Properties of Gray and Binary Representations

Jonathan Rowe J.E.Rowe@cs.bham.ac.uk
Computer Science Department, University of Birmingham, Birmingham B15 2TT, UK

Darrell Whitley whitley@cs.colostate.edu
Laura Barbulescu laura@cs.colostate.edu
Jean-Paul Watson watsonj@cs.colostate.edu
Department of Computer Science, Colorado State University, Fort Collins, Colorado 80523, USA

Abstract

Representations are formalized as encodings that map the search space to the vertex set of a graph. We define the notion of bit equivalent encodings and show that for such encodings the corresponding Walsh coefficients are also conserved. We focus on Gray codes as particular types of encoding and present a review of properties related to the use of Gray codes. Gray codes are widely used in conjunction with genetic algorithms and bit-climbing algorithms for parameter optimization problems. We present new convergence proofs for a special class of unimodal functions; the proofs show that a steepest ascent bit climber using any reflected Gray code representation reaches the global optimum in a number of steps that is linear with respect to the encoding size. There are in fact many different Gray codes. *Shifting* is defined as a mechanism for dynamically switching from one Gray code representation to another in order to escape local optima. Theoretical results that substantially improve our understanding of the Gray codes and the shifting mechanism are presented. New proofs also shed light on the number of unique Gray code neighborhoods accessible via shifting and on how neighborhood structure changes during shifting. We show that shifting can improve the performance of both a local search algorithm as well as one of the best genetic algorithms currently available.

1 Introduction

Choosing a good representation is a vital component of solving any search problem. Wolpert and Macready's No Free Lunch (NFL) theorem (Wolpert and Macready, 1995, 1997) proves that no search algorithm is better than any other over all possible discrete functions. Radcliffe and Surry (1995) extended these notions and showed that all representations are equivalent when their behavior is considered over all possible functions.

The evolutionary computing community has long been concerned with the use of binary encodings, the use of Gray codes and the use of real valued representations for parameter optimization problems. However, arguments that one representation might be better than another have largely been based on limited empirical data, practical experience and historical bias.

This paper is primarily concerned with the use of Gray and Binary representations for parameter optimization. Most of the results relate to local optima as defined under local search, but some of the results also relate to genetic algorithms.

Given a set Ω and a function $f : \Omega \rightarrow \mathbb{R}$, we seek to maximize or minimize f. A

local search algorithm for this problem is defined by assigning a neighborhood structure to Ω. That is, for each point $x \in \Omega$, we assign a subset $N(x) \subseteq \Omega$, called the *neighborhood* of x. The local search algorithm (for minimizing f) is then:

1. Pick a random point $x \in \Omega$.
2. Find a point $y \in N(x)$ such that $f(y) < f(x)$. If no such point can be found, then terminate.
3. Assign $x := y$.
4. Go to 2.

Local search can also be restarted to locate multiple local optima. To understand the behavior of this algorithm, we need to understand the relationship between the neighborhood structure and the objective function f.

A neighborhood structure can be represented as a directed graph, with points of Ω as vertices and edges (x, y) if $y \in N(x)$. When the neighborhood structure is *symmetric*, as is the case for a binary hypercube (that is, $x \in N(y) \Leftrightarrow y \in N(x)$), we have an undirected graph. This graph, together with the objective function (which is now thought of as a function of the vertices) is sometimes called a *landscape*.

A particular *encoding* is a map from the set Ω to the vertex set of a graph. We will be especially interested in the different encodings that arise from choosing different maps from Ω into a given graph. In particular, we will look at the different landscapes that arise from using binary strings to represent points of Ω, under the Hamming neighborhood structure. When x is labeled using a binary string, the Hamming neighborhood, $N_h(x)$, is the set of stings that can be generated by changing exactly 1 bit in string x.

The remainder of this paper is organized as follows. Section 2 introduces a formal description of representations as encodings that map the search space to the vertex set of a graph. We define the notion of equivalent encodings based on graph automorphisms and prove properties of equivalent bit encodings. Section 3 explains Gray codings and introduces some examples of different Gray encodings. It also relates equivalent bit encodings to the Walsh coefficients. We also prove that for a 1-dimensional unimodal function, steepest ascent using the ℓ-bit Hamming Distance 1 neighborhood of any reflected Gray encoding will converge in at most $\mathcal{O}(\ell)$ steps and order $\mathcal{O}(\ell^2)$ total evaluations. We next investigate functions that are unimodal in Gray space but cannot be solved efficiently by local search algorithms. Finally, in Section 4 we introduce *shifting* (Rana and Whitley, 1997) as a mechanism for changing between a set of restricted Gray codes in an effort to escape local optima. Theoretical results that substantially improve our understanding of the shifting mechanism are presented. New proofs also shed light on the number of unique Gray code neighborhoods accessible via shifting and on how neighborhood structure changes during shifting. We show that shifting can sometimes improve the performance of both a local search algorithm as well as one of the best genetic algorithms currently available.

2 Representations for Local Search

A representation can be modeled in various ways. In the current paper, a representation is an *encoding* that maps a search space onto a graph, which induces a search neighborhood. More specifically, let $\Omega = \{0, 1, \ldots, n-1\}$ be the search space, and let G be a graph on n vertices. An *encoding* of Ω with respect to G is a bijection $c : \Omega \to G$ from the search space to the vertices of the graph. Elements $x, y \in \Omega$ are *neighbors* under the encoding c if $(c(x), c(y))$ is an edge of G.

2.1 Equivalence of Encodings

Two encodings c_1 and c_2 are *equivalent* if, for all $x, y \in \Omega$, x and y are neighbors under c_1 if and only if they are also neighbors under c_2. If two encodings are equivalent then any local neighborhood search algorithm will perform identically with either encoding.

The following definition will help to characterize the equivalence of encodings.

Definition 1 *Given a graph G, an* automorphism *of G is a bijection $g : G \to G$, that maps vertices to vertices, such that (a, b) is an edge of G if and only if $(g(a), g(b))$ is also an edge (for all vertices a, b).*

The set of all automorphisms of a graph forms a group (under function composition) called the automorphism group of the graph. It is a subgroup of the group of all permutations of the vertices of G.

Theorem 1 *Two encodings c_1 and c_2 with respect to a graph G are equivalent if and only if there is an automorphism $g : G \to G$ such that $c_2 = g \circ c_1$.*

Proof: Given an encoding c_1 and any automorphism g, let c_2 be the encoding given by $c_2(x) = g(c_1(x))$. Then $x, y \in \Omega$ are neighbors under c_1 if and only if $(c_1(x), c_1(y))$ is an edge in G, which happens if and only if $(g(c_1(x)), g(c_1(y)))$ is an edge and, equivalently, x and y are neighbors under c_2.

Conversely, if c_1 and c_2 are equivalent then define $g : G \to G$ by $g = c_2 \circ c_1^{-1}$. Recall that encodings are bijections, so the inverse of an encoding is well defined. Then g is clearly a bijection. For any vertices a, b of G, it follows that $(g(a), g(b))$ is an edge if and only if $(c_2 \circ c_1^{-1}(a), c_2 \circ c_1^{-1}(b))$ is an edge, if and only if $c_1^{-1}(a)$ and $c_1^{-1}(b)$ are neighbors under c_2, if and only if they are neighbors under c_1, if and only if (a, b) is an edge. Thus g is an automorphism of G. □

The notion of equivalent encodings forms an equivalence relation on the set of all possible encodings. Each equivalence class corresponds to an *orbit* of the group of automorphisms acting on the set of encodings.

If we label the vertices of G with the numbers $0, 1, \ldots, n-1$ then any encoding can be identified with a permutation of Ω or, equivalently, with a permutation of the vertices of G. The sets of equivalent encodings then correspond to cosets of the subgroup of automorphisms of G. This means, among other things, that if there are a automorphisms of G then a divides $n!$. Each equivalence class contains a encodings, and there are $n!/a$ equivalence classes. In other words, there are exactly $n!/a$ distinct ways of encoding Ω with the vertices of G.

2.2 Example: Bit Encodings

As an example, suppose that $n = 2^\ell$ for some integer ℓ and let H_ℓ be the ℓ-dimensional hypercube whose vertices are binary strings of length ℓ. Two strings have an edge between them if they differ in only one bit position. Automorphisms of H_ℓ arise in two ways. First, given any bit string m (which we will call a *mask*), consider the map $x \longmapsto x \oplus m$ where $\oplus$ indicates exclusive-or. It is not hard to see that (x, y) is an edge if and only if $(x \oplus m, y \oplus m)$ is an edge. Each bit string mask gives rise to an automorphism in this way.

The second way of getting an automorphism is to permute some of the bit positions in the strings. Suppose we have a permutation π of the set $\{1, 2, \ldots, \ell\}$. Let $a_1 a_2 \ldots a_\ell$ be a binary string and consider the map

$$a_1 a_2 \ldots a_\ell \longmapsto a_{\pi(1)} a_{\pi(2)} \ldots a_{\pi(\ell)}$$

It is easy to check that this map gives an automorphism of H_ℓ.

Harary (2000) has proven that any automorphism of H_ℓ can be described as a unique combination of a mask and a permutation. Since there are 2^ℓ masks and $\ell!$ permutations, then there are $2^\ell \ell!$ unique automorphisms altogether. The total number of permutations of Ω is $n!$, where $n = 2^\ell$. The number of distinct encodings of Ω with respect to H_ℓ is therefore $(n-1)!/\ell!$.

2.3 Objective Functions as Vectors

Given a fitness function $f : \Omega \to \mathbb{R}$ and an encoding $c : \Omega \to G$, we can define a fitness function with respect to the encoding $f_c : G \to \mathbb{R}$ by

$$f_c(a) = f(c^{-1}(a))$$

for all vertices a of G. We pick an arbitrary encoding b to be a *reference*. Then any fitness function defined with respect to some other encoding c can be written as a vector $(f_0, f_1, \ldots, f_{n-1})$ where

$$f_k = f_c(b(k)) = f(c^{-1} \circ b(k))$$

Note that $f_k = f(k)$ and letting f_b denote the binary reference encoding,

$$f_k = f_b(b(k)) = f(b^{-1} \circ b(k)) = f(k)$$

For example, suppose $n = 4$ and we have a fitness function $f(x) = 2x$. If our graph is the hypercube H_2 then we can pick ordinary binary encoding as our reference b. The fitness vector with respect to the binary encoding is then $(0, 2, 4, 6)$. However, if we have the following encoding c:

$$\begin{aligned} 0 &\longmapsto 00 \\ 1 &\longmapsto 01 \\ 2 &\longmapsto 11 \\ 3 &\longmapsto 10 \end{aligned}$$

then the fitness vector becomes $(0, 2, 6, 4)$. It is important to note the role of standard binary as the reference coding:

$$f_2 = f_c(10) = f(c^{-1} \circ 10) = f(3) = 6$$

The encoding c when inverted map backs from the bit string to a point in the domain, which allows function f to be evaluated. In this case, the coding c happens to behave as a Gray code.

We can think of fitness vectors as living in an n-dimensional vector space. Changing encodings changes the fitness vector by permuting the entries around (since different codes now stand for different elements of the search space). Given an encoding c, define a matrix $A(c)$ by

$$A(c)_{i,j} = [b(i) = c(j)]$$

where b is our reference encoding. If f is thought of as a fitness vector with respect to b then

$$(A(c)f)_j = f(c^{-1} \circ b(j))$$

which is f_j under encoding c. In other words, if f is the fitness vector under the reference encoding then $A(c)f$ is the fitness vector under encoding c. Obviously $A(b)$ is just the identity matrix. For the example encoding above, we have

$$A(c) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

so that

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 4 \\ 6 \end{bmatrix} = \begin{bmatrix} 0 \\ 2 \\ 6 \\ 4 \end{bmatrix}$$

If g is an automorphism of G, then we can define a matrix $\sigma(g)$ relative to the reference encoding b by

$$\sigma(g)_{i,j} = [b(i) = g(b(j))]$$

We say that $\sigma(g)$ is the *matrix representation* of the group element g. It is easy to verify that if g and h are automorphisms then $\sigma(g \circ h) = \sigma(g)\sigma(h)$.

In the case of equivalent encodings we then have:

Theorem 2 *Let c be an encoding with respect to a graph G and g an automorphism of G. Then*

$$A(g \circ c) = \sigma(g)A(c)$$

Proof: Let $c_2 = g \circ c_1$, then we have

$$\begin{aligned} (\sigma(g)A(c_1))_{i,j} &= \sum_k [b(i) = g(b(k))][b(k) = c_1(j)] \\ &= \sum_k [g^{-1}(b(i)) = b(k)][b(k) = c_1(j)] \\ &= [g^{-1}(b(i)) = c_1(j)] \\ &= [b(i) = g \circ c_1(j)] \\ &= A(c_2)_{i,j} \end{aligned}$$

□

3 Gray Codes

This section focuses on Gray codes. Four main results are presented. First, we prove that functions with identical Hamming distance 1 neighborhoods have similar Walsh coefficients. Second, we prove that any function that is unimodal under the natural encoding is also unimodal on the hypercube under a Gray encoding. Third, we prove that for a 1-dimensional unimodal function, steepest ascent using the ℓ-bit Hamming Distance 1 neighborhood of any "reflected" Gray encoding will converge in at most $\mathcal{O}(\ell)$ steps and using $\mathcal{O}(\ell^2)$ evaluations. Finally, we show that there exist functions that are unimodal in Gray space but that cannot be solved efficiently by local search algorithms.

3.1 Standard Binary Reflected Gray Code (and other Gray Codes)

An encoding $c : \Omega \rightarrow G$ is called a *Gray code* if the sequence

$$c(0), c(1), \ldots, c(n-1)$$

forms a Hamiltonian cycle in G. The canonical example of a Gray code on the hypercube is the Standard Binary Reflected Gray code. This reflected Gray code can be defined recursively.

- When $\ell = 1$ then the Gray code sequence is $c(0) = 0$, $c(1) = 1$.
- Given a Gray code c on ℓ bits, construct a new sequence on $\ell + 1$ bits according to the scheme:

  $$0c(0), 0c(1), \ldots, 0c(n-1), 1c(n-1), \ldots 1c(1), 1c(0)$$

  where $n = 2^\ell$.

Note that there is a reflection point between $0c(n-1)$ and $1c(n-1)$ such that the new hypercube has a neighborhood structure that folds along this reflection point. Note also that we could define a different reflected Gray code by defining the Gray code sequence to be $c(0) = 1$, $c(1) = 0$ for $\ell = 1$.

In general, an encoding is a Gray code if adjacent integers are represented by bit strings that are Hamming distance 1 from each other; in other words, they differ by a single bit. By way of contrast, standard Binary encodings have "Hamming cliffs" where complementary bit strings representing adjacent integers: for example, 0111 = 15 and 1000 = 16. Obviously, Gray codes also have the property that the maximal distance between 2 strings in Hamming space (i.e., a hypercube) is at most the difference between the two integers that those strings represent. The distance is typically less: given a set of 2^l integers, no two strings are more than l moves away in the corresponding hypercube.

Let $\Omega = \{0, 1, \ldots, 2^\ell - 1\}$. For any $x, y \in \Omega$ define an *interval*

$$[x, y] = \{x, x+1, x+2, \ldots, y\}$$

where addition is always modulo 2^ℓ. The number of elements in $[x, y]$ is $y - x + 1 \bmod 2^\ell$.

The Standard Binary Reflected Gray code is easy to compute: construct the standard Binary bit representation of an integer, then apply logical exclusive-or to the Binary ℓ-bit representation and a duplicate of the Binary bit representation that has been shifted 1 bit to the right. The first ℓ bits of this exclusive-or operation is the Standard Binary Reflected Gray code of the integer (Bitner et al., 1976).

There also exists an $\ell \times \ell$ matrix G_ℓ that transforms a string of length ℓ from Binary to the reflected Gray representation. There also exists a matrix D_ℓ that maps the Gray string back to its Binary representation. Given a bit vector x representing the standard Binary vector for integer I, $x^T G_\ell$ produces a Gray coding of x and integer I; $x^T D_n$ produces a de-Gray coding of x, where x is an ℓ-bit column vector. The G_ℓ matrix has "1" bits along the diagonal and the upper minor diagonal; and D_ℓ has "1" bits in the upper triangle and on the diagonal.

For example, for $\ell = 4$:

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

Again, recall that when a Gray code is used in conjunction with a genetic algorithm or local search algorithm, the string is actually de-Grayed during evaluation.

Transformation matrices for other Gray codes can be produced by permuting the columns of G_ℓ. Other Gray codes can also be produced by first adding a constant K to each integer (using addition mod 2^ℓ), then generating the Binary representation and Graying using G_ℓ or some permutation of G_ℓ. The exact number of Gray codes is unknown.

Of the exponentially many Gray codes, at least $\ell!$ of these induce exactly the same neighborhood structure; permutations of the columns of the Gray transform matrix "reflect" the space along a different dimension of the hypercube without changing the actual connectivity of the search space. This is in effect the same as permuting the bits of each string in the search space. Permuting the bits creates a "new Gray code," but this does not change the actual neighborhood structure. It is also possible to apply different *conditional transforms* of submatrices of the Gray matrix to produce Gray codes with different connectivities. While this yields different Gray codes, this method is complex and cumbersome.

Another simple way to produce different Gray codes is to "shift" the integer domain. This results in reflected Gray codes with different connectivity; therefore, shifting can change the number of local optima. Yet we still retain all the advantages and properties of a reflected Gray code.

When a bit string is decoded, the string is mapped to an integer value and appropriately scaled to the function domain. Shifting is simply an integer offset that is applied to the intermediate integer that is generated during the decoding process. The shifted result is wrapped around the ends of the valid integer domain. To be more precise, "shifting" can be accomplished by 1) de-Graying the bit string, 2) converting the resulting bit string to the Binary representation and converting to integer, and then 3) shifting the integer by a constant (using addition mod 2^ℓ). After this, the integer can be mapped to the actual discretized domain.

Figure 1 provides examples of different ways to produce different Gray codes. The first three rows of bit strings represent different circular shifts of the bits that make up the strings. This corresponds to part of an automorphic group which permutes bit positions. (The set of shifts is also an automorphism which is a subset of the set of permutations). Each such bit shift produces a "different" Gray code, but does not change neighborhood connectivity in the hypercube. The last three rows of integers are different shifts of the integer sequence. Note that if we start with a "reflected" Gray code, all shifts of the integer sequence remain a "reflected" Gray code. (Note that this blurs the distinction between transforming the function versus transforming its encoding; in effect, there is no difference between the two).

Pairing any Gray sequence of bit strings with a different shift of the integers produces a different Gray code, but only some of these combinations change the neighborhood structure under local search. This makes it important to understand when encodings will be equivalence and the formal properties associated with equivalence encodings.

3.2 The Walsh Transform of Equivalent Encodings

Any automorphism of the hypercube H_ℓ can be written as the product of a mask and a permutation. That means that the matrix corresponding to any automorphism of H_ℓ is of the form $\sigma(\pi)\sigma(m)$, where $\sigma(\pi)$ is the matrix representing the permutation π and

000	001	011	010	110	111	101	100	A Circular Shift of the bit strings to create different Gray codes
000	010	110	100	101	111	011	001	
000	100	101	001	011	111	110	010	
0	1	2	3	4	5	6	7	A Circular shift on the sequence of integers to create different Gray codes
1	2	3	4	5	6	7	0	
2	3	4	5	6	7	0	1	

Figure 1: The first three rows of bit strings represent different circular shifts of the bits in the strings themselves. Each such bit shift produces a "different" Gray code, but does not change neighborhood connectivity in the hypercube. The last three rows of integers are different shifts of the integer sequence. Pairing the first row of bits strings with the first row of the integers 0 to 7 yields the Standard Binary Reflected Gray code.

$\sigma(m)$ represents the application of mask m.

Define the Walsh matrix with respect to the reference encoding b to be

$$W_{i,j} = \frac{1}{\sqrt{n}}(-1)^{b(i)^T b(j)}$$

where we view the strings $b(i)$ and $b(j)$ to be ℓ-dimensional vectors. (The constant $1/n$ is sometimes used in place of $1/\sqrt{n}$. The current form simplifies the inverse of W, such that $W = W^{-1} = W^T$). The Walsh Transform of a vector $x \in \mathbb{R}^n$ is given by

$$\widehat{x} = Wx$$

and the Walsh Transform of a matrix M is given by

$$\widehat{M} = WMW^{-1}$$

The Walsh Transform of a vector produces another vector composed of what are known as the Walsh coefficients. This is similar to a Fourier transform in that a point-value representation of the function is transformed into a polynomial representation of the function made up of the Walsh coefficients.

Whitley (2000) used one particular set of transforms to map one function to another function in such as way as to preserve the local search neighborhood. Furthermore, such a transformation appeared to have a conservative effect on the Walsh coefficients of the function. In particular, it was observed that the magnitudes of the Walsh coefficients did not change, but rather the coefficients "move" and "change signs". By "moving" we mean that a coefficient associated with one subset of bits may now be associated with a different subset of bits. The following example helps to make this clear.

One way to transform the function and to preserve the neighborhood is to "shift" the integer sequence corresponding to the decode bit strings by $2^{\ell-2}$. Let f_i index the i^{th} element of f. We can construct a new function, f^4, by the following shift in the indexing: $f^4_i = f_{i-4}$. Addition is modulo 16. Viewing the vector as a permutation, this

Table 1: Walsh coefficients of a function after shifting by 0, 4, 8, and 12. The set of integers obtained by taking the absolute value of the Walsh coefficients is identical for all four functions.

Shift	w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}
f	1500	-204	-324	436	-68	-44	-36	404	-596	-380	-340	100	332	-412	-372	68
f^4	1500	-204	324	-436	-596	-380	340	-100	68	44	-36	404	-332	412	-372	68
f^8	1500	-204	-324	436	68	44	36	-404	596	380	340	-100	332	-412	-372	68
f^{12}	1500	-204	324	-436	596	380	-340	100	-68	-44	36	-404	-332	412	-372	68

shift moves the first 4 elements of the permutation to the end of the permutation. We also define $f_i^8 = f_{i-8}$ and $f_i^{12} = f_{i-12}$.

$$
\begin{aligned}
f &=< 16, 32, 1072, 48, 176, 272, 112, 80, 496, 592, 752, 656, 304, 368, 208, 816 > \\
f^4 &=< 176, 272, 112, 80, 496, 592, 752, 656, 304, 368, 208, 816, 16, 32, 1072, 48 > \\
f^8 &=< 496, 592, 752, 656, 304, 368, 208, 816, 16, 32, 1072, 48, 176, 272, 112, 80 > \\
f^{12} &=< 304, 368, 208, 816, 16, 32, 1072, 48, 176, 272, 112, 80, 496, 592, 752, 656 >
\end{aligned}
$$

Under Gray code, the i^{th} component of the permutation will be presented by the Standard Binary Reflected Gray encoding of the integer i. This means that the Walsh coefficients are computed with respect to the fitness vector after decoding, so that $f_i = f_c(i) = f(c^{-1}(i))$. By inspection, all of these functions have identical neighborhoods under Gray code. Table 1 shows that there are also strong similarities between the Walsh coefficients of these functions under Gray code.

The practical implications of shifting will be examined in more detail later in this paper. For now, we prove that any transformation that preserves neighborhood structure also has a similar, conservative impact on the Walsh coefficients.

Lemma 3 *Let π be a permutation of bit positions of strings of length ℓ. Then for any strings u, v,*

$$u^T \pi(v) = \pi^{-1}(u)^T v$$

Proof: We can think of π as being a permutation of the set $\{1, 2, \ldots, \ell\}$ so that

$$(\pi(x))_k = x_{\pi(k)}$$

Then for any strings u, v we have

$$
\begin{aligned}
u^T \pi(v) &= \sum_k u_k \pi(v)_k \\
&= \sum_k u_k v_{\pi(k)} \\
&= \sum_j u_{\pi^{-1}(j)} v_j \\
&= \sum_j \pi^{-1}(u)_j v_j \\
&= \pi^{-1}(u)^T v
\end{aligned}
$$

□

Theorem 4 *If $\sigma(\pi)$ is the matrix representing permutation π then*

$$W\sigma(\pi)W^{-1} = \sigma(\pi)$$

Proof:

$$\begin{aligned}
(\sigma(\pi)W)_{i,j} &= \sum_k \sigma(\pi)_{i,k} W_{k,j} \\
&= \frac{1}{\sqrt{n}} \sum_k [b(i) = \pi(b(k))](-1)^{b(k)^T b(j)} \\
&= \frac{1}{\sqrt{n}} (-1)^{\pi^{-1}(b(i))^T b(j)} \\
&= \frac{1}{\sqrt{n}} (-1)^{b(i)^T \pi(b(j))} \\
&= \frac{1}{\sqrt{n}} (-1)^{b(i)^T b(k)} [b(k) = \pi(b(j))] \\
&= (W\sigma(\pi))_{i,j}
\end{aligned}$$

□

Theorem 5 *If $\sigma(m)$ is the matrix representing the action of mask m then $W\sigma(m)W^{-1}$ is a diagonal matrix with i^{th} diagonal entry given by*

$$\left(W\sigma(m)W^{-1}\right)_{i,i} = (-1)^{b(i)^T m}$$

Proof:

$$\begin{aligned}
\widehat{\sigma(m)}_{i,j} &= \sum_u W_{i,u} \sum_v \sigma(m)_{u,v} W_{v,j} \\
&= \frac{1}{n} \sum_{u,v} (-1)^{b(i)^T b(u) + b(v)^T b(j)} [b(u) = m \oplus b(v)] \\
&= \frac{1}{n} \sum_v (-1)^{b(i)^T (m \oplus b(v)) + b(v)^T b(j)} \\
&= \frac{1}{n} \sum_v (-1)^{b(i)^T m} (-1)^{b(v)^T (b(i) \oplus b(j))} \\
&= (-1)^{b(i)^T m} [b(i) = b(j)] \\
&= (-1)^{b(i)^T m} [i = j]
\end{aligned}$$

□

We will write this diagonal matrix as $\mathbf{diag}_k((-1)^{b(k)^T m})$.

Now let f be a fitness vector with respect to some encoding b, and let c be any equivalent encoding. Then there exists a permutation π and a mask m such that $\sigma(\pi)\sigma(m)f$ is the fitness vector under encoding c. Now $\widehat{f} = Wf$ contains the Walsh coefficients of the fitness function f under our first encoding, b. The Walsh coefficients under the new encoding c are given by

$$\begin{aligned}
W\sigma(\pi)\sigma(m)f &= W\sigma(\pi)W^{-1}W\sigma(m)W^{-1}Wf \\
&= \sigma(\pi)\mathbf{diag}_k((-1)^{b(k)^T m})\widehat{f}
\end{aligned}$$

So the ith Walsh coefficient of f under encoding c is

$$\begin{aligned}(W\sigma(\pi)\sigma(m)f)_i &= \sum_j \sigma(\pi)_{i,j}(-1)^{b(j)^T m}\widehat{f}_j \\ &= \sum_j [b(i)=\pi(b(j))](-1)^{b(j)^T m}\widehat{f}_j \\ &= (-1)^{\pi^{-1}b(i)^T m}\widehat{f}_{b^{-1}\pi^{-1}b(i)}\end{aligned}$$

This means that the Walsh coefficient associated with string s in the new encoding has the same magnitude as the string $\pi^{-1}(s)$ in the old encoding, with a sign change if $\pi^{-1}(s)$ and m have an odd number of ones in common.

As an example, let b be the standard binary encoding and $f = (1,3,6,8)$. The Walsh Transform of this vector is

$$\widehat{f} = (9,-2,-5,0)$$

Now consider the automorphism based on the mask 11. That is

$$\begin{aligned} s &\longmapsto s \oplus 11 \\ 00 &\longmapsto 11 \\ 01 &\longmapsto 10 \\ 10 &\longmapsto 01 \\ 11 &\longmapsto 00 \end{aligned}$$

We know that this is an equivalent encoding to the original. The fitness vector with respect to the new encoding is $(8,6,3,1)$. The Walsh Transform of this vector is $(9,2,5,0)$. In this case the automorphism is just a mask, with no permutation of bit positions, and the theory therefore indicates that the coefficients will retain their positions and magnitudes, with a change of sign when string m and $s \oplus 11$ have an odd number of ones in common.

3.3 Local Optima under Gray Encodings

It is well known that local search will terminate at a *local optimum*, that is at some point $x \in \Omega$ such that none of the points in the neighborhood $N(x)$ improve upon x according to the objective function. We say that a problem is *unimodal* if it has exactly one local optimum, *bimodal* if it has exactly two, and *multimodal* if it has more than one local optimum. Of course, the neighborhood structure of a problem depends upon the coding scheme used, and in this section we will look at the effect of using Gray-encoding on local optima.

Suppose the objective function is defined on the unit interval $0 \le x < 1$. One way to optimize such a function is to discretize the interval and then use local search. That is, we select n points $0, 1/n, 2/n, \ldots (n-1)/n$ in the interval, and identify them with the points $\{0,1,2,\ldots,n-1\}$ of Ω. The *natural encoding* is then a map from Ω to the graph that has edges between points x and $x+1$ for all $x = 0,1,\ldots,n-2$. If the function is periodic, we may also include an edge from $n-1$ to 0. For simplicity, we will assume that our functions have this periodic property.

If we choose $n = 2^\ell$ for some ℓ then we can encode the points of Ω as binary strings of length ℓ. The choice of a Gray-encoding has the following nice property.

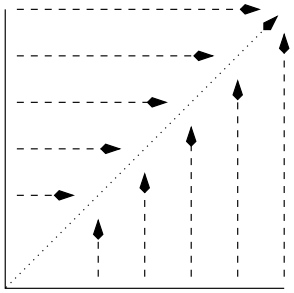

Figure 2: Local search moves only in the horizontal and vertical directions. It therefore "finds" the diagonal, but becomes stuck there. Local search is blind to the fact that there is gradient information moving along the diagonal.

Theorem 6 *A function $f : \Omega \rightarrow \mathbb{R}$ cannot have more local optima under Gray encoding than it does under the natural encoding.*

Proof: Let $x \in \Omega$ be a point that is not a local optimum under the natural encoding. Therefore either $x - 1 \pmod{n}$ or $x + 1 \pmod{n}$ is better than x. But both these points are neighbors of x under any Gray code. Therefore x cannot be a local optimum under any Gray encoding. □

The original proof appeared in Whitley and Rana (1997). This theorem states that when using a Gray code, local optima of the objective function considered as a function on the unit interval can be destroyed, but no new local optima can be created. In particular we have

Corollary 7 *If a function is unimodal under the natural encoding, then it is also unimodal under any Gray encoding.*

However, this corollary also begs the question: Are there functions which are unimodal in real space that are multimodal in Gray space, because the "natural encoding" is also multimodal? If the function is 1-dimensional, the answer is no. But if the function is not 1-dimensional, the answer can be yes. Consider the following 2-dimensional function.

$$f(x, y) = (x - y)^2 + 1/(x + y)$$

A simple representation of this function appears in Figure 2.

Changing one variable at a time will move local search to the diagonal. However, looking in either the x-dimension or the y-dimension one parameter at a time, every point along the diagonal appears to be a local optimum to local search. There is actually gradient information if one looks *along* the diagonal; however, this requires either 1) changing both variables at once, or 2) transforming the ordinate system used for sampling the search space so as to "expose" the gradient information.

Winston's classic AI text (Winston, 1984) describes this as the *ridge problem* and points out that this is one of the basic limitations of simple hill-climbing. Surprisingly, this problem has received very little attention from the local search community.

3.4 Convergence of Local Search for Unimodal 1-D Functions

We have shown that if the natural encoding of a function is unimodal, then the Gray encoding is also unimodal. It can also be proven that the convergence time for steep-

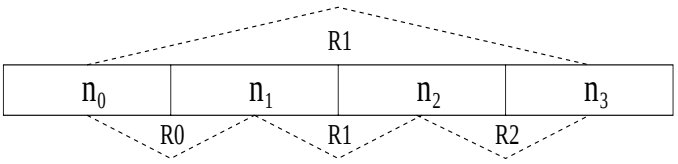

Figure 3: The dashed lines connect Gray encoded neighbors. Neighbors in adjacent quadrants are 1 move away. No quadrant is more than 2 moves away.

est ascent local search is linear for 1-dimensional bijective unimodal functions. Since the functions are 1-dimensional, the number of optima in real space must be the same at the number of optima under Gray code as well as the natural encoding. Because the function is a bijection, each evaluation is unique and there are no plateaus or "flat spots" where local search might become stuck.

The proof looks at adjacent quadrants of the search space. Some of the key ideas in the proof are illustrated in Figure 3. Gray code generates a circuit between 4 critical neighbors. These neighbors are found by reflecting, or folding, the search space around the reflection points (R0, R1, R2 in Figure 3) used in constructing the reflected Gray code. Under a Standard Binary Reflected Gray code, the point n_0 has a neighbor n_3 which differs in the first bit. Point n_0 has a second neighbor in position n_1. However n_2 (which is the *binary* neighbor of n_0) is a neighbor of n_1 under Gray code and is reached by flipping the first bit; n_2 is also a neighbor of n_3 reached by flipping the second bit. Thus, a circuit exists between n_0, n_1, n_2, n_3 which is traversed by changing one of the first two bits.

A sketch of a proof for convergence in linear time on unimodal 1-dimensional functions was first presented by Whitley et al. (2001). The proof presented here fills in some critical details and also requires less explicit sampling.

We have a search space $\Omega = \{0, 1, ..., 2^\ell - 1\}$ and a function $f : \Omega \rightarrow \mathbb{R}$. To begin, we consider arbitrary intervals of the search space. For a 1-dimensional function with range 0 to M, an *interval* of the search space is a continuous region of the search domain from point x to y: $[x, y] = \{x, x + 1, x + 2, ..., y\}$. Let Q_i denote an interval. We will define a *partition* of the search space as a set of d intervals such that

$$(\forall i, j) \;\; \{Q_i \cap Q_j = \emptyset \text{ and } \cup_{i \in d} Q_i = \Omega\}.$$

Let Q_i and Q_{i+1} denote adjacent intervals. The indices for adjacent intervals start at 0 and addition is modulo $d - 1$. Also, $f(n)$ wraps so that n_0 is a neighbor of $n_{2^\ell - 1}$. We will mainly be concerned with the partition of the search space into quadrants:

$$\begin{aligned} Q_0 &= \left\{0, ..., 2^{\ell-2} - 1\right\} \\ Q_1 &= \left\{2^{\ell-2}, ..., 2 \cdot 2^{\ell-2} - 1\right\} \\ Q_2 &= \left\{2 \cdot 2^{\ell-2}, ..., 3 \cdot 2^{\ell-2} - 1\right\} \\ Q_3 &= \left\{3 \cdot 2^{\ell-2}, ..., 4 \cdot 2^{\ell-2} - 1\right\} \end{aligned}$$

These quadrants of the search space corresponding to the hyperplanes 00*...*, 01*...*, 10*...* and 11*...* are shown in Figure 3. However, some preliminary concepts apply to arbitrary intervals as well. Let the function $MAX(Q_i)$ return the maximum value of $f(n)$ over all $n \in Q_i$.

Definition 1: The critical interval. Given an interval of a 1-dimensional bijective function into intervals indexed by j, one can identify a unique *critical interval*, Q_i, where

$$(\forall j, j \neq i)\ \ MAX(Q_i) < MAX(Q_j).$$

It immediately follows from this definition that a critical interval also has the following property. *If Q_i is* **critical** *there exists $n_q \in Q_{i-1}$ and $n_p \in Q_{i+1}$ such that for all $(n_i \in Q_i),\ f(n_p) > f(n_i) < f(n_q)$.*

Lemma 8 *Let $f : \Omega \rightarrow \mathbb{R}$ be unimodal and 1 dimensional. Let Q_i be a critical interval of a partition. Then either Q_i contains the optimum, or one of its neighboring intervals, Q_{i-1} or Q_{i+1} contains the optimum. Furthermore, there exists a neighboring interval Q_j such that*

$$(\forall x \in Q_i), (\forall y \notin \{Q_i \cup Q_j\}), f(y) > f(x)$$

Proof: If $(\forall x \in Q_i)$ and $(\forall y \notin Q_i),\ \ f(x) < f(y)$, then the results hold.

Otherwise, there exists $y \in Q_j$ such that $f(y) < MAX(Q_i)$, where Q_j is an adjacent interval. Recall that n_0 is a neighbor of $n_{2^\ell-1}$. Therefore we can then think of the search space as being equivalent to the concatenation of two subfunctions. One monotonically increases to the left from the global minimum to the global maximum, the other monotonically increases to the right from the global minimum to the global maximum.

Consider the point $y \in Q_j$, where $f(y) < MAX(Q_i)$. Because the function is a bijection, we can also associate points with their evaluation. We know $MAX(Q_j) > f(y) < MAX(Q_i)$; it follows that the global optimum must lie between the points associated with $MAX(Q_j)$ and $MAX(Q_i)$ since the function is unimodal. The global maximum of the function cannot lie in Q_i; therefore $MAX(Q_i)$ must occur at the beginning or end of the interval Q_i, since the function is unimodal.

If the global maximum lies outside Q_j, then starting at the points associated with $MAX(Q_j)$ and $MAX(Q_i)$ the function must climb monotonically to the global maximum. Since $MAX(Q_j) > MAX(Q_i)$ all points outside Q_j and Q_i are worse than $MAX(Q_i)$.

If the global maximum lies inside Q_j, then $MAX(Q_j)$ *is* the global maximum. In this case, we can start at the point associated with $MAX(Q_i)$ and the functions must climb monotonically to the global maximum. Thus, all points outside Q_j and Q_i are worse than $MAX(Q_i)$. □

Corollary 9 *If Q_i is the critical interval of a partition, there exists a neighboring interval Q_j such that*

$$(\forall y \notin \{Q_i \cup Q_j\}), f(y) > MAX(Q_i)$$

Next, let the intervals explicitly be quadrants.

Lemma 10 *Let Q_i be an interval corresponding to some quadrant of f. There exists points n_0, n_1, n_2, n_3, where $n_i \in Q_i$ such that there is a cyclic path between n_0, n_1, n_2, n_3. Each point is reachable in at most 2 moves under a Gray encoding.*

Proof: Let R_i be the Gray code reflection point between neighbors n_i and n_{i+1}. Thus n_0 and n_1 are neighbors equidistant from R_0, n_1 and n_2 are neighbors equidistant from R_1 and n_2 and n_3 are neighbors equidistant from R_2. By construction the entire search

space folds around $R1$, therefore reflection $R0$ folds onto $R2$ and n_0 folds onto n_3 and thus is a neighbor of n_3. This forms a circuit:

$$n_0 \leftrightarrow n_1 \leftrightarrow n_2 \leftrightarrow n_3 \leftrightarrow n_0$$

where $\leftrightarrow$ denotes one move in Hamming space. □

Theorem 11 *For any 1-dimensional unimodal bijective function, steepest ascent local search over the reflected Gray code neighborhood of the function converges to the global optimum after $\mathcal{O}(\ell)$ moves and $\mathcal{O}(\ell^2)$ evaluations. The total number of moves is at most 2ℓ. The total number of evaluation is at most $2\ell^2$.*

Proof: Since the function is a bijection, there is a unique optimum and there is a single best point in every neighborhood. At each step, we assume the current point is not the optimum. If the current point is the optimum, it is identified as such since its immediate neighbors to the left and right are Gray code neighbors and evaluating these proves optimality.

The search space can be broken into quadrants. We start at a point n_i, which we know from Lemma 10 is part of a circuit of neighboring points. Assume the search starts in quadrant Q_i and the neighbors of n_i have already been evaluated.

Case 1. Assume the best neighbor is found inside of Q_i. Let m_i represent that point. Thus, $f(n_{i-1}) > f(m_i) < f(n_{i+1})$. Under Gray code, the only neighbors of n_i outside of Q_i are n_{i-1} and n_{i+1}. It must also be the case that $f(m_i) < f(n_{i+2})$, otherwise the function is not unimodal. Therefore we know $f(m_i) < MIN\{MAX(Q_i), MAX(Q_{i-1}), MAX(Q_{i+1})MAX(Q_{i+2})\}$ This means that $f(m_i) < MAX(Q_c)$ of the critical quadrant, Q_c. Note that this determination is made without actually moving.

Thus, after finding the best move possible, Corollary 9 implies at least two quadrants are strictly worse than the best move and the search space can be cut in half.

Case 2. The first move is to point $n_j = n_{i-1}$ or n_{i+1}; denote this quadrant by Q_j. After this move, n_{i+2} is evaluated. Assume the best move is inside of Q_j. After 1 move, this case is now exactly like case 1.

Case 3. After the first move, a second move is required to point n_{i+2}. We know $f(n_{i-1}) > f(n_{i+2}) < f(n_{i+1})$. The next move must be inside of Q_{i+2}. Thus, after 2 moves, this case is now exactly like case 1.

There are no other cases.

We next must show that local search using a Gray encoding recursively decomposes the search space, eliminating half the space after at most 2 moves. By the reflected construction, there is a single bit which when set eliminates two quadrants from further consideration. The bit need not be eliminated explicitly, because after at most 2 moves, flipping the "eliminated" bit always produces an inferior neighbor. Lemma 8 proves the eliminated quadrants are always adjacent. If quadrants 1 and 2 or quadrants 3 and 4 are eliminated then the first bit is set (to 1 or 0). If quadrants 2 and 3 or quadrants 1 and 4 are eliminated then the second bit is set (to 1 or 0). There are no other cases.

Lemma 8 and Lemma 10 prove that after at most 2 moves, at least one key bit is implicitly fixed and the search space is cut in half. This implicitly describes a recursive process that defines a new unimodal bijective 1-dimensional unimodal function.

After the search space has been reduced to 4 points, the global optimum is automatically sampled and verified. This provides the necessary basis step and reduction of the search space from 2^ℓ to $2^{\ell-1}$ necessary for an inductive proof. This means that the maximum possible number of neighborhood moves is at most 2ℓ, with each neighborhood evaluation having cost ℓ. □

The limitation to 1-dimensional unimodal functions might seem restrictive. But we can now generalize this result in various ways.

A multiparameter separable function can be viewed as a composition of real valued 1-dimensional subfunctions. If each real-valued subfunction is unimodal and encoded with at most l bits, a specialized form of steepest ascent Hamming Distance 1 local search using any reflected Gray-code representation will converge to the global optimum after at most $\mathcal{O}(l)$ steps in each dimension. If there are Q dimensions and exactly l bits per dimension, note that $Ql = \ell$, and the overall convergence is still bounded by $\mathcal{O}(\ell)$.

Theorem 12 *Given a sphere function, $c_0 + c_1 \sum_i^n (x_i - x_i^*)^2$, where $(x_1^*, ...x_n^*)$ denotes the minimum, steepest ascent local search using the ℓ-bit Hamming Distance 1 neighborhood of any reflected Gray-code representation will converge to the global optimum after at most $\mathcal{O}(\ell)$ steps.*

This directly follows from the proof of the 1-dimensional case, since each dimension is independent and unimodal.

3.5 The Empirical Data

The proof obviously shows that steepest ascent, where all ℓ neighbors are checked, will converge in at most $\mathcal{O}(\ell)$ steps and order $\mathcal{O}(\ell^2)$ total evaluations. A set of experiments with different unimodal sphere functions encoded using a varying number of bits shows behavior consistent with the proof. The number of *steps* of steepest ascent local search needed to reach the optimum is linear with respect to string length. This is shown in Figure 4. This figure also shows that the number of evaluations used by steepest ascent is order $\mathcal{O}(\ell^2)$ with respect to string length.

But what about next ascent? In the best case, next ascent will pick exactly the same neighbors as those selected under steepest ascent, and the number of steps needed to reach the optimum is $\mathcal{O}(\ell)$. However, in the worst case an adjacent point in real space is chosen at each move and thus the number of steps needed to reach the optimum is $\mathcal{O}(2^\ell)$. The linear time best case and exponential time worst case makes it a little less obvious what happens on average. The advantage of next ascent of course, is that there can be multiple improving moves found by the time that ℓ neighborhoods have been evaluated.

We used Davis's Random Bit Climber (RBC) as our next ascent algorithm. RBC randomized the order in which the bits are checked. After every bit has been tested once, the order in which the bits are tested is again randomized. This randomization may be important in avoiding worst case behavior.

The empirical data seems to suggest that the convergence time of next ascent is nearly linear. The rightmost graph in Figure 4 shows the empirical behavior of a next ascent bit climbing algorithm, which appears to be bound by $\ell \log(\ell)$ on average. This is in fact better than the ℓ^2 behavior that characterizes the number of *evaluations* required by steepest ascent. The variance of both the steepest ascent and RBC algorithms is too small to be seen on the graphs shown here.

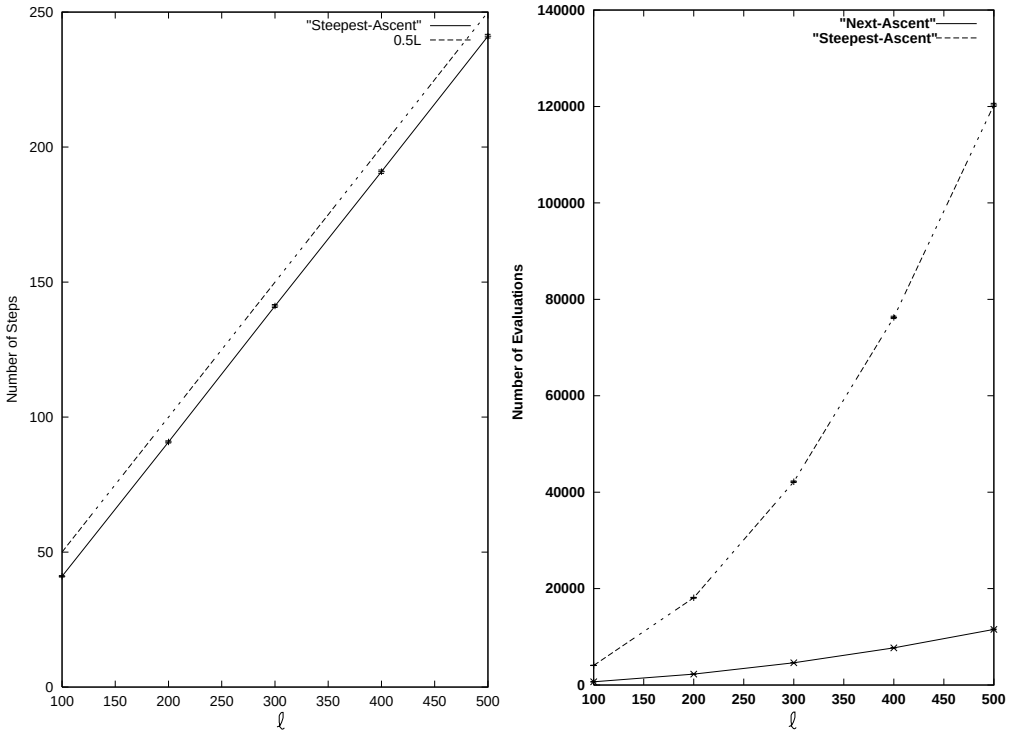

Figure 4: The leftmost graph shows the number of steps needed to reach the global optimum for various sphere functions. The rightmost graph shows the number of *evaluations* needed to reach the global optimum for various sphere functions for both regular steepest ascent and for the next ascent algorithm RBC. RBC requires far fewer evaluations.

3.6 Finding Other Gray-Unimodal Functions

A function which is unimodal on Ω (under the natural encoding) will also be unimodal on the hypercube under a Gray code. However, there are many more functions which are unimodal on the hypercube.

The number of unimodal bijective functions which can be constructed as a 1 dimensional function is given by:

$$2^{\ell}2^{2^{\ell}-2}$$

If we sort the points in the search, there are 2^{ℓ} ways to place the first point. Each point after that can be placed in two ways, either to the left or the right. We assume the search space wraps around. The position of the last point is then determined.

We can also construct examples of unimodal functions under a reflected Gray code. Define a *division* as an interval of the form $[a2^k, (a+1)2^k - 1]$ for some $0 \leq k \leq \ell$ and $0 \leq a < 2^{\ell-k}$. Each point in $z \in \Omega$ is contained in $\ell + 1$ divisions and is uniquely specified by them. The reflected Gray code has the property that for each division $[x, y]$, with $x \neq y$, containing z, the point $x + y - z$ is also in that division and is a neighbor of z. That is, if we split a division in half, then all points in one half have neighbors in the other half. These two halves are also divisions and each point in one half has a

reflected neighbor in the other half.

Select some threshold value $\Theta_0 > 0$. Start with the division $[0, 2^\ell - 1]$, and split it. Choose a new threshold value Θ_1, with $\Theta_0 > \Theta_1 > 0$. Assign fitness values randomly between Θ_0 and Θ_1 to one half. Then repeat the process with the other half, choosing a new lower threshold Θ_2 with $\Theta_1 > \Theta_2 > 0$. Repeat this process, recursively dividing up the set Ω until you reach a single point which you assign a value lower than any others. Functions so defined are unimodal in Gray space *and* a steepest descent algorithm will find the global minimum in a linear number of steps.

Obviously, from this construction there are at least

$$\Pi_{i=1}^{l}(2^{l-i})!$$

different functions that are unimodal *and* solved in a linear number of steps under the Standard Binary Reflected Gray code neighborhood.

For example, let $\ell = 4$, and define the thresholds $\Theta_k = 2^{4-k}$. Randomly generating a function according to the above scheme gives us the following Gray-unimodal function:

$x \in \Omega$	$c(x)$	$f(x)$	$x \in \Omega$	$c(x)$	$f(x)$
0	0000	13	8	1100	5
1	0001	14	9	1101	7
2	0011	9	10	1111	6
3	0010	11	11	1110	8
4	0110	12	12	1010	3
5	0111	10	13	1011	4
6	0101	15	14	1001	2
7	0100	16	15	1000	1

We can also create many other functions which can be optimized efficiently. Take one function f that we already know can be solved efficiently by a local search algorithm, under Gray code. If $c : \Omega \to H_\ell$ is the Gray code, and g is an automorphism of H_ℓ then define a fitness function

$$f'(x) = f(c^{-1} \circ g \circ c(x))$$

Since equivalent encodings have the same neighborhood structure on H_ℓ, this means that the new function f' can be solved just as efficiently as f, by local search methods.

For example, if we use the previously defined function with the automorphism defined by swapping the first and last bit, we get a new Gray-unimodal function that can be optimized in linear number of steps.

$x \in \Omega$	$c(x)$	$f(x)$	$x \in \Omega$	$c(x)$	$f'(x)$
0	0000	13	8	1100	15
1	0001	1	9	1101	7
2	0011	3	10	1111	6
3	0010	11	11	1110	10
4	0110	12	12	1010	9
5	0111	8	13	1011	4
6	0101	5	14	1001	2
7	0100	16	15	1000	14

These two functions are illustrated in Figure 5.

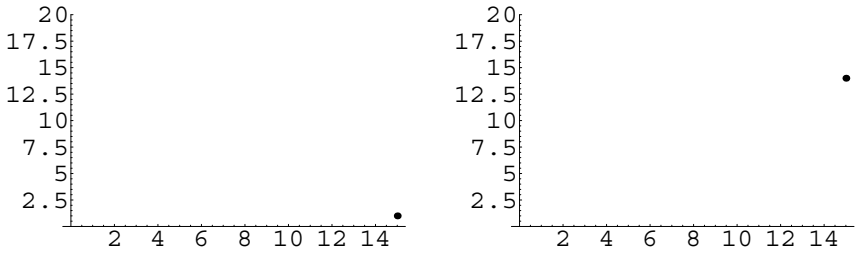

Figure 5: Two fitness functions which are unimodal under reflected Gray encoding. Moreover, the functions have identical landscapes (their neighborhood structures are equivalent) and local search will optimize them in a linear number of steps.

3.7 Long Path Problems

There are also functions which are unimodal in reflected Gray space, but which local search algorithms cannot solve efficiently. An example is the class of *long-path* problems: these problems are unimodal, but the number of local steps required to find the optimum is exponential with respect to ℓ. Such functions cannot correspond to a class of functions of the set Ω under Gray code which are also unimodal under their real-valued encoding. So how badly multimodal are they? We will consider the construction by Horn et al. (1994). Their path is defined recursively. For $\ell = 1$, the path is the sequence $(0, 1)$, which is the whole of the search space. Then, given any path P of length r, we construct a new one according to the formula

$$(00P_1, 00P_2, \ldots, 00P_r, 01P_r, 11P_r, 11P_{r-1}, \ldots, 11P_1)$$

For example, given a path in a 3-dimensional hypercube, we can construct the following path in a 5-D hypercube as shown in Figure 6.

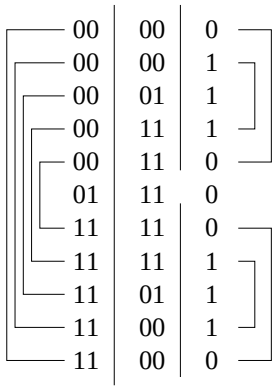

Figure 6: Recursive construction of long-path problems for $\ell = 5$ showing the decomposition based on paths for $\ell = 3$ and $\ell = 1$.

We assign fitness values to increase (we are maximizing) along the path. Any point not on the path gets a fitness value less than any on the path. It is easy to see that the path is designed so that it is impossible to take a short cut from one point to another.

At the t^{th} stage of construction, the path length is given by

$$p(t) = 2p(t-1) + 1$$

(notice the addition of the "bridging" point). We know that $p(0) = 2$. Solving this recurrence gives:

$$p(t) = 3(2^t) - 1$$

The string length increases by 2 at each step, so that $\ell(t) = 2t + 1$. Therefore

$$p(\ell) = 3(2^{\frac{\ell-1}{2}}) - 1$$

(where ℓ is an odd number) and we can see the path length is exponential in ℓ.

The formula for constructing the path shows a close connection with the structure of the reflected Gray code. All the points to the left of the central bridge point are in the first quadrant of Ω and the points to the right are in the third quadrant. The bridge point is somewhere in the second quadrant. However, at the next level of recursion, this same structure is imposed: the first quadrant will be split into four, with points existing in the first and third quarters of the quadrant. The third quadrant will similarly be split. This recursive procedure gives rise to a fractal pattern. This splitting means that at each step there are twice as many local optima as at the previous step, plus an extra one for the bridge point. So if $q(t)$ is the number of local optima at construction step t we have

$$q(t) = 2q(t-1) + 1$$

the same recurrence we had for the path length! Actually, this only holds for $t > 1$ since at the first construction step the bridge point happens to be contiguous with the left side of the path. $q(1)$ therefore has a value of 2. But this is the same as $p(0)$ and so

$$q(t) = p(t-1)$$

for all $t > 0$. We can see then, that the number of local optima is exponential in ℓ. In fact almost exactly half of the points on the path correspond to local optima on Ω. This can be seen in Figure 7 which shows the long-path construction for $\ell = 1, 3, 5, 7$. This would seem to suggest that "long path problems" are a somewhat unusual construction and that they correspond to functions which are highly multimodal in real space.

4 Shifting Gray Code Representations

In this section we introduce *shifting* (Rana and Whitley, 1997) as a mechanism for changing between a set of restricted Gray codes in an effort to escape local optima. We use shifting in conjunction with both a simple hill-climbing algorithm and a state-of-the-art genetic algorithm. These algorithms are evaluated using test functions that have been empirically demonstrated to be both resistant to simple hill-climbing algorithms and which pose a challenge to genetic algorithms (Whitley et al., 1996). The results show that shifting improves both algorithms.

4.1 Shifted Gray Codes

Let $c_0 : \Omega \rightarrow H_\ell$ be the reflected Gray code. For each $k \in \Omega$ define a *shifted* code

$$c_k(x) = c_0(x - k \bmod 2^\ell)$$

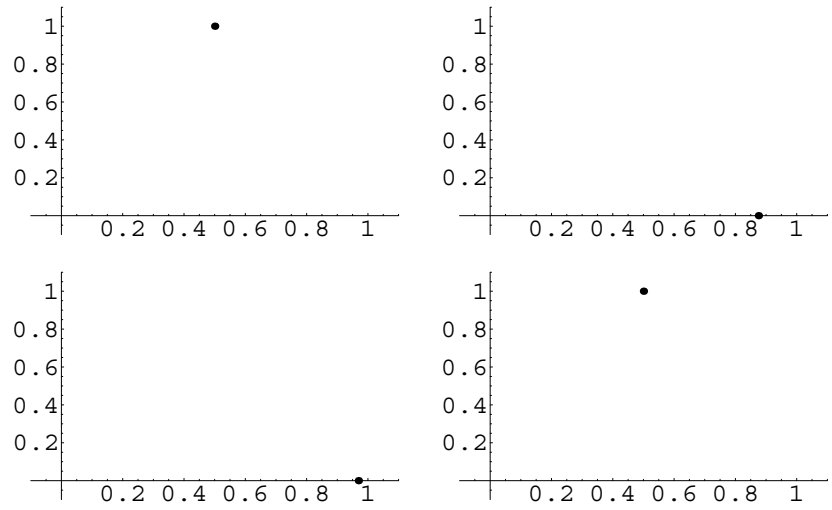

Figure 7: Construction of long-path problems for $\ell = 1, 3, 5, 7$. We interpret the problems as being reflected Gray encodings of functions on the unit interval. The number of local optima (we are maximizing) increases exponentially with the string length ℓ.

It is obvious that each shifted code is also a Gray code. *Shifting* provides a mechanism for switching from one Gray code to another in such a way that it is possible to change the number and/or location of local optima. For a 1 dimensional function, or for any single dimension of a multidimensional function, Gray codes form a circuit. This circuit represents all possible (discretized) inputs to a real-valued function. As the circuit is shifted, points pass by other points in different reflections. This can be seen in Figure 8 for $\ell = 4$. The 4 neighbors are North-South-East-West and the graph is a torus. At the order-3 reflection, strings differ only in the 3rd bit; this connects the North-South neighbors in rows 1 and 2, as well as rows 3 and 4. The directional arrows show that these points move in opposite directions when shifting occurs, and hence neighbors flow past each other. The order-4 reflection (where the 4th bit differs) are the North-South connections between rows 2 and 3, as well as the toroidal North-South wrap around between rows 1 and 4. When two local optima "pass" one another in the shifted Gray encoding, one of the two optima must collapse. For example, in Figure 8 positions 4 and 9 are not neighbors in the integer mapping induced by the Standard Binary Reflected Gray code c_0. However, when the integer space is shifted by k=15 positions 4 and 9 become neighbors under the mapping c_1. If there are local optima at positions 4 and 9, one of these optima must collapse when the search space is shifted.

If we shift by $2^{\ell-2}$ then we get an equivalent code to c_0. In fact, this shift corresponds to the automorphism of using the mask $0110\ldots0$ followed by swapping the first two bits. This means that the set of shift codes divides up into equivalence classes of size 4, and we have $2^{\ell-2}$ codes giving distinct neighborhood structures.

This can be seen by studying Figure 8; a shift of $2^{4-2} = 4$ (or any multiple of 4) leaves the neighborhood structure unchanged. For numbers that are powers of 2, smaller shifts change more neighbors. Note that a shift of 1 implies $1 = 2^0 = 2^{\ell-\ell}$, which changes $\ell - 2$ neighbors. This is the largest number of neighbors that can change

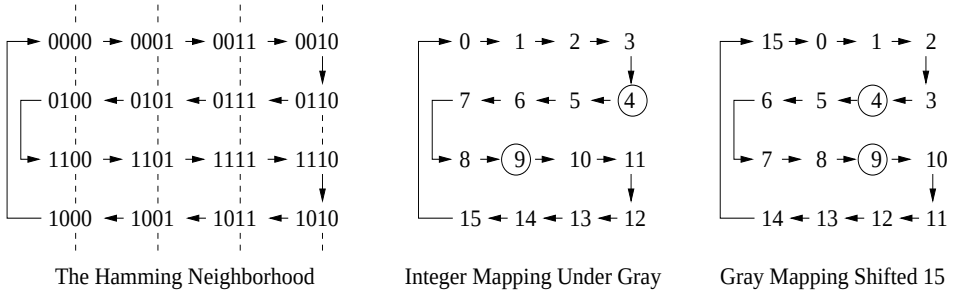

Figure 8: A simple example of shifting. Note that integers are associated with the corresponding positions in the Hamming Neighborhood. Neighbors are N-S-E-W and the arrows map the surface of the real-valued function. Points 4 and 9 are not neighbors under standard reflected Gray code, but become neighbors when the Gray code is shifted by 15.

in a Gray code. Numbers that are not powers of 2 can be viewed as a combination of large and small shifts.

Theorem 13 *For any Gray encoding, shifting by $2^{\ell-k}$ where $k \geq 2$ will result in a change of exactly $k-2$ neighbors for any point in the search space.*

Proof: Consider an arbitrary Gray coding, and $k \geq 2$. Next, divide the 2^ℓ positions into 2^k continuous blocks of equal size, starting from position 0. Each block contains exactly $2^{\ell-k}$ positions (see Figure 9a). Consider an arbitrary block X and arbitrary position P within X. Exactly $\ell - k$ neighbors of P are contained in X. The periodicity of both Binary and Gray bit encodings ensures that the $\ell-k$ neighbors of P in X do not change when shifting by $2^{\ell-k}$. Two of the remaining k neighbors are contained in the blocks preceding and following X, respectively. Since the adjacency between blocks does not change under shifting, the two neighbors in the adjacent blocks must stay the same.

The remaining $k-2$ neighbors are contained in blocks that are not adjacent to X. We prove that the rest of these $k-2$ neighbors change. Consider a block Y that contains a neighbor of P. A fundamental property of a reflected Gray code is that there is a reflection point exactly halfway between any pair of neighbors. For all neighbors outside of block X and which are not contained in the adjacent blocks, the reflection points must be separated by more than $2^{\ell-k}$ positions. Shifting X by $2^{\ell-k}$ will move it closer to the reflection point, while Y is moved exactly $2^{\ell-k}$ positions farther away from the reflection point (see Figure 9b). Point P in X must now have a new neighbor (also $2^{\ell-k}$ closer to the reflection point) in the block Z. If the reflection point between X and Z is at location R, then for the previous neighbor in Y to still be a neighbor of P in X it must have a reflection point at exactly $R + 2^{\ell-k}$. This is impossible since for all neighbors outside of block X which are not contained in the adjacent blocks, the reflection points must be separated by more than $2^{\ell-k}$ positions. A similar argument goes for the case when shifting by $2^{\ell-k}$ moves X farther away from the reflection point (while Y is moved closer). Thus, none of the previous $k-2$ neighbors are neighbors after shifting by $2^{\ell-k}$. □

Corollary of Theorem 13: *In any reflected Gray code representation of a 1-dimensional function, or parameter of a multidimensional function, there are only $2^{\ell-2}$ unique shifts. When $k = 2$, exactly $k - 2 = 0$ neighbors change.*

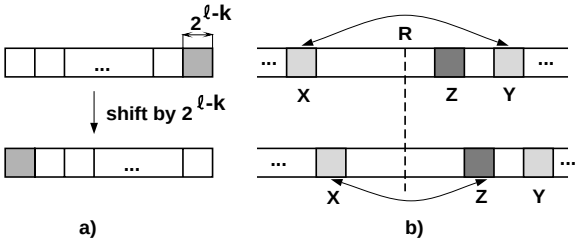

Figure 9: Shifting by $2^{\ell-k}$. **a)** An unwound representation of the Gray codes circuit. **b)** For an arbitrary position in a block X, and an arbitrary neighbor of this position in the block Y, after shifting by $2^{\ell-k}$, the neighbor moves from block Y to block Z.

4.2 Bimodal Objective Functions

Suppose we have a function which is bimodal with respect to both the natural encoding of Ω and a Gray-encoding, and that our local search (using the Gray-encoding) has become stuck at the non-global optimum. All the points of Ω that are better than our current point form an interval containing the global optimum. We will call this interval the *target*, T. If we could only move into the target interval, we would be able to find the global optimum by local search. The question is, can we find a shift in representation so that our current position has a neighbor in T?

Theorem 14 *Let our current point be $x \in \Omega$. Let T contain at least 2^b points, where $b > 0$ (so we have at least two points to aim at). Then there is a point $a \in T$ which is a neighbor of x under the encoding c_k, where $k = \lambda 2^{b-1}$ for some integer λ.*

Proof: Since T contains at least 2^b points, we can find $a \in T$ such that

$$a \equiv -(x+1) \pmod{2^b}$$

Then

$$a + x + 1 = \lambda 2^b$$

for some integer λ. Therefore

$$(a - \lambda 2^{b-1}) + (x - \lambda 2^{b-1}) = -1$$

Setting $k = \lambda 2^{b-1}$, this means that the points $a - k \bmod 2^\ell$ and $x - k \bmod 2^\ell$ are neighbors with respect to the reflected Gray code (they are reflections of each other around the center point). Therefore a and x are neighbors with respect to c_k. □

This theorem means that at least one of the shifts from the set

$$\{0, 2^{b-1}, 2 \cdot 2^{b-1}, 3 \cdot 2^{b-1}, \ldots, (2^{\ell-b-1} - 1) \cdot 2^{b-1}\}$$

is guaranteed to have a neighbor in the target interval. More generally, we have the following:

Theorem 15 *Given any $i \in \{0, 1, \ldots, 2^{b-1} - 1\}$, there exists $k \equiv i \pmod{2^{b-1}}$ such that x has a neighbor in the target T under encoding c_k.*

Proof: Consider the interval

$$T - i = \{w - i \bmod 2^\ell : w \in T\}$$

and the point $x - i$. By the previous theorem, there is an $a - i \in T - i$ which is a neighbor of $x - i$ under encoding c_j, where $j = \lambda 2^{b-1}$. Therefore a is a neighbor of x under encoding c_k where $k = i + j$, and so $k \equiv i \pmod{2^{b-1}}$. □

Thus the set of all distinct shifted Gray codes divides up into 2^{b-1} intervals, which are also quadrants, each of size $2^{\ell-b-1}$. At least one shift code in each partition induces a neighbor in the target interval. Picking a shift code at random gives a probability of at least

$$\left(\frac{1}{2}\right)^{\ell-b-1}$$

of choosing a shift that enables the search process to continue. This only counts shifts that result in neighbors being created as reflections around the point $2^{\ell-1}$. Depending on how close the target is to the local optimum, there may be many more shifts that create neighbors, by reflecting around other points.

We see that the chance of finding a shift that enables the search to continue is at least twice the probability of randomly finding a point in the target interval.

However, an alternative strategy for local search when it is stuck is simply to restart at a random point. Whether or not this will be more successful than the shift strategy will depend on the relative sizes of the basins of attraction of the two optima.

The assumption that we are at a local optimum in a bimodal problem is just for ease of understanding. We saw earlier that there are lots of shapes of functions which are unimodal with respect to a (shifted) Gray code. Any of these will do as targets to aim at. We had to assume, though, that the target contained at least two points. This is because to get a neighbor, you have to be able to balance the neighbors evenly on either side of a reflection point. When the target contains just one point (so the optimum is *isolated*) this cannot be guaranteed. If x is our current point and our target is $\{a\}$, and if the number of points in the interval $[a, x]$ is odd, then these points can never be neighbors under any shift. If the size of $[a, x]$ is even, then it is still possible.

4.3 Algorithms and Test Functions

We investigate the utility of incorporating shifting into both a simple hill-climbing algorithm, RBC, and a state-of-the-art genetic algorithm, CHC. Shift values are uniformly sampled from the set of unique shifts. We use the test functions described in Whitley et al. (1995, 1996), shown in Table 2 (along with the associated variable domains). These test functions range from simple, separable functions which are easily solved by hill-climbers to more complex non-linear, non-separable functions.

RBC (Random Bit Climber) (Davis, 1991) is a next-descent hill-climbing algorithm. Search begins from a random bit string, and proceeds by testing each of the ℓ Hamming-1 neighbors in some randomized order. Both equal and improving moves are accepted. If the search is stuck in a local optimum, it is re-started from a new random bit string. Otherwise, a new random visitation sequence of the ℓ neighbors is generated.

Once a local optimum is detected by RBC, the representation can be shifted to try to escape the local optimum. Search then proceeds from the new bit-string until convergence is re-established. We use two types of restarts: 'soft' and 'hard'. The 'soft' re-start merely changes the representation, without changing the current point in the search space. A 'hard' re-start reinitializes search from a new random point in the search space. Depending on the experiment, we perform either 10 or 50 soft restarts before each hard re-start.

The Hamming-1 neighborhood at the bit-string level translates into a neighborhood which is capable, in one step, of altering only a single parameter value. This restriction suggests that RBC should perform well on separable test functions, and perform worse on non-separable, non-linear test functions which may require simultaneous manipulation of multiple parameters to yield improvements in the evaluation function. These hypotheses are supported by our experimental results.

Table 2: The test functions described in Whitley et al. (1996).

$x_i \in [-5.12, 5.11]$	
Rastrigin	$F(x_i\|_{i=1,N}) = (N * 10) + [\sum_{i=1}^{N}(x_i^2 - 10cos(2\pi x_i))]$
$x_i \in [-512, 511]$	
Schwefel	$F(x_i\|_{i=1,N}) = \sum_{i=1}^{N} -x_i sin(\sqrt{\|x_i\|})$
Griewangk	$F(x_i\|_{i=1,N}) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N}(cos(\frac{x_i}{\sqrt{i}}))$
Powell	$F(x_1, x_2, x_3, x_4) = (x_1 + 10x_2)^2 + (\sqrt{5}(x_3 - x_4))^2 + ((x_2 - 2x_3)^2)^2 + (\sqrt{10}(x_1 - x_4)^2)^2$
EF101	$F(x, y) = -xsin(\sqrt{\|x - \frac{y+47}{2}\|}) - (y + 47)sin(\sqrt{\|y + 47 + \frac{x}{2}\|})$
Rana (EF102)	$F(x, y) = xsin(\sqrt{\|y + 1 - x\|})cos(\sqrt{\|x + y + 1\|}) + (y + 1)cos(\sqrt{\|y + 1 - x\|})sin(\sqrt{\|x + y + 1\|})$

The **CHC genetic algorithm** (Eshelman, 1991) maintains a parent population of size μ (= 50 in our experiments). CHC randomly pairs members of the parent population for reproduction. Once paired, reproduction is only permitted if the Hamming distance between the two parents is greater than some threshold value, resulting in a child population of size λ. The HUX crossover operator is used which ensures that each child is of maximal Hamming distance from the parents. The μ best of the $\mu + \lambda$ individuals form the parent population for the next generation. CHC *guarantees* survival of the best μ individuals encountered during the search.

CHC also uses a re-start mechanism if the parent population remains unchanged for some number of generations. During a re-start, a population containing μ copies of the best individual is formed; all but one copy undergo extensive mutation (35% of the bits). Shifting can be applied each time CHC converges. The best individual is re-encoded using a new, randomly selected shift value.

4.4 Selecting Shift Values

In general, we are interested in locating the global optimum of arbitrary (multimodal) real-valued functions. In terms of algorithm design, the question becomes: What is the best method to select a value k for shifting? The corollary of Theorem 13 indicates $1 \leq k \leq 2^{\ell-2} - 1$. However, as shown in Section 4.2, many of the values in this interval are redundant in that multiple values can collapse a particular local optimum. Consider the set of shifts $k = \lambda 2^{b-1}$ for some $\lambda \in \mathcal{Z}^+$ such that $1 \leq k \leq 2^{\ell-2} - 1$. The optimal value of b clearly depends on the function under consideration.

To analyze the potential impact of b on the performance of shifting, we apply RBC to 1-dimensional versions of the Rastrigin, Griewangk, and Schwefel test functions (see Table 2). We use 10-bit encodings for the Rastrigin and Schwefel functions, and a 20-bit encoding for Griewangk's function; no shift is able to collapse two of the local optima in 10-bit encodings of Griewangk's function (Barbulescu et al., 2000). We vary b from 2 to 10 in increments of 2 and execute 30 trials of RBC for each b on each test function, using 50 soft restarts between each hard re-start. In Table 3 we report the mean number of evaluations over the 30 trials required to locate the global optimum. The results

indicate that (1) the particular value of b can have a significant impact on performance and (2) the optimal value of b is, as expected, dependent upon the test function. For many of the test functions we consider, even small values of b can prevent selection of those values of k that are necessary to collapse particular local optima. Further, *a priori* identification of the optimal b is intractable. Consequently, we adopt a conservative strategy of *uniform shifting* in subsequent experiments, such that $b = 1$.

Table 3: Mean number of function evaluations over 30 trials to locate the global optimum using RBC.

	Mean Number of Evaluations to Optimal		
b	Rastrigin ($\ell = 10$)	Griewangk ($\ell = 20$)	Schwefel ($\ell = 10$)
2	184	34977	197
4	90	10855	109
6	412	2417	71
8	487	791	150
10	304	200131	164

4.5 Experimental Results and Discussion

We ran 30 trials of both RBC and CHC on each test function, allowing a maximum of 500K evaluations per run. With the exception of Powell's function (whose dimension was fixed at 4), 10-dimensional versions of each test function were used; each variable was encoded using 10 bits except for Powell's function, which used 20-bit encodings. For the Rana EF102 and EF101 functions, 10-dimension versions were constructed using the Weighted-Wrap expansion method, described in (Whitley et al., 1996). Two versions of CHC, both with and without shifting, were considered. We tested RBC without shifting, in addition to versions using both 10 and 50 soft restarts between hard-restarts, denoted shift-10 and shift-50 respectively. The results for RBC and CHC are reported in Tables 4 and 5, respectively. The #-Solved column indicates the number of times the algorithm found the optimal solution, out of the 30 total trials. The statistics for both solution quality and number of evaluations are taken over all 30 trials. All statistical comparisons between algorithms are made using two-tailed t-tests (significance level $p < 0.01$).

For RBC using 10 soft restarts (Table 4, shift-10), shifting yields a statistically significant improvement in both solution quality and number of evaluations for the Rastrigin, Schwefel, and EF101 test functions; for Rana's function, the improvement applies only to solution quality. For both the Powell and Griewangk test functions, no statistically significant difference in either measure was observed. For CHC (Table 5), shifting fails to yield any significant differences in mean solution quality. However, shifting does significantly reduce the number of evaluations required for the Rastrigin, Griewangk, and Powell test functions. While substantially improving the performance of both RBC and CHC, significant performance differences between the algorithms still exist. Next, we increase the number of soft restarts used with RBC (shift-50), and compare the resulting performance with CHC.

On Rastrigin's and Schwefel's functions, both versions of RBC (shift-10 and shift-50) and CHC found the global optimum in all trials. Both of these functions are separable. While there was no statistical difference between RBC-shift-10 and CHC, RBC-shift-50 required significantly fewer evaluations than CHC on these simpler problems.

Table 4: RBC results on six test functions, with 10 and 50 shift attempts before a hard re-start is performed. Results are averaged over 30 independent trials of 500K function evaluations each. All functions are being minimized.

Function	Experiment	Mean Sol.	σ	Mean Evals.	σ	#-Solved
Rastrigin	no shift	6.35088	1.34344	500000	0.0	0
	shift-10	0.0	0.0	38368	39201	30
	shift-50	0.0	0.0	4488	1384	30
Schwefel	no shift	-3860.79	116.343	500000	0.0	0
	shift-10	-4189.0	0.0	6888	5481	30
	shift-50	-4189.0	0.0	3588	1058	30
Griewangk	no shift	0.001894	0.007322	160318	150648	28
	shift-10	0.000818	0.00448	93488	99037	29
	shift-50	0.009069	0.013182	289021	193251	20
Powell	no shift	0.000258	6.009e-5	500000	0.0	0
	shift-10	0.000224	6.53e-5	500000	0.0	0
	shift-50	0.000235	6.95e-5	500000	0.0	0
EF101	no shift	-770.518	29.7811	500000	0.0	0
	shift-10	-905.879	43.9736	351004	187004	14
	shift-50	-909.183	45.014	303511	201526	19
Rana	no shift	-449.115	8.88101	500000	0.0	0
	shift-10	-469.255	7.58475	500000	0.0	0
	shift-50	-488.151	5.59812	500000	0.0	0

Using 50 instead of 10 soft restarts reduces the variance of the RBC results. The performance improvement can be explained, since all the local optima are collapsible under some shift. Ten and fifty soft restarts sample a maximum of 3.9% and 19.6% of the possible (255) unique shifts, respectively. Thus, sampling more shifts increases the chances of collapsing a particular local optimum. This explanation is further supported by empirical evidence: under shift-50, none of the Schwefel trials required a hard restart to find the global optimum, while only one Rastrigin trial required a (single) hard re-start.

Griewangk's function is the simplest non-separable test function considered: the function becomes easier (more parabolic) as the dimensionality is increased (Whitley et al., 1996). Intuitively, a highly parabolic structure should be easily solved by a hill-climbing algorithm. However, CHC significantly outperforms both versions of RBC for the number of evaluations. Furthermore, increasing the number of soft restarts resulted in poorer RBC performance. To explain this apparent anomaly, we examined the shifting properties of a 1-dimensional version of Griewangk's function. For each local optimum, we enumerated all possible shifts and recorded which shifts were able to collapse the local optimum. We found that the two best local optima were not collapsible by any shift - i.e., not collapsible with the global optimum. In addition, these local optima flank the global optimum; to find the global optimum, the initial starting point of RBC must be in its attraction basin to begin with, which is relatively small. If the precision is fixed at 10 bits, shifting is futile, and hard restarts are the only way to find the global optimum.

For both the Powell and EF101 functions, the performance of RBC shift-10 and shift-50 is indistinguishable (the increase in the number of trials identifying the global optimum of EF101's function is a statistical artifact), and both were strongly outperformed by CHC. Although no run of either CHC or RBC ever solved Rana's function to optimality, RBC-shift-50 significantly outperformed RBC-shift-10 in terms of solution quality. While CHC does slightly outperform RBC-shift-50, the difference is minimal

Table 5: CHC results on the six test functions, without and with a shifting attempt made each time the algorithm restarts. Results are averaged over 30 independent trials of 500K function evaluations apiece. All functions are being minimized, with the lowest mean solution or lowest mean number of evaluations in bold.

Function	Experiment	Mean Sol.	σ	Mean Evals.	σ	#-Solved
Rastrigin	no shift	0.0	0.0	34998	14365	30
	shift	0.0	0.0	22297	7299	30
Schwefel	no shift	-4189.0	0.0	9667	31579	30
	shift	-4189.0	0.0	7148	2224	30
Griewangk	no shift	0.0	0.0	58723	52101	30
	shift	0.0	0.0	24354	12890	30
Powell	no shift	0.0	0.0	200184	80041	30
	shift	0.0	0.0	96497	36756	30
EF101	no shift	-939.88	0.0	23798	9038	30
	shift	-939.88	0.0	25331	12881	30
Rana	no shift	-497.10	5.304	500000	0.0	0
	shift	-494.50	5.207	500000	0.0	0

(though still statistically significant). This result is particularly interesting, as Rana's function proved to be the most difficult for CHC.

Our results demonstrate that a deeper understanding of shifting can be used to significantly improve the performance of RBC. Looking at both solution quality and number of evaluations, RBC using shifting statistically outperforms CHC on Rastrigin's and Schwefel's separable functions, and nearly equals the performance of CHC on non-linear, non-separable problems (Griewangk and Rana's). But on Powell's and EF101, CHC outperforms RBC.

5 Conclusions

Developing a good representation is a key part of any search. There have long been questions in the evolutionary computation community about the relationship between Gray code representations, standard Binary and real valued representations. In this paper, a number of basic results related to the use of bit representations have been presented. The concept of equivalent encodings is important because the resulting equivalence classes can impact the behavior of different search algorithms in different ways.

This paper also presents several results related to the use of Gray codes. Convergence proofs are presented for special classes of unimodal functions under local search. These results have significance for the evolutionary computation community for two reasons. First, the relationship between local optima in Hamming space and the dynamic behavior of genetic algorithms, for example, is still being defined. Second, the convergence proofs in this paper were used to motivate the construction of a hybrid genetic-local search algorithm which has proven to be more effective than regular RBC or steepest ascent hybrids (Whitley et al., 2003).

The use of different Gray codes has also been explored. Dynamic representations are an alternative to mechanisms that escape local optima; such mechanisms include local search with restarts, tabu search, and simulated annealing. Shifting uses multiple Gray code representations to escape local optima. New upper bounds on the number of unique Gray code neighborhoods under shifting are established. We also characterized neighborhood structures under similar shifted Gray codes. By augmenting a simple hill-climber with a dynamic representation scheme, we achieve improved performance

on test functions which are difficult for a simple hill-climbing algorithm and a challenge to CHC, a state-of-the-art genetic algorithm.

Acknowledgments

Jonathan Rowe was funded by a University of Birmingham Pilot Research Grant. Support was also supported by the National Science Foundation under grant number IIS-0117209, and by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-00-1-0144. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation.

References

Barbulescu, L., Watson, J. P., and Whitley, D. (2000). Dynamic representations and escaping local optima. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*.

Bitner, J. R., Ehrlich, G., and Reingold, E. M. (1976). Efficient Generation of the Binary Reflected Gray Code and Its Applications. *Communications of the ACM*, 19(9):517–521.

Davis, L. (1991). Order-Based Genetic Algorithms and the Graph Coloring Problem. In Davis, L., editor, *Handbook of Genetic Algorithms*, pages 72–90, Van Nostrand Reinhold.

Eshelman, L. (1991). The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. In Rawlins, G., editor, *FOGA-1*, pages 265–283, Morgan Kaufmann.

Harary, F. (2000). The Automorphism Group of a Hypercube. *Journal of Computer Science*, 6(1):136–138.

Horn, J., Goldberg, D., and Deb, K. (1994). Long Path Problems. In Schwefel, H.-P. et al., editors, *Parallel Problem Solving from Nature, 3*, pages 149–158.

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

Rana, S. and Whitley, D. (1997). Bit Representations with a Twist. In Bäck, T., editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 188–195. Morgan Kaufmann.

Whitley, D. (2000). Functions as Permutations: Regarding No Free Lunch, Walsh Analysis and Summary Statistics. In Schoenauer, M. et al., editors, *Parallel Problem Solving from Nature, 6*, pages 169–178, Springer.

Whitley, D., Garrett, D., and Watson, J.P. (2003). Genetic Quad Search. In *GECCO 2003*. pages 1469–1480. Springer.

Whitley, D. and Rana, S. (1997). Representation, search and genetic algorithms. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*.

Whitley, D., Barbulescu, L., and Watson, J. P. (2001). Local Search and High Precision Gray Codes. In *Foundations of Genetic Algorithms (FOGA)-6*. Morgan Kaufmann.

Whitley, D., Mathias, K., Rana, S., and Dzubera, J. (1995). Building Better Test Functions. In Eshelman, L., editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, Morgan Kaufmann.

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

Winston, P. (1984). *Artificial Intelligence (2nd ed)*. Addison-Wesley.

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

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 4:67–82.