dynamic ranking based on selection applied to a uniform sample over the entire function.

4.2 Test Functions with Controlled Nonlinearity: Walsh Sums

Nonlinearity is one of the major factors that makes search difficult. One way to look at the amount of interaction between bits is to use Walsh functions to examine the Walsh coefficients of the function. For an L bit function there are 2^L values in each of the domain and range of the function. There are also 2^L Walsh coefficients, denoted by w_i.

Walsh functions are used to analyze a function f, $f : \mathcal{B}^L \rightarrow \mathcal{R}$, that is the encoding of a problem domain in L bit space as a real valued function.

$$f(x) = \sum_{i=0}^{2^L-1} w_i \psi_i(x)$$

where:

- $\psi_i(x) : \mathcal{B}^L \times \mathcal{B}^L \rightarrow \{1, -1\}$ is the i^{th} **Walsh Function** of x.

- w_i : $\mathcal{B}^L \rightarrow \mathcal{R}$ is the i^{th} **Walsh coefficient** and indicates the degree of interaction of bits indicated by the positions of the 1's in i. The **order** of w_i is the number of one bits in i.

A Walsh Function can be expressed as:

$$\psi_j(x) = (-1)^{bc(j \wedge x)} = (-1)^{j^T x}$$

where $j, x \in \mathcal{B}^L$ and the function "bc" counts the number of 1 bits in the string $(j \wedge x)$. Thus, if $j^T x$ is odd, then $\psi_j(x) = -1$ and if $j^T x$ is even, then $\psi_j(x) = 1$.

A zero value for a Walsh coefficient means that the bit combination represented by the index of the coefficient does not play a role in the computation of the function [13].

For our experiments we wished to generate functions with limited nonlinearity. In this regard, Walsh functions can be used somewhat like NK-Landscapes [8]. We can therefore use Walsh coefficients to control for nonlinearity. Heckendorn and Whitley [7] discuss the connection between nonlinearity and Walsh coefficients in more detail. Goldberg [2] [3] also provides an introduction to Walsh functions.

We also would like to create a condensed measure of the degree of nonlinear interaction. Let the **Walsh sum**, denoted by W_z, be the sum of the absolute value of the coefficients for Walsh functions with exactly z bits set to 1. This is said to be the **order** z Walsh sum. For example $W_0 = |w_0|$, where w_0 is the only Walsh coefficient with no 1 bits, and $W_1 = |w_1| + |w_2| + |w_4| + \ldots$, where each w_i has only one 1 bit set in the binary representation of i. W_2 is the sum over the $\binom{n}{2}$ Walsh coefficients for the pairwise interactions between bits, etc. This means that for an L bit function there are $L + 1$ Walsh sums. Unlike the Walsh coefficients, which allow for the complete reconstruction of the function, the Walsh sums preserve only the orders of interactions.

The **order of a function**, $\Omega(f)$, is defined as the largest i such that $W_i \neq 0$. So, for example, if $\Omega(f) = 1$ then the function is bitwise linearly independent and there are no pairwise or higher order interactions.

4.2.1 Generating the Test Functions

The process of generating each test function began by randomly generating the Walsh coefficients for the function in the range $[-1, 1]$. This created a set of 2^8 Walsh coefficients for each 8 bit function. If the index of the Walsh coefficient had a number of 1's greater than the intended Ω for the function then the coefficient was set to 0 to limit the degree of interaction. We generated 6 sets of 100 test functions with $\Omega(f)$ equal to 1 through 6. We wanted to ensure a good diversity of functions along two measures: internal consistency as measured by ϕ_{sum} and limited order of interactions as measured by $\Omega(f)$.

For any given $\Omega(f)$, a scaling factor was devised so that the W_i would decay linearly starting at $W_0 = 1.0$ decaying to $W_{(\Omega(f)+1)} = 0$. These factors were applied to the Walsh coefficients. Our intent was to let the Walsh sums at higher orders, and hence the nonlinearity at higher orders, decay smoothly to zero. We conjecture that many naturally occurring optimization problems may exhibit some sort of bounded nonlinearity. A second feature of the scaling factor was that it fixed the value of Walsh sums at a constant from one function to the next so that fluctuations in interaction levels are not a variable in the our experiments.

From the Walsh coefficients a complete function table of 2^8 elements was generated using the fast Walsh transform of Cooley and Tukey [2] to map the function domain to its range. These

z	W_z	NumCoefs	%Nonzero
0	1	1	100.00
1	0.8333	8	100.00
2	0.6667	28	100.00
3	0.5	56	100.00
4	0.3333	70	100.00
5	0.1667	56	100.00
6	0	0	0.00
7	0	0	0.00
8	0	0	0.00

Table 2: Table of Walsh Sums for a Function from our Experiments with $\Omega(f) = 5$.

functions were also then transformed in the domain space using exclusive-or so that the maximum was at $\vec{1}$.

The particular infinite population model we used for the genetic algorithm requires that the values of the functions be nonnegative everywhere. The generated functions that had negative minimums were translated by subtracting the value of the minimum, $f(min)$, from each element in the function table. This translation is often done when using genetic algorithms and has no effect on the value of ϕ_{sum}. The value of W_0 will now be $W_0 - f(min)$ and all other W_i will remain constant. The results we present for the translated set of functions will be the same for the untranslated functions.

Table 2 is an example of a Walsh sum table for a function with $\Omega(f) = 5$ that we used in our experiments. In the first column is the order of the Walsh sum, z. The second is the Walsh sum W_z. The last two columns are the count of the number of Walsh coefficients of order z that are nonzero and what percentage this count is of the total possible Walsh coefficients of order z.

5 Results

Our initial experiment examines the relationship between static and dynamic ranking using random functions. We present similar results here, but using 6 sets of functions with $\Omega(f)$ equal to 1 through 6 to control for nonlinearity. At $\Omega(f) = 1$ the functions are linear, and higher values of $\Omega(f)$ correspond to higher levels of nonlinear in the sample functions.

5.1 Static versus Dynamic Ranking

Figure 2 shows the static ranking ($\phi_{sum}(R_{stat})$) plotted against the dynamic ranking ($\phi_{sum}(R_{dyn}(t))$) at generations 0, 1, 10 and 20. Data was generated for sets of 100 functions for each of the 6 values of Ω's; however, only 20 randomly selected functions out of the 100 functions for each $\Omega(f)$ were used in the plots. Each of the 120 points on the graph corresponds to a single function.

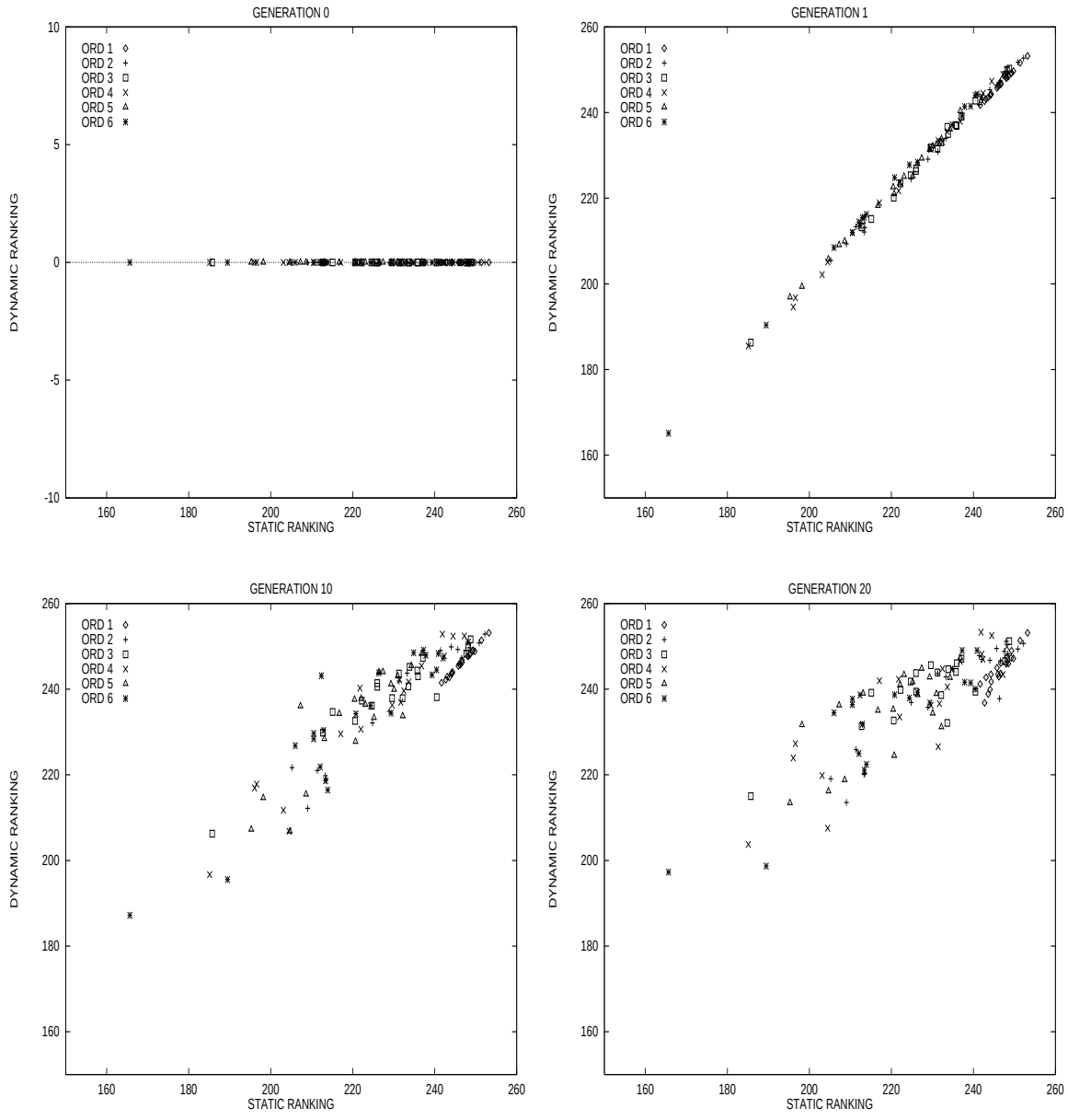

Figure 2: Static Ranking versus Dynamic Ranking for Generations 0, 1, 10 and 20, based on the dynamic metric $\phi_{sum}(R_{dyn}(t))$ and the static metric $\phi_{sum}(R_{stat})$. The target string is the global optimum.

$\Omega(f)$	Gen 1	Gen 5	Gen 10	Gen 20
1	0.9994	0.9984	0.9975	0.9223
2	0.9985	0.9748	0.9339	0.8639
3	0.9990	0.9791	0.9417	0.8622
4	0.9993	0.9870	0.9641	0.8954
5	0.9994	0.9731	0.9320	0.8525
6	0.9991	0.9747	0.9281	0.8454

Table 3: Pearson's r correlation between Static and Dynamic Ranks for functions with $\Omega(f) = 1$ through 6 (with 100 functions for each value of $\Omega(f)$).

At generation 0, all hyperplanes in a partition have equal representation. Therefore, all functions have the same $\phi_{sum}(R_{dyn}(0))$. At generation 1, however, $\phi_{sum}(R_{stat})$ and $\phi_{sum}(R_{dyn}(1))$ are very strongly correlated which is in part due to the fact that the genetic algorithm uniformly samples the space at the first generation. As the genetic algorithm continues execution, the correlation between the static and dynamic ranks decreases as can be seen in the two lower plots corresponding to generations 10 and 20. The data also suggests that functions with lower nonlinearity are somewhat more likely to display higher static and higher dynamic ranking overall; this suggests that functions with lower nonlinearity are more likely to display consistency in the static hyperplane rankings that can be exploited by a simple genetic algorithm.

Table 3 illustrates that the correlation between static and dynamic ranks also changes as we move from lower to higher orders of nonlinearity. The table was generated using the full set of functions, 100 functions per $\Omega(f)$. Given the correlation can range from 1 to -1 (1 being perfectly correlated and -1 begin perfectly negatively correlated), it is clear that the static and dynamic ranks are still strongly correlated at the 20th generation. However, the trend moving from lower to higher orders of nonlinearity is that the correlation between the static and dynamic ranks decreases. This behavior is due to the decreasing *consistency* in functions as $\Omega(f)$ is increased.

5.2 Static Ranking and Point of Convergence

It is clear that the static ranking is strongly correlated with the dynamic ranking using functions with varying degrees of nonlinearity. However, this does not answer the question of whether the static ranking can be used to predict the convergence behavior of a simple genetic algorithm.

Let τ_{opt} denote the global optimum and τ_C denote the point of convergence. Since ϕ can be computed about arbitrary points in the search space, we compare the ϕ computed respect to the global optimum, $\phi_{sum}(R_{stat}, \tau_{opt})$ and the ϕ at the point of convergence, $\phi_{sum}(R_{stat}, \tau_C)$. We specifically examined all of the cases where the point of convergence was some string other than the global optimum; in 93% of these cases $\phi_{sum}(R_{stat}, \tau_C)$ was higher than $\phi_{sum}(R_{stat}, \tau_{opt})$.

Table 4 shows, for sets of function with $\Omega(f) = 1$ through 6, the number of times the infinite population model genetic algorithm converged to a point at a given Hamming distance from the global optimum after 20 generations. For instance, of the 100 fourth order functions tested, 11 converged to a point 2 bits away from the optimum. A Hamming distance of zero indicates convergence to the optimum. Two observations follow from these results. First, as $\Omega(f)$ increases, the genetic algorithm becomes less likely to converge to the global optimum. Second, as $\Omega(f)$ increases, the Hamming distance between the point of convergence and the global optimum increases.

$\Omega(f)$	Hamming Dist.					% Not Converged to Optimum
	0	1	2	3	4	
1	100	0	0	0	0	0%
2	78	18	4	0	0	22%
3	78	19	2	1	0	22%
4	60	28	11	1	0	40%
5	56	26	15	2	1	44%
6	52	31	9	7	1	48%

Table 4: Hamming Distance from the Optimum for $\Omega(f) = 1$ through 6.

Ω	Average ϕ_{sum} for Convergence Point		Average ϕ_{sum} for Optimum Point	
	Mean	Std Dev	Mean	Std Dev
1	246.27	3.14	246.27	3.14
2	238.92	9.43	235.06	11.79
3	233.92	11.48	229.57	15.16
4	234.03	11.48	224.13	18.27
5	229.34	12.21	217.46	17.35
6	227.40	13.23	213.84	20.09

Table 5: Average ϕ_{sum} for Each Ω Tested.

The histogram in Figure 3 compares the distribution of $\phi_{sum}(R_{stat})$ with respect to both the global optimum and the point of convergence; the infinite population model was used to determine the point of the point of convergence. In the case of those functions where $\Omega(f) = 1$, the functions are linear and all of runs of the infinite population model converge to the global optimum. The histogram in Figure 3 counts how the remaining 500 nonlinear functions (with $\Omega(f) = 2$ to 6) are distributed with respect to $\phi_{sum}(R_{stat})$ when the target string is the global optimum versus when the target string is the point of convergence. Despite the fact that the global optimum was also the point of convergence in 70% of the cases (see Table 5), the graph clearly shows that more functions have a larger $\phi_{sum}(R_{stat})$ value associated with the point of convergence, τ_C, than with the global optimum, τ_{opt}.

The results from the histogram can be partitioned based on the varying values of $\Omega(f)$. Table 5 shows the average ϕ_{sum} for both the convergence point and the optimum. It is easy to see that, on average, for the sample functions ϕ_{sum} of the converged point is always greater than or equal to the ϕ_{sum} for the optimum. The means are equal for the order 1 functions since all bitwise linearly independent functions converged to the optimum. The means of the ϕ_{sum} values decrease as the $\Omega(f)$ increases. This observation can be explained using the earlier observation that as the degree of nonlinearity is increased functions become less *consistent*. When functions become less *consistent*, the mean ϕ_{sum} for those function will be lower than that of functions that are more *consistent*. Also note the standard deviation of the ϕ_{sum} increases as the $\Omega(f)$ increases. The increased degrees of freedom afforded functions with higher degrees of nonlinearity creates a more diverse set of functions yielding a wider possible range of attainable ϕ_{sum} values.

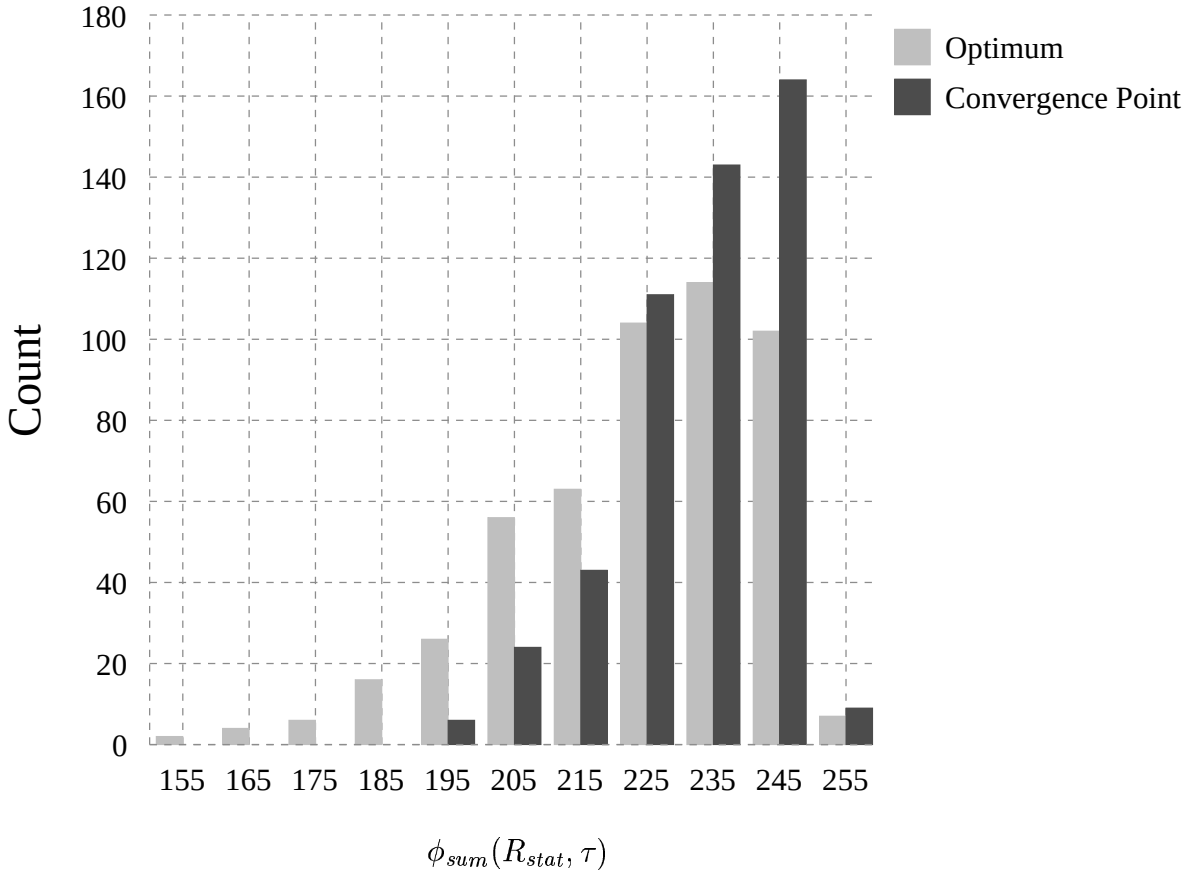

Figure 3: Count of the Number of Functions with particular values of $\phi_{sum}(R_{stat}, \tau_{opt})$ and $\phi_{sum}(R_{stat}, \tau_C)$. More functions have higher values for $\phi_{sum}(R_{stat}, \tau_C)$ than $\phi_{sum}(R_{stat}, \tau_{opt})$.

5.3 The ϕ Spectrum

Comparing the convergence point and the global optimum illustrates that computing ϕ generally results in higher values for the convergence point than the global optimum. However, in order to determine how accurate the ϕ metric is at predicting the convergence point, the ϕ_{sum} at the convergence point needs to be compared to ϕ_{sum} at all other points in the space. We define the ϕ **spectrum** to be a vector in which each element is associated with one of the 2^L possible strings in the population. The vector is indexed by τ and the value of the τ^{th} element is computed using $\phi_{sum}(R, \tau)$ where the string τ is the target string. Each element in the vector therefore represents the degree to which that string could be "supported" as a potential solution by some particular ranking R. With $R = R_{stat}$, the ϕ spectrum provides a much broader view of the biases that exist in the ranking information inherent in the function.

The infinite population model of the simple genetic algorithm is a useful tool for studying the various metrics described here because it simplifies the problem of modeling the genetic algorithm. In Figure 4 we compare the results of the infinite population model and a finite population simple genetic algorithm run on the same functions. The graphs show $\phi_{sum}(R_{stat}, \tau)$ versus $P(\tau, 20)$. The points in the graphs represent each possible string τ in the search space. On the x-axis is the $\phi_{sum}(R_{stat}, \tau)$. On the y-axis is $P(\tau, 20)$, i.e. the string's proportional representation in the infinite population at generation 20.

The leftmost graphs plot the infinite population model run on two randomly chosen functions: one with $\Omega(f) = 2$ and another with $\Omega(f) = 6$. The rightmost graphs corresponds to 5 runs of a finite population simple genetic algorithm with a population of 50 using the same functions used by the infinite population model. In both cases, the probability of crossover is 0.6, no mutation is used and the algorithms were halted at 20 generations. Note that the behavior of the finite population examples was not radically different from the infinite population model; the population size was less than 20% of the total possible number of strings.

The graph shows that the strings with highest proportional representation in the population are also those strings that have the highest $\phi_{sum}(R_{stat}, \tau)$. It also drives home the point that strings with even moderate $\phi_{sum}(R_{stat}, \tau)$ values have little chance of competing for representation in the population. These graphs are universally consistent with the behavior found for all the functions tested, regardless of their order: high values of $\phi_{sum}(R_{stat}, \tau)$ translate into high values of representation. It is also notable that such strings need not have high fitness. They can simply be in hyperplane regions where there are other strings with high fitness, and thus are frequently sampled. Their "nearness" to other high fitness strings translates into higher $\phi_{sum}(R_{stat}, \tau)$ values in the ϕ *spectrum* and higher representation in the population.

When we computed the ϕ *spectrum* for all 600 functions, we observed that the point of convergence did not always produce the highest $\phi_{sum}(R_{stat}, \tau)$ measures in the ϕ *spectrum*, but the $\phi_{sum}(R_{stat}, \tau)$ for the point of convergence was always among the highest values in the ϕ *spectrum*. Bitwise linear functions with $\Omega(f) = 1$ always had the convergence point (the global optimum) ranked highest in the ϕ *spectrum* and were not included in any analysis.

The convergence points of the infinite population genetic algorithm usually rank very high in the ϕ *spectrum* over all functions. In fact, 89.4% of the convergence points fell into the top 5 positions in the ϕ *spectrum* when the infinite population model is used. This experiment suggests that the static metric, $\phi_{sum}(R_{stat}, \tau)$, is a strong indicator of dynamical convergence behavior.

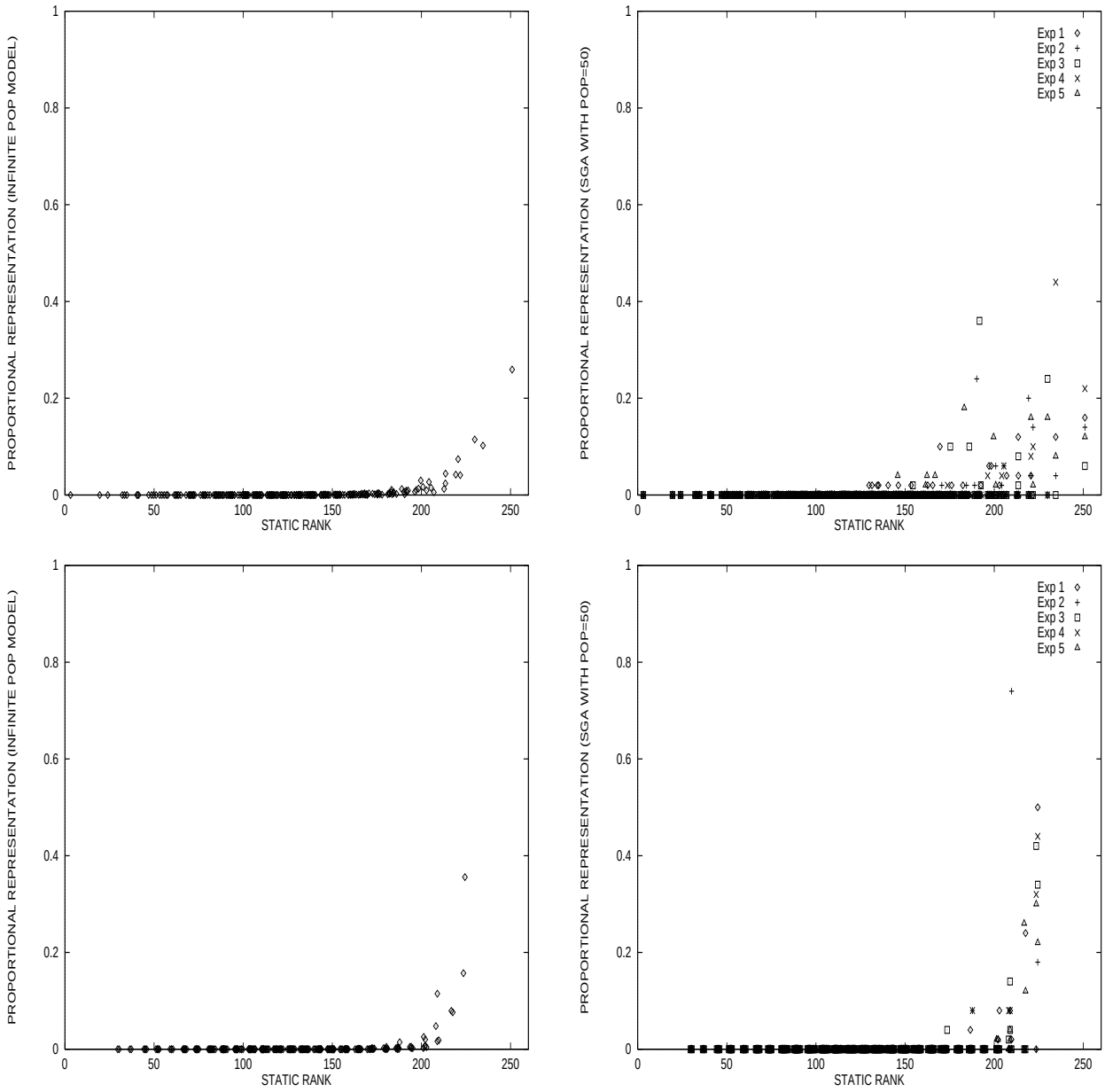

Figure 4: A Comparison of Convergence in Infinite and Finite Populations at Generation 20. Results for the infinite population model behavior are given on the left and results for the finite population genetic algorithm are given on the right.

6 Conclusions

There is no doubt that the role of schema processing in genetic algorithms has been overstated over the last 25 years. There are clearly limits on the genetic algorithms ability to rank schemata because of the kind of constraints that have been outlined in this paper. Yet Holland's original schema theorem is true. So the questions remains, what relationship is there between schemata and genetic algorithms?

The majority of this paper is dedicated to studying the ϕ metric in order to explore the relationships between hyperplanes and the convergence behavior of the simple genetic algorithm. To this end, we refine the notion of consistency as introduced by Whitley et. al. [17]. We relate consistency and the ϕ metric and illustrate that increasing nonlinearity results in a lower ϕ value, and thus a lower degree of consistency. We empirically show that the static ϕ is strongly correlated with the dynamic ϕ which illustrates that there is a relationship between average hyperplane fitness and proportional representation of a hyperplane during the execution of a simple genetic algorithm.

The most compelling results we show are using the ϕ spectrum. These results show that the degree of consistency is (more often than not) higher for the point of convergence in the simple genetic algorithm than for the global optimum. Upon closer examination, the vast majority of the convergence points were found to occur in the top five positions in the ϕ spectrum across 600 functions. These results indicate that the ϕ metric can be a strong indicator of dynamical convergence behavior for a simple genetic algorithm under select conditions.

ACKNOWLEDGEMENTS

This work was supported by NSF Award 0117209. Additional support was provided by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-97-1-0271. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

References

[1] D.B. Fogel and A. Ghozeil. Schema Processing Under Proportional Selection in the Presence of Random Effects. *IEEE Trans Evolutionary Computation*, 1(4):290–293, 1997.

[2] David Goldberg. Genetic Algorithms and Walsh Functions: Part I, A Gentle Introduction. *Complex Systems*, 3:129–152, 1989.

[3] David Goldberg. Genetic Algorithms and Walsh Functions: Part II, Deception and its Analysis. *Complex Systems*, 3:153–171, 1989.

[4] John Grefenstette. Deception Considered Harmful. In L. Darrell Whitley, editor, *FOGA - 2*, pages 75–91. Morgan Kaufmann, 1993.

[5] J. Greffenstette and J. Baker. How Genetic Algorithms Work: A Critical Look at Implicit Parallelism. In J. D. Schaffer, editor, *Proc. of the 3rd Int'l. Conf. on GAs*, pages 20–27. Morgan Kaufmann, 1989.

[6] John Greffenstette. Conditions for Implicit Parallelism. In G. Rawlins, editor, *FOGA -1*, pages 252–261. Morgan Kaufmann, 1991.

[7] R. Heckendorn and D. Whitley. Predicting Epistasis from Mathematical Models. *Evolutionary Computation*, 7(1):69–101, 1999.

[8] R.B. Heckendorn and D. Whitley. A Walsh Analysis of NK-Landscapes. In T. Bäck, editor, *Proc. of the 7th Int'l. Conf. on GAs*, pages 41–48. Morgan Kaufmann, 1997.

[9] J. Holland. Building Blocks, Cohort Genetic Algorithms and Hyperplane Defined Functions. *Evolutionary Computation*, 8(4):373–391, 2000.

[10] John Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[11] H. Kargupta and B.H. Park. Gene Expression and Fast Construction of Distributed Evolutionary Representations. *Evolutionary Computation*, 9(1):43–69, 2001.

[12] R. Poli. Exact Schema Theorem and Effective Fitness for GP with One Point Crossover. In *GECCO-00*, pages 469–476. Morgan Kaufmann, 2000.

[13] Colin Reeves and Christine Wright. An Experimental Design Perspective on Genetic Algorithms. In D. Whitley and M. Vose, editors, *FOGA - 3*, pages 7–22. Morgan Kaufmann, 1995.

[14] M. Vose and G. Liepins. Punctuated Equilibria in Genetic Search. *Complex Systems*, 5:31.44, 1991.

[15] M. Vose and A. Wright. Form Invariance and Implicit Parallelism. *Evolutionary Computation*, 9(3):355–370, 2001.

[16] Michael Vose. *The Simple Genetic Algorithms.* MIT Press, 1999.

[17] Darrell Whitley, Keith Mathias, and Larry Pyeatt. Hyperplane Ranking in Simple Genetic Algorithms. In L. Eshelman, editor, *Proc. of the 6th Int'l. Conf. on GAs.* Morgan Kaufmann, 1995.

[18] L. Darrell Whitley. Fundamental Principles of Deception in Genetic Search. In G. Rawlins, editor, *FOGA -1*, pages 221–241. Morgan Kaufmann, 1991.

[19] L. Darrell Whitley. An Executable Model of the Simple Genetic Algorithm. In L. Darrell Whitley, editor, *FOGA - 2*, pages 45–62. Morgan Kaufmann, 1993.