3

Representation Issues in Neighborhood Search and Evolutionary Algorithms

D. WHITLEY, S. RANA, R. HECKENDORN

3.1 Introduction

Evolutionary Algorithms are often presented as general purpose search methods. Yet, we also know that no search method is better than another over all possible problems and that in fact there is often a good deal of problem specific information involved in the choice of problem representation and search operators. In this paper we explore some very general properties of representations as they relate to neighborhood search methods. In particular, we looked at the expected number of local optima under a neighborhood search operator when averaged overall possible representations. The number of local optima under a neighborhood search operator for standard Binary and standard binary reflected Gray codes is developed and explored as one measure of problem complexity. We also relate number of local optima to another metric, ϕ, designed to provide one measure of complexity with respect to a simple genetic algorithm.

Choosing a good representation is a vital component of solving any search problem. However, choosing a good representation for a problem is as difficult as choosing a good search algorithm for a problem. Wolpert and Macready's (1995) No Free Lunch (NFL) theorem proves that no search algorithm is better than any other over all possible discrete functions. Radcliffe and Surry (1995) extend these notions to also cover the idea that all representations are equivalent when their behavior is considered on average over all possible functions. To understand these results, we first outline some of the simple assumptions behind this theorem. First, assume the optimization problem is discrete; this describes all combinatorial optimization problems–and really all optimization problems being solved on computers since computers have finite precision. Second, we ignore the fact that we can resample points in the space.

The "No Free Lunch" result can be stated as follows:

> The performance of all possible search algorithms is

Genetic Algorithms in Engineering and Computer Science
Editor D. Quagliarelli, J. Periaux, C. Poloni and G. Winter
©1997 John Wiley & Sons Ltd.

exactly the same when averaged over all possible functions.

Abstractly, we can represent an algorithm as an ordering, or permutation, over the points in the search space. We can also view all search algorithms as being deterministic; "stochastic" algorithms are, in practice, deterministic, which we can model by describing a stochastic algorithm as the search method coupled with a specific corresponding random seed. Thus, for any particular problem with a fixed search space of size N, an algorithm is just one of N! ordering of points in the search space. Furthermore, from this operational point of view, all possible representations of a search space is also the same set of N! orderings. Put another way, the N! orderings represent all possible functions that can be constructed from the set of N points that constitute the search space. Thus, there is an isomorphic relationship between all possible search algorithms over this collection of points, all possible functions that can be contructed from this set of points and even all possible representations over this set of points.

Note that if we fix the search algorithm, then these permutations represent all possible functions (or representations) over this set of points. On the other hand, if we fix the function (representation), then these permutations represents all possible search algorithms over this set of points. If we vary both search algorithms and functions, then every search algorithm is searching over the same set of permutations and the performance of all search algorithms is identical. Since this is true for any discrete finite set of points we might wish to pick, all search algorithms are the same over all possible discrete functions.

This way of viewing functions and representations also leads to other observations. For example, when we get stuck in a local optimum, why don't we just transform or remap the representation of the search space so that the point where we are currently stuck is no longer a local optimum? After all, how likely is a local optimum in one representation also a local optimum in another arbitrary representation?

3.2 Representation and Local Optima

For local neighborhood search methods over any discrete function of size N, one can compute the expected number of optima that will occur over all possible functions given a local search operator with a fixed neighborhood of size k. We can also compute the probability that a local optimum will remain a local optimum under an arbitrary change in representation. As will be shown, local optima representing relatively good solutions continue to be local optima with high probability under most representations.

Suppose we have N points in our search space and a search operator that explores k points before making its next move. A point is considered a local optimum if its evaluation is better than all of its k neighbors. Further suppose that the N points in our search space each have unique evaluations. We can sort those points to create a ranking R based on the evaluation function: $R = r_1, r_2, ..., r_n$, where r_1 is the best point in the space and r_n is the worst point in the space. Using this ranking, we can compute the probability $P(i)$ that a point ranked in the i-th position in r is a local optimum under an arbitrary representation of the search space. This probability is given by the formula:

$$P(i) = \frac{\binom{N-i}{k}}{\binom{N-1}{k}} \qquad [1 \leq i \leq (N-k)] \tag{3.1}$$

Proof:
For any point in the search space, there are $\binom{N-1}{k}$ possible neighbors for that point. If the point is ranked in position r_1, then there are $\binom{N-1}{k}$ sets of neighbors that do not contain a point of higher evaluation than the point r_1. Therefore, the point ranked in position r_1 will always be a local optimum under all representations of the function. In the general case, a point in position r_i has only $\binom{N-i}{k}$ sets of neighbors that do not contain a point of higher evaluation than the point r_i. Therefore the probability that the point in position r_i remains a local optimum under an arbitrary representation is $\binom{N-i}{k} / \binom{N-1}{k}$. □

3.2.1 Expected Number of Optima

These probabilities enable us to count the expected number of local optima that should occur in any function of size N. The formula for computing the expected number of times a particular point i will be a local optimum is simply $N! \times P(i)$. Therefore the expected number of optima over the set of all representations with neighborhood size k is:

$$\mathcal{E}(N,k) = \sum_{i=1}^{N-k} P(i) \times N! \tag{3.2}$$

If we want to find the average number of local optima for a single representation instance, we divide $\mathcal{E}(N,k)$ by $N!$, which yields:

$$\mu(N,k) = \sum_{i=1}^{N-k} P(i) \tag{3.3}$$

Finally, using simple counting principles, it can be shown:

$$\mu(N,k) = \sum_{i=1}^{N-k} P(i) = \sum_{i=1}^{N-k} \frac{\binom{N-i}{k}}{\binom{N-1}{k}} = \frac{N}{k+1} \tag{3.4}$$

3.3 A Finite Set of Discrete Functions

In this paper we will focus our attention to the set of functions which can be represented as bit strings. We also restrict our attention to a special set of discretized functions whose domain are the integers 0 to $2^L - 1$ and hence can be represented using an L-bit representation. If the natural domain of the function and its parameters is not these integers, we assume some auxiliary mapping exists onto this integer domain. The auxiliary mapping should map points that are adjacent in the natural domain of the discretized function onto adjacent integers.

We also restrict our attention to functions that have as their range 0 to $2^L - 1$. For discrete functions with different ranges we assume that the values in the natural range of the functions can be sorted; ties can be broken in an arbitrary fashion. Starting at 0, we then map the i^{th} element in the sorted natural range to the integer i. Note that functions with some other range can still be mapped onto this range without changing the basins of attraction or the number and locations of local optimal under a neighborhood search operator. This transformation also produces functions that are processed in an identical fashion by those evolutionary algorithms that use a rank-based selection method, including Tournament Selection and Truncation Selection.

3.3.1 Gray and Binary Representations

The most commonly used representations among evolutionary algorithms that use bit representations are standard Binary and standard binary reflected Gray encodings. There are in fact many Gray codes. A "Gray code" is any bit representation that has the property that adjacent integers are also adjacent neighbors in the bit-neighborhood hypercube graph defined over the set of bit strings of length L. A Gray code also maps onto a Hamiltonian circuit in the hypercube. Every Hamiltonian circuit in the hypercube can be used to form a Gray code by assigning the integers in sequence to the vertices along the Hamiltonian circuit. In fact, we can start the assignment at any arbitrary point along the circuit, so in fact, each circuit represents up to 2^L Gray codes under a simple shift operation which moves the first integer (e.g. 0) to some arbitrary point along the Hamiltonian circuit.

A standard reflected Gray code representation can be constructed by taking a standard Binary representation of an integer, shifting it 1 bit position to the right, then applying the *exclusive-or* operator to the original string and the shifted string. The first L bit represent the Gray code. For example, we compute the Gray code of several 4 bit strings as follows:

```
Binary   0000     0100     0001     1111     1010
          0000     0100     0001     1111     1010
         ----     ----     ----     ----     ----
Gray     0000     0110     0001     1000     1111
```

The conversion from Binay to standard reflected Gray can also be performed using a matrix operation. There exists an $L \times L$ matrix G_L that maps a string of length L from Binary to reflected Gray representation. There also exists a decoding matrix D_L that maps the reflected Gray back to its original representation.

Note that a "bit string" is not inherently a Binary or a reflected Gray representation. It is only when we produce a mapping between the set of integers and the set of bit strings can we identify it as a particular representation.

The following are the G_4 and D_4 matrices.

$$G_4 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad D_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In higher dimensions the G_L matrix continues to have 1 bits along the diagonal and the upper minor diagonal, and D_L has 1 bits in the diagonal and the upper triangle. Assuming the L-bit column vector x is a standard Binary representation, binary matrix multiplication (i.e. matrix multiplication mod 2), $x^T G_L$ produces a Gray coding.

For example, if $x^T = [1\ 0\ 0\ 1]$, then $(x^T G_4) = [1\ 1\ 0\ 1]$.

Assuming the L-bit column vector x is a standard reflected Gray representation and $x^T = [1\ 1\ 0\ 1]$, then $(x^T D_4) = [1\ 0\ 0\ 1]$ produces the corresponding Binary representation of the corresponding integer.

An interesting property of the G matrix is that all reorderings of the columns of the matrix produce a matrix that is also a Gray transformation corresponding to a different Hamiltonian circuit through the hypercube graph. Thus, there may be as many as $L! * 2^L$ different Gray codes, since all 2^L shifts along each Hamiltonian circuit is also a Gray code. There may of course be redundancy in this count. The exact number of possible Gray representations is an open question.

Empirically, Gray coding usually results in better performance than using a standard Binary coding when applying various forms of genetic algorithms and neighborhood search algorithm using common test functions (Caruana and Schaffer, 1988; Mathias and Whitley, 1994a; 1994b). It has long been known that Gray coding removes *Hamming cliffs* (Caruana and Schaffer, 1988). A Hamming cliff corresponds to a pair of numbers in numeric/integer space whose bit representations are complementary. Thus in a four bit space, 7 and 8 are adjacent in numeric space but their representations as bit strings are 0111 and 1000.

Typically, when generating a bit representation for a function, there exists some mapping of the domain of the function onto a set of integers and the integers are then mapped to bit strings. On the one hand, it makes sense that one would like to preserve the adjacency of the original numeric/integer representation of the function in the bit representation.

The graph on the right in figure 3.1 shows that all the adjacency relationships found in the numeric representation are preserved in Gray space. On the left, one can see that only half of the adjacent edges found in numeric representations are preserved in Binary space; for representations of arbitrary length one can show by induction (Whitley et al., 1996) that it continues to be true that half of the edges from the numeric representation are preserved under the Binary representation.

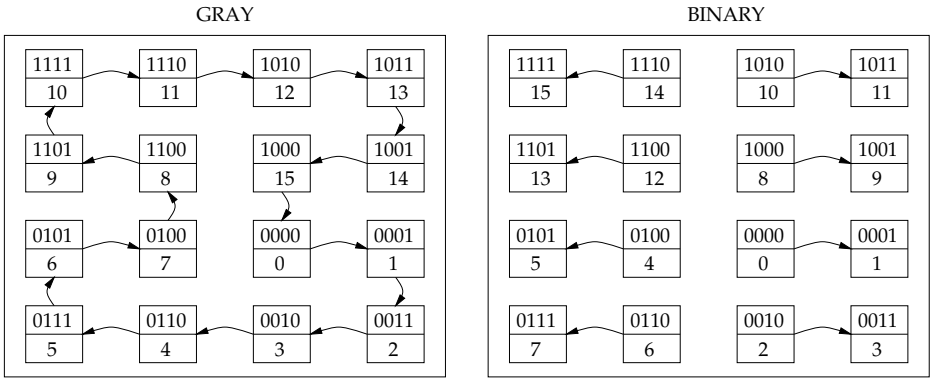

Figure 3.1 Adjacency in 4-bit Hamming space for Gray and Binary encodings.

3.3.2 NFL and Equivalence Chains under Gray and Binary

There exists a set of equivalences for the G_L and D_L matrices such that:

$$\text{For } k = 1..(L-1), (G_L)^k = (D_L)^{(L-k)} \text{ and } (G_L)^L = (D_L)^L = I_L$$

where I_L is the $L \times L$ identity matrix. For simplicity, we will denote $(D_L)^k$ as D_L^k. In **4-bit space**, $G_4^1 = D_4^3$, $G_4^2 = D_4^2$, $G_4^3 = D_4^1$ and $G_4^4 = D_4^4 = I_4$; also note that $G_4 \cdot D_4 = I_4$.

Using these equivalences, it is possible to construct what we will refer to as *chains* of functions. Recall that F is a finite set of functions representing all possible permutations over 2^L points in the search space. Let C represent a coding (i.e. representation) of bit strings to this same set of points.

Select any function $f_1 \in F$. Convert the integers corresponding to all possible input parameter sets in the domain of function f_1 to standard Binary strings. The function f_1 will be the first function in the chain. Let the Binary strings representing the encoded domain of f_1 be representation (i.e., coding) c_1 in the chain, where $c_1 \in C$. Each string in c_1 has an evaluation such that the evaluation of string s in c_1 is given by f_1 where f_1 is the evaluation of function f_1 for input parameter set i and string s is the Binary representation of parameters i. Apply the Gray code transform G to the all strings in c_1. This generates a new representation c_2. Strings in c_2 inherit their evaluation from c_1. This means that strings in c_2 can be degrayed, then converted to a set of integers representing the set of input parameters of f_1. But c_2 is also the Binary representation of some function. Since there is an evaluation associated with every string in c_2, decoding strings in c_2 as Binary strings generates a new function f_2, such that the Gray code of f_1 is c_2 and the Binary code of f_2 is c_2. Continuing this process for each few function we generate a chain of functions which we can represented as follows:

Function	f_1	f_2	f_3	f_4	$f_5 \ldots$	
Binary	$c_1->$	$c_2->$	$c_3->$	$c_4->$	$c_5->$	$c_6->$
Gray	$c_2->$	$c_3->$	$c_4->$	$c_5->$	$c_6->$	$c_7->$

Thus, if we take all strings in function f_1, convert them to binary strings, we obtain representation c_1. If we Gray those strings using matrix G, then we obtain representation c_2. (We assume that G is the matrix for reflected Gray code, but chains can also be formed using any other Gray code transformation matrix.) If we apply Gray twice to c_1, (i.e., if we Gray all the bit strings in c_2) we obtain representation c_3. Under this system, the evaluations associated with the strings in representation c_i is also carried forward; thus Binary and Gray codings are just different mappings from the space of functions to the corresponding space of bit representations. If we then decode the strings of c_3 as Binary strings we obtain function f_3; if we decode the strings of c_3 using DeGray and then convert the resulting strings from Binary to integers, then we obtain function f_2.

For any representation of L bits, the set of functions and representations begin to repeat; there are at most 2L functions (or representations) in any chain. Let $L' = 2^{\lceil log_2 L \rceil}$; note $L \leq L' < 2L$. The Gray and DeGray transform matrices are such that $G_L^{L'} = D_L^{L'} = I_{L'}$. Thus applying G to c_i L' times yields c_i, causing the chain to repeat.

For multi-parameter N-dimensional functions, the length of the chains are at most $2l_m$, where l_m is the maximum number of bits used to code any single parameter. All cycles are synchrononized because if ${l_1}' \geq {l_2}'$ then ${l_1}'/{l_2}' = k$ yields an integer which implies that parameter 2 cycles exactly k times for each cycle of parameter 1.

A No Free Lunch result holds for every chain of functions. Every function in the chain has its Gray and Binary representation in the chain; every representation c_i in the chain is the Binary code of function f_i and the Gray code of function f_{i+1}. It follows that a general No Free Lunch result holds for Gray and Binary representations over all function–since the set of all functions can be decomposed into sets of chains and No Free Lunch holds for every chain.

3.4 Gray Beats Binary: An Example

To find differences between Gray representations and Binary representations, we must partition the set of functions into special subsets–and then discard some subset in which we are not interested. Clearly, we would like such subsets to be indicative of some practial notion of problem complexity.

We look at all functions in F that have K local optima in their integer/numeric defining neighborhood topology. We then ask the following: for functions with exactly K local optima, which induces fewer local optima in the resulting sets of bit representation, Gray or Binary? Over all possible K the No Free Lunch theorem holds, *but not for individual values of* K.

To examine this question we enumerated all possible permutations over 8 values; this corresponds to all possible functions which can be represented with 3 bits given the restrictions imposed on F. In the following table we look at how many functions have 1, 2, 3 or 4 minima in F. We next look at how many total optima are inducted

under Gray and Binary over all functions with 1 minima, those with 2 minima, etc.

There are of course 8! or 40,320 functions in this case. All that No Free Lunch tells us is that the number of minima for Gray or Binary representations must be equal. In this case, there are a total of 80,640 minima under both Gray and Binary.

We next look at each individual function to see when one representation induces fewer minima than another.

K	# of F with K minima	# of Optima in Gray	# of Optima in Binary
1	512	512	960
2	14,592	23,040	27,344
3	23,040	49,152	49,392
4	2,176	7,936	2,944
TOTAL	40,320	80,640	80,640

The next table shows that of the 512 functions with 1 minima in F, there are 448 cases where the Gray coded versions of these functions induce fewer minima than the Binary represenations, but no cases where Binary induces fewer minima than Gray. There are 64 cases where Gray and Binary induce the same number of minima.

K	Gray-fewer optima	Binary-fewer optima	Ties
1	448	0	64
2	6384	2176	6032
3	7088	6704	9248
4	0	2160	16

For higher dimensional functions enumeration quickly becomes impossible. But it is easy to prove that for all represenations of functions in F defined over L bits, Gray always induces fewer minima that Binary over the set of functions with a single optimum (for a proof, see Whitley and Rana, 1997). For problems with encodings up to 6 bits we can also prove that Binary induces fewer minima over worst-case problems when compared against Gray. By worst-case, we mean problems that have N/2 minima in a search space of N points. When bit representations are involved, $N = 2^L$. Specifically in the above table, we consider "worse-case" functions to be those with K = 4 optima.

Both tables make it clear that there is a strong bias under Gray and Binary encodings. Gray coding is more tightly coupled to the original numeric/integer representation of the function. As a side effect, Gray is biased toward producing fewer local minima for functions that have few minima in the original numeric/integer representation and producing a larger number of minima for functions that have large number of minima in the original numeric/integer representation of the function. This implies that 1) if we care about number of local minima and 2) if we are more likely to work with functions that have fewer local minima in the numeric/integer function representation, then *on average* Gray coding is a better representation over this subset

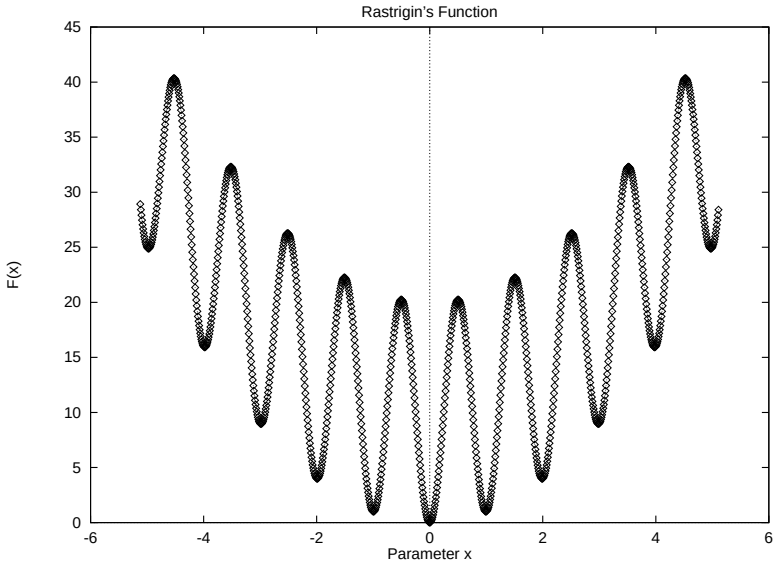

Figure 3.2 Rastrigin's function encoded using 1024 data points on a single dimension.

of functions. "Better" is defined in terms of inducing fewer optima in the corresponding bit representations. In fact, we have found that common test problems do indeed have fewer minima than the expected number over all possible functions and representations under local operators with neighborhood size L, the length of the bit representation.

Assuming Binary is indeed "better" than Gray over worst-case problems, it follows that if we remove these worst-case problems from consideration, Gray is "better" than Binary over all other remaining functions. We have a good deal of evidence to support the conjecture that for all L-bit functions, Binary is better than Gray over all functions with exactly $2^L/2$ optima in the numeric/integer representation; if follows that Gray is better than Binary over all other functions. In the next section we empirically establish a relationship between number of local optima and problem difficulty for a simple genetic algorithm.

3.5 Counting Optima and Common Test Problems

Given a search space of N points and a neighborhood search operator over a neighborhood of size k, do common test problems have more or less minima than the expected number? We can answer this question for small problems. We can also answer this question for test problems which are a linear combination of subproblems.

Rastrigin's function is illustrated in figure 3.5 as a 1-D function.

$$f(x_i \mid_{i=1,N}) = (10N) + \left[\sum_{i=1}^{n} \left(x_i^2 - 10cos(2\pi x_i)\right)\right] \qquad (3.5)$$
$$x_i \in [-5.12, 5.11]$$

In numeric space with a neighborhood size of two (looking only at a point on either side of the current point) there are 11 local minima along any dimension. This also means that there are 11^N possible local minima in an N-dimensional version of the function.

When we impose a 10 bit encoding on the function, however, the number of locally optimal points that occur will change. Using reflected Gray coding there are only 5 locally optimal points in Rastrigin's function. And while there were 11 optima in numeric space, under standard binary encoding there are 19 locally optimal points. The expected number of local optima for an arbitrary function/representation under a L neighborhood search operator over 1024 points is 93. Extending values to an N-dimensional problem, we obtain 5^N for Gray versus 11^N for the numeric representation and 19^N possible local optima for standard Binary.

We also analyzed 2 other common test functions. The following function was introduced by Schwefel.

$$f(x_i \mid_{i=1,N}) = \sum_{i=1}^{N} -x_i sin(\sqrt{\mid x_i \mid}) \qquad x_i \in [-512, 511]$$

It has 8 local optima in the numeric/integer representation, 5 local optima under standard binary reflected Gray code and 12 local optima under a Standard Binary representation.

The following is Griewangk's function.

$$f(x_i \mid_{i=1,N}) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} (cos(x_i/\sqrt{i})) \qquad x_i \in [-512, 511]$$

It is interesting that this function has 163 minima in the numeric/integer representation. In this case, Gray induces 22 optima and Binary results in 18 optima. In general, Gray coding has an advantage when the number of minima is the numeric/integer representation is relatively small; Binary will have an advantage when the number of minima is large in the numeric/integer representation. Griewangk's function appears to be consistent with this general trend, but one cannot apply this principle to individual functions. The only think we can say about specific functions is that Gray coding ensures that the number of minima in the Gray code will be less than or equal to the number of minima in the numeric representation. Also, the expected number of minima in the numeric representation (i.e., a neighborhood of size 2) is 341, so while this function has more minima than Rastrigrin's or Schwefel's, it still has less than the average over all possible functions composed of 1024 points using a neighborhood size of 2.

A more detailed consideration of these functions is given by Whitley et al. (1996) and further representation analysis is given by Rana and Whitley (1997).

3.5.1 Local Optima and Other Measures

The number of local optima in the search space is related to other measures of complexity, especially the notion of "correlated search spaces" and "path length." Path length can be defined as the average distance from any point in the search space to a local optimum under a steepest ascent or a next-ascent local neighborhood search. It is rather intuitive that as the number of local optima increase the set of local optima on average have smaller basins of attraction and average path length is shorter. Functions with only a few optima will also have correspondingly longer average path lengths.

The connection between a metric that measures the degree of correlation in a search space or "correlated fitness landscape" is perhaps less direct. Assume we compute correlation using a random walk in the search neighborhood. We then correlate the difference in evaluations between points in the search space with the distance between points under the neighborhood operator. A stronger correlation generally translates into more smoothness along the paths under a random walk, and this translates into longer path length before reaching a local optimum. Of course, this isn't always true. One can construct a 1-D function where every other point is a local optimum, but where there is still a strong trend in the constructed function for points that are nearer to one another to have more similar evaluations. Nevertheless, over all possible functions it is reasonable to expect that functions with higher correlation will on average have fewer local optima.

We still might ask what local optima and path length and correlation have to do with evolutionary algorithms?

3.5.2 Optima and Hyperplanes

In order to better understand the relationship between the number of optima in the search space and problem complexity for a genetic algorithm, we generated a number of test problems with varying numbers of local optima. We then use a metric designed to measure the difficulty of a problem for a genetic algorithm in terms of hyperplane information.

It has long been argued that simple genetic algorithms as defined by Holland (1975) and Goldberg (1989) obtain their search power from the ability to sample subsets of strings in the space corresponding to hyperplanes of the search space defined over shared bit patterns known as schema.

Whitley, Mathias and Pyeatt (1995) introduce the ϕ metric to measure the consistency of an arbitrary ranking of hyperplanes in a partition with respect to a target string. Do those hyperplanes that have the highest fitness consistently lead a genetic algorithm to focus on a particular subregion of the search space? Or are the bit patterns inconsistent across those hyperplanes with the highest fitness–thereby providing conflicting information to the genetic algorithm about where to sample?

Heckendorn, Whitley, and Rana (1995) provide the following characterization of the ϕ metric. Consistency of a hyperplane partition is measured by ranking the hyperplanes in a partition, then measuring the consistency of the ranking with respect to a target string. In this case we will use the global optimum as the target string. By measuring consistency over every possible partition we obtain an overall measure of

consistency for a function.

The degree of ranking for an order k partition denoted by π containing hyperplanes $\xi_1, \xi_2, ...\xi_{2^k}$ is defined as follows:

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

where $\mathcal{R}$ is a ranking function of over a set of hyperplanes in π (note this is different from the R used as a ranking over all point in the search space), τ is a target string and Pred(*expression*) returns a 1 if the expression is true and 0 if it is false. $M(\xi, \tau)$ 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 $\tau = \vec{1}$ and we rotate the search space using exclusive-or to locate the target string at $\vec{1}$ (see Whitley 1994). We thus often denote $M(\xi, \vec{1})$ as $M(\xi)$ and $\phi(\mathcal{R}, \pi, \vec{1})$ as $\phi(\mathcal{R}, \pi)$.

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

For the specific case of when $\mathcal{R}$ is a ranking based on the static average fitness of all strings in the hyperplanes, we say that ϕ measures the **degree of static ranking** for a partition.

The maximum possible ϕ value for a partition π of order k can be computed 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 $\mathcal{R}$ 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.

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

Table 3.1 presents an example by Heckendorn, Whitley, and Rana (1995) which shows an example computation of $\phi(\mathcal{R}, \pi)$ for two different ranking functions $\mathcal{R}_1$ and $\mathcal{R}_2$ 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 $\mathcal{R}$ from greatest to least to aid hand verification of the results at the bottom of the table. Notice that $\mathcal{R}_1$ yields a maximum possible ϕ value and therefore, there can be no misleading hyperplane information (e.g., deception) in this partition (see Goldbergf 1987; Whitley 1991).

$\mathcal{R}_2$ provides a good 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:

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

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

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

giving a value of 2 which is added to the sum. The other terms are similarly computed to produce the final calculation, $\phi(\mathcal{R}_2, \pi) = 20$, which is normalized using $\phi_{max} = 30$ to yield $\widehat{\phi}(\mathcal{R}_2, \pi) = 0.66\overline{6}$.

ξ_i	$\mathcal{R}_1(\xi_i)$	$M(\xi_i)$	ξ_i	$\mathcal{R}_2(\xi_i)$	$M(\xi_i)$
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
$\phi(\mathcal{R}_1, \pi) = 30$			$\phi(\mathcal{R}_2, \pi) = 20$		
$\widehat{\phi}(\mathcal{R}_1, \pi) = 1.0$			$\widehat{\phi}(\mathcal{R}_2, \pi) = 0.66\overline{6}$		

Table 3.1 Computations of ϕ for two example rankings.

In the next section we relate ϕ to the complexity of functions with varying numbers of optima.

3.6 Some Test Functions and Results

We generated test functions with varying numbers of optima by randomly picking a set of Z points to be minima and another set of Z points to be maxima. These can be thought of a 1-dimensional functions with a neighborhood size of 2. The range of these functions corresponding to the output evaluations is 0 to $2^L - 1$. Clearly, 0 must belong to the set of Z minima and $2^L - 1$ must belong to the set of maxima. After having picked the sets of minima and maxima, there are then constraints on where other points in the space can be placed in the function. We place the remaining points in the space randomly subject to the contraint that we cannot change the number of minima in the space. For example if the point with evaluation $2^L - 2$ is not also a maxima, it must be adjacent to the point $2^L - 1$. While minima and maxima are chosen randomly, this is also subject to constraints. For example, the points with evaluation $2^L - 1$ and $2^L - 2$ can never be minima and the points with evaluation 0 and 1 can never be maxima. We generated functions over 256 points, which map to an 8 bit representation.

We generated 1-dimensional (1-D) functions with 1, 5, 10, 15, ... 85, 90 minima (i.e. 5 to 90 optima in increments of 5). For a function over 256 points, the maximum

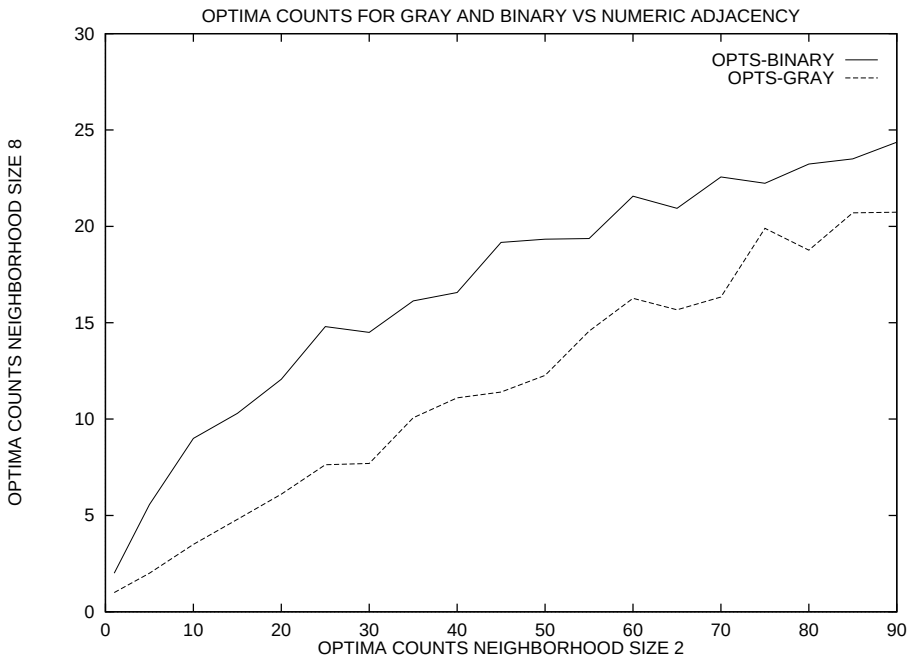

Figure 3.3 Number of Optima (Minima) under Gray and Binary. The integer neighborhood is of size 2 and the bit neighborhood is of size 8.

number of optima under an integer neighborhood of size 2 is 128 and the expected number of optima averaged over all functions is 85. Thus, we are working with subsets of all possible functions over 256 points where we might expect Gray codes to have an advantage. Figure 3.3 shows the number of optima that results for these functions under the binary reflected Gray and standard Binary encodings. Samples were taken over 30 functions for each optima count. As can be seen, both encodings follow the same trend in that both show an increasing number of minima in the 8 bit Hamming space representation as the number of minima also increases in the generated functions. Still Gray is "better" in the sense that it consistently yields in fewer minima than Binary over these subsets of functions.

We next look at the relationship between the ϕ metric and number of optima in the numeric function (i.e., the 1-dimension function with neighborhood size 2). Specifically we compare the ϕ values for the generated functions against the number of optima. Since ϕ is computed with respect to a bit representation, it is computed for both a reflected Gray encoding and a standard Binary encoding. Again, the number of minima on the x-axix is the number in the original 1-dimensional numeric functions. For the binary encodings, figure 3.4 suggests that there does not appear to be a relationship between the resulting ϕ value and the number of optima in the search space. However, using a reflected Gray encoding, figure 3.5 suggests a relationship between ϕ and the number of optima in the functions. Under the Gray encoding, functions with fewer

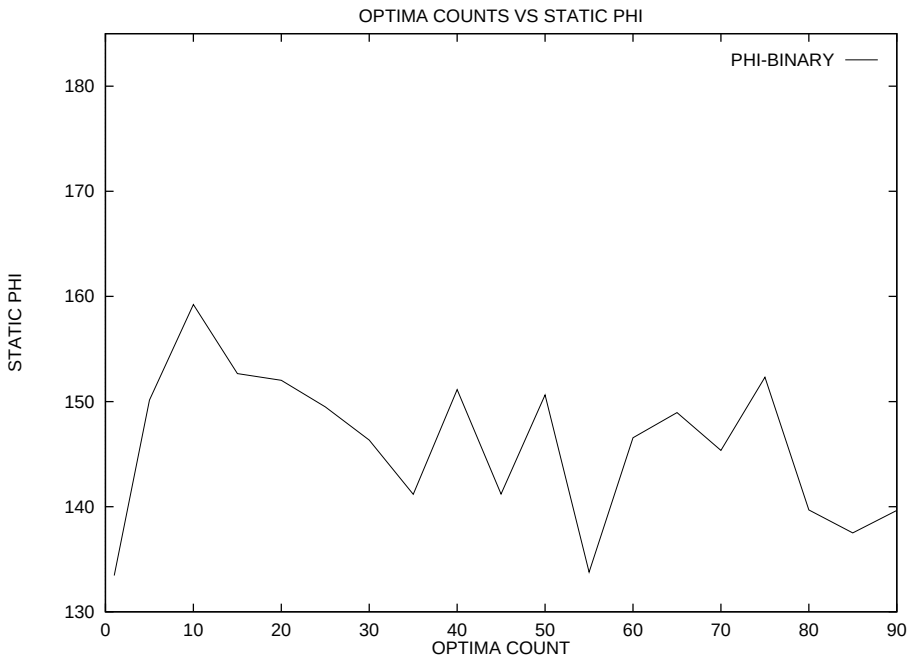

Figure 3.4 Variation in the ϕ metric for a Binary problem encoding as a function of the number of optima under an integer neighborhood representation. For a Binary coding no clear trend emerges. The results are averaged over 30 sample functions for each optima count.

optima have a higher ϕ value, which suggests they are are more organized in terms of providing consistent hyperplane information. Those functions with fewer minima should be easier for a simple genetic algorithm to process. As the number of optima increase, the ϕ value decreases, indicating that there is more conflict in the various hyperplane competitions, which should make these representations more difficult for a genetic algorithm to process.

This is the first tangible evidence to suggest that there is a relationship between the number of optima in a numeric/integer representation of a function and the structure of a function as expressed in terms of hyperplane information. These results also suggest that this relationship holds under a Gray encoding, but does not necessarily hold under a Binary encoding.

To take this investigation one step further we ran 2 search algorithms on 10 specific functions with different numbers of optima as shown in Table 3.6. In particular we ran Goldberg's Simple Genetic Algorithm (1989) using a population of size of 200 with a crossover rate of 0.6 and a mutation rate of 0.01. Tournament selection was used in place of fitness proportionate selection. Elitism was used so that the best solution was carried forward each generation. Davis' (1991) Random Bit Climber is an L-bit neighbor local search that tests the L neighbors by flipping the L individual bits

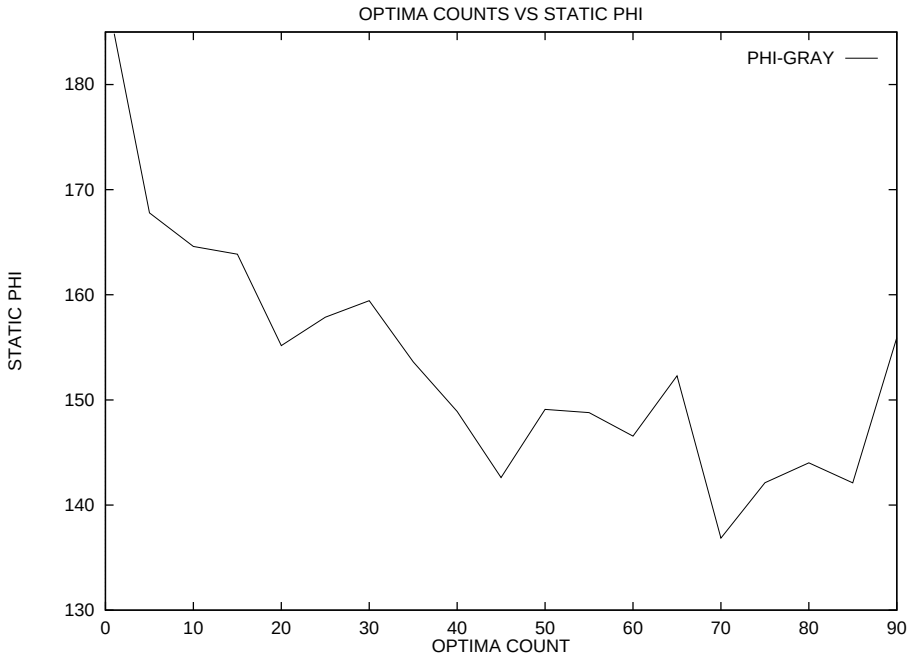

Figure 3.5 Variation in the ϕ metric for Gray coding as a function of the number of optima under an integer neighborhood representation. For the Gray coding, *phi* decreases as the number of minima increase. Results are averaged over 30 sample functions for each optima count. This suggests that functions with fewer optima are easier to solve that functions with more optima when a Gray representation is used.

Opt Count			SGA			RBC			
1-D	Gray	ϕ	Solution	No.	Evals	Solution	No.	Evals	Res
5	2	126.670	0.06401	13	154194	0.04864	11	210976	895
10	2	199.529	0.00656	29	10932	0.00000	30	2108	4
20	8	182.894	0.00000	30	53156	0.11948	0	–	1265
30	7	157.175	0.06400	21	88736	0.03475	19	244829	920
40	12	95.660	0.10752	0	–	0.11443	0	–	1599
50	15	141.428	0.02515	2	374250	0.08748	0	–	1590
60	18	169.428	0.00131	0	–	0.08087	0	–	1512
70	20	114.029	0.18684	0	–	0.26385	0	–	1783
80	22	131.882	0.04669	0	–	0.14456	0	–	1727
90	24	150.589	0.08496	0	–	0.15166	0	–	1927

Table 3.2 Results for sample functions with varying number of local optima in numeric 1-D space. (Solution=Average Solution Found, No.=Number solved out of 30, Evals=Average Number of Evaluations for Solved Problems) and Res=Number of Restarts under RBC.

in random order (determined by a random permutation) and takes the first available improving move. The number of optima is still measured in the numeric representation, but the bit representation was generated using reflected Gray coding. The actual difficulty of these functions thus depends a great deal on the specific details of the bit representation which results.

To generate a larger function we concatenate 10 copies of each function and sum the resulting evaluations. This of course results in separable functions, and we are well aware of the limitations of using separable test functions (Whitley, Mathias, Rana and Dzubera, 1995; 1996). However, this does allow us to control the number of optima while also generating larger test problems. It also gives us a way to compute ϕ for the individual primitive subfunctions, since the functions/subfunctions must be enumerable in order to compute ϕ. Thus, the ϕ values we compute do not measure any interactions between variables, but in this case this should not be a problem since summation over multiple copies of the original 1-D functions cannot introduce any additional higher order hyperplane interactions.

Table 3.6 shows the number of optima under the 1-dimensional (1-D) numeric/integer representation as well as the number of optima under a Gray encoding. The ϕ value associated with the Gray encoding of each function is also shown. Because there are 255 possible partitions of the search space for an 8 bit function ($2^L - 1$), the ϕ value can range from 0 to 255. A *fully deceptive function* (Whitley 1991) is such that the hyperplane with the highest average fitness in every partition leads away from the global optimum; thus, a function with ϕ of 0 would be fully deceptive. A function with the maximum possible ϕ of 255 has no deception. Thus, a high ϕ value should indicate that a function will be easier to solve by the Simple Genetic Algorithm. Also, note that since every 1-D subfunction ranges over values between 0 and $2^L - 1$ and each

test function is 10 dimensional, the ranges over all the test functions are comparable. Averages are calculated over 30 runs per function.

Results are shown for the simple genetic algorithm and random bit climbing. Our expectation is that SGA should be sensitive to the ϕ metric and that RBC should be sensitive to the number of optima in the search space. The ϕ metric indeed appears to be a relatively good predictor of the performance of the Simple Genetic Algorithm: there is a -0.7519 correlation between the ϕ value and the average solution found. In other words, high ϕ values would appear to be a good predictor of a low average solution. (The sign of the correlation really isn't important; changing these problems from minimization to maximization will also flip the sign of the correlation values.)

We also computed the correlation between RBC's average solution and ϕ, and found that the correlation was -0.5280. The correlation between the behavior of SGA and ϕ is clearly stronger, but this correlation still suggests a relationship between ϕ and the behavior of RBC. This also underscores the fact that there is an overall similiarity in solution quality under SGA and RBC. The one exception to this seems to be the 1-D function with 20 minima, which SGA solves every time and RBC fails to solve. Still, the ϕ value indicates that the function with 10 optima in 1-D is the easiest function in the set, while the function with 5 optima in 1-D is actually more difficult. Both functions have 2 minima in Gray space, yet both algorithms have more difficulty solving the function with 5 optima. The ϕ value is lowest for the functions with 40 and 70 optima in 1-D and indeed these appear to be two of the hardest functions for both SGA and RBC.

There is also a general trend in the data such that as the number of optima increases in the numeric 1-dimensional version of these function, the number of optima also generally increases in the Gray representation. In addition, across the set of problems there is a trend for problems to be more difficult to solve as the number of local optima increase.

3.7 Discussion and Conclusions

Theoretically, since every search algorithm is different there is no single answer as to what constitutes a hard search problem. What is hard for one algorithm may be easy for another method. On the other hand, search algorithms have been refined using a relatively small set of common test problems and it is reasonable to assume that search algorithms have been developed and tuned to solve a relatively small subclass of functions drawn from the space of all possible functions.

This paper focuses on "number of local optima" as one measure of problem complexity which, at least in definition, is somewhat algorithm independent. We also present evidence to suggest that the computational behavior of a genetic algorithm can be sensitive to number of local optima. One can certainly construct problems that have few local optima, and yet which are difficult. But when averaging over sets of functions, having more local optima generally appears to contribute to problem complexity.

References

Caruana, R., and Schaffer, J. (1988) Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms. In *Proceedings of the Fifth International Conference on Machine Learning.* Morgan Kaufmann.

Davis, L. (1991) Bit-Climbing, Representational Bias, and Test Suite Design. In Booker, L., and Belew, R., eds., *Proc. of the 4th Int'l. Conf. on GAs*, 18–23. Morgan Kauffman.

Goldberg, D. (1987) Simple Genetic Algorithms and the Minimal, Deceptive Problem. In *Genetic Algorithms and Simulated Annealing*, pp: 74–88. L. Davis, ed., Pitman.

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

Heckendorn, R., Whitley, D., and Rana, S. (1996). Nonlineary, Hyperplane Ranking and the Simple Genetic Algorithm. *Foundations of Genetic Algorithms 4.* Morgan Kaufmann.

Holland, J. (1975) Adaptation in Natural and Artificial Systems. Michigan University Press. 1992 edition, MIT Press.

Mathias, K.E., and Whitley, L.D. (1994a) Changing Representations During Search: A Comparative Study of Delta Coding. *Journal of Evolutionary Computation* 2(3):249–278.

Mathias, K.E., and Whitley, L.D. (1994b) Transforming the Search Space with Gray Coding. In Schaffer, J. D., ed., *IEEE Int'l. Conf. on Evolutionary Computation*, 513–518. IEEE Service Center.

Radcliffe, N.J. and Surry, P.D. (1995) Fundamental limitations on search algorithms: Evolutionary computing in perspective. In *Lecture Notes in Computer Science*, Vol 1000, J. van Leeuwen, ed., Springer-Verlag.

Rana, S. and Whitley, D. (1997). Bit Representations with a Twist. *International Conference on Genetic Algorithms.* Morgan Kaufmann

Whitley, D. and Rana, S. (1997) Representation, Search and Genetic Algorithms. Proc. National Conf. on Artificial Intelligence (AAAI-97).

Whitley, D., Mathias, K., Rana, S. and Dzubera, J. (1996). Evaluating Evolutionary Algorithms. *Artificial Intelligence, 85:* 1-32.

Whitley, D., Mathias, K., Rana, S. and Dzubera, J. (1995). Building Better Test Problems. *International Conference on Genetic Algorithms.* Morgan Kaufmann

Whitley, D. (1994) A Genetic Algorithm Tutorial. *Journal of Statistics and Computing.* Vol 4. pp: 65-85.

Whitley, D., Mathias, K. and Pyeatt, L. (1995). Hyperplane Ranking During Genetic Search. *International Conference on Genetic Algorithms.* Morgan Kaufmann

Whitley D. (1991) Fundamental Principle of Deception in Genetic Search. *Foundations of Genetic Algorithms -1-,* G. Rawlins, ed., pp: 221-241, Morgan Kaufmann.

Wolpert, D.H., and Macready, W.G. (1995) No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute.

©1997 John Wiley & Sons Ltd.