# Bit Representations with a Twist

**Soraya B. Rana**
Computer Science Department
Colorado State University
Fort Collins, CO 80523
email: rana@cs.colostate.edu

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

## Abstract

When a function is mapped onto a bit representation, the structure of the fitness landscape can change dramatically. Techniques such as Delta Coding have tried to dynamically adapt the representation during the search process in hopes of making the problem easier for a genetic algorithm to solve. We introduce a new technique for changing between a restricted set of Gray representations during search; the method involves little overhead but can offer a significant performance improvement for a variety of search algorithms.

## 1 Introduction

The No Free Lunch Theorem (NFL) [WM95] proves (subject to certain assumptions) that on average, all search algorithms have the same performance over the set of all functions. Radcliffe and Surry [RS95] extend the NFL results to include all possible representations as well. That is, if we look at all search algorithms over a *single* function and all possible representations, then all search algorithms will still perform the same on average.

A caveat of the No Free Lunch Theorem is that practitioners are generally not trying to optimize "all functions". Assuming that we are trying to solve a select subset of functions, how can we choose a binary representation that makes those functions more readily solved by the search algorithm we use? Without having *a priori* information about the specific function being optimized, that question cannot be answered definitively.

Most often, genetic algorithm researchers using a bit representation choose between a standard Binary Reflective Gray encoding and a standard Binary encoding. The intuition for either choice is that one encoding strategy makes the functions we are interesting in solving more amenable to search via genetic algorithms. In this paper, we make two conjectures. First, the choice of a Gray encoding is *usually* a better choice than Binary for most test functions and real world applications. Second, given some Gray representation, it is possible to improve search performance by searching over multiple **shifts** of the representation, thereby sampling a small subset of Gray representations.

This paper also introduces a new form of local search called *Optima Linking*. Optima Linking is a mechanism for escaping a local optimum without using probabilistic non-improving moves like those found in Simulated Annealing [KJV83] or complex "Tabu" mechanisms as found in Tabu search [Glo94].

## 2 Background

This paper is built around two previous theoretical papers. In the first paper, Rana and Whitley [RW97] calculate the expected number of optima induced over all possible bit representations of a function; the paper also illustrates how to calculate the probability that any point is a local optimum over all possible representations. These values are then compared specifically to the number of optima that occur in a Binary Reflective Gray coding [MW94] and a standard Binary encoding for some commonly used test functions. For all test functions examined, the number of minima in a Gray encoding is less than a Binary encoding, but both Gray and Binary produce dramatically fewer minima than the average over all possible representations. This can be explained by the limited degree of nonlinearity and the locally correlated structure of the functions analyzed.

In the second paper [WR97], it was shown theoretically that Gray coding is better than Binary encoding for a special class of functions with a bounded number of optima. Gray coding will always induce fewer local minima than Binary coding on average over all functions in this special class. Of course, having a smaller number of local minima under one bit representation does not guarantee that the function is actually easier than another representation that induces more minima. Nevertheless, empirically the use of Gray codes often result in better search – both for local bit-climbing methods as well as genetic algorithms. In this paper we empirically apply the results of this theoretical work and look at a method that dynamically explores different Gray representations of an objective function during search. We apply this dynamic strategy in conjunction with two forms of local search, as well as the CHC algorithm.

## 3 Gray Coding

We begin by discussing our first conjecture: Gray coding is the better choice for a bit representation for problems with a limited degree of nonlinearity and a locally correlated structure. Assume an integer domain 0 to $2^L - 1$ for an arbitrary function. Any point in the domain can be represented using an $L$ bit string. A Gray code is defined as a binary encoding scheme that guarantees that points that are adjacent in the integer domain have a Hamming distance of one.

Keeping that property in mind, we now count the number of local optima that occur in the integer neighborhood versus the number of local optima that occur under a Gray representation. The integer neighborhood is such that adjacent integers are neighbors and the space wraps around so that 0 is adjacent to $2^L - 1$. A value is locally optimal if no neighboring points are better than the current value (for simplicity, assume no duplicates). Suppose we have $K$ locally optimal points in an integer neighborhood representation of the domain. Since Gray Coding preserves the numeric adjacency of the domain, there are no more than $K$ locally optimal points under Gray encoding. We refer to this as the *Gray Compactness Theorem*:

**THEOREM:** *The Gray-Compactness Theorem*
Let $X$ be the number of local optima induced by a Hamming distance one neighborhood search operator over the bits in any Gray coded representation of a function in $F$. Let $Y$ be the number of local optima induced under the integer neighborhood topology of the same function in $F$: $X \leq Y$.

A formal proof is given by Whitley and Rana [WR97]. Simply put, Gray coding results in a function landscape that is *at worst* as difficult as the landscape induced under an integer neighborhood search operator. A Binary encoding offers no such guarantee. In fact, for simple functions Binary encodings usually increase the number of optima compared to the integer neighborhood. The following table contains a comparison of Gray and Binary representations over the set of all possible 3-bit representations.

| K | No. of F with K minima | No. of Min in Gray | No. of Min 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 |

Since we are considering 3-bit representations, we begin by generating all possible functions (permutations) containing 8 distinct values. This set of functions is called $F$. We partition the set $F$ according to how many local optima occur in the function where $K$ is the number of optima. The number of functions falling into each partition is given in column two of the table. All of the functions in $F$ are then mapped onto the corresponding Gray and Binary representations. The total number of optima (i.e. Min) that occur under the bit representations for each partition are given in the third and fourth columns. When $K$ is small, the Gray representations consistently produce fewer total optima; however, as $K$ approaches the maximum number of possible optima, the number of optima in the Binary representation quickly drops. This trend continues for larger problem sets.

If we assume that the set of functions we are interested in solving have a bounded number of optima, then Gray Coding appears to be the better encoding choice.

## 4 Shifting

We have provided evidence to support the choice of a Gray encoding for arbitrary functions. However, there are many different Gray encodings one can choose from – we speculate there are $O(L! \times 2^L)$ different Gray codes. Even the different Gray encodings will result in different bit interactions and consequently, different Gray encodings will result in fitness landscapes of varying difficulty. Since there is no practical way of knowing which encoding will make a problem the easiest, we introduce a new way of sampling many different encodings during search. This should allow for a search algorithm to exploit the different encodings and ultimately arrive at good solutions with less work.

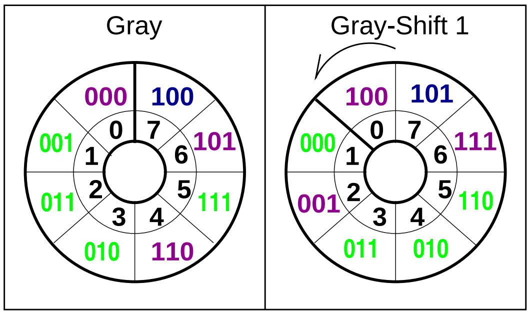


Figure 1: A simple example of Shifting.

**Shifting** is a technique for altering the encoding and decoding of bit strings during search. When a bit string is encoded or decoded, the string is mapped to an integer value and appropriately scaled to the function domain. A **shift** is an integer offset that is applied to the intermediate integer result that is generated during the decoding process. The shifted result is wrapped around the ends of the valid integer domain then scaled to the function domain. Figure 1 provides an example of a single shift applied to all 3-bit Gray coded strings. The outer dial contains all 3-bit strings while the inner dial contains the integer values that map onto the bit strings. A shift occurs as a counterclockwise turn of the outer dial – creating a new mapping.

Shifting can be applied with any bit encoding method, but the effects of shifting can be best understood using a Gray encoding. Each shift corresponds to a new Gray encoding; therefore, all shifts still preserve the adjacency that occurs in the integer neighborhood. However, for any string of length $L$, only two of its neighbors will be the points that are numerically one above and one below its value. The remaining $L-2$ neighbors can change as a result of shifting. This change can cause the total number of optima that occur to change over different shifts.

Looking back to the example in Figure 1, the neighbors of the point with integer value 7 change with a single shift. The integer 7 changes from 100 to 101 and the neighbors change from 000, 101, and 110 (mapping to 0, 6 and 4) to 100, 111, and 001 (mapping to 0,6 and 2). The adjacent integers 0 and 6 must remain as neighbors. The change in the set of the other $L-2$ neighboring points is more dramatic for longer bit-strings.

### 4.1 An Example of Shifting

We have done some preliminary analyses to determine how shifting can change the difficulty of common test

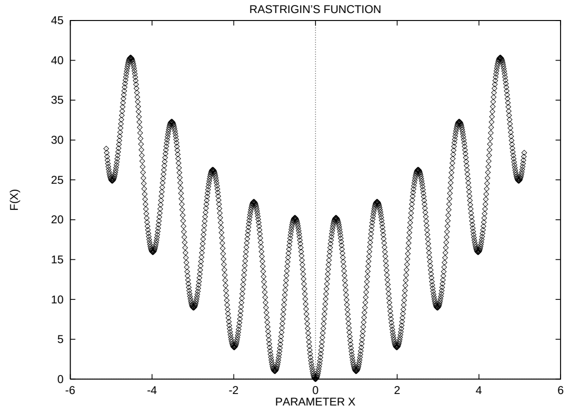


Figure 2: A plot of Rastrigin's function.

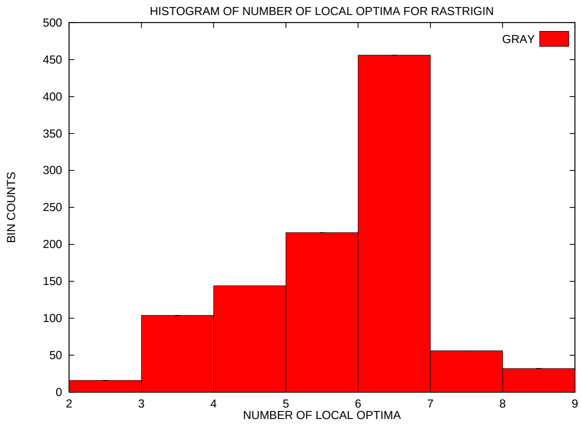


Figure 3: Histogram of the number of optima that occur under all shifts.

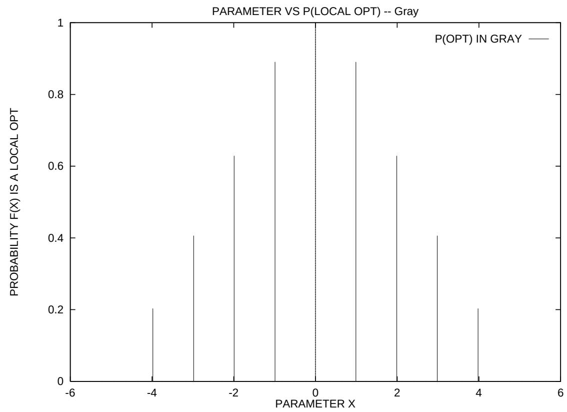


Figure 4: Probability that any point occurs as an optimum under all shifts.

problems. In particular, we looked at Rastrigin's function encoded using a single 10 bit parameter that was coded using Binary Reflective Gray Coding. A plot of Rastrigin's function is shown in Figure 2. From the graph, it is easy to count the 11 local minima that occur in numeric space. We counted the number of local optima that occur under all possible (1024) shifts of the Gray encoding. The results are given in the histogram shown in Figure 3. There are 5 local minima on average with a range from 2 to 8 local minima over the set of all shifts representing a subset of all possible Gray representations. This illustrates that this set of Gray codes results in representations of varying difficulty. Figure 4 shows the probability that any point in the space is a local optimum over all shifts. Looking at the set of points that occur as local optima over all possible shifts, there are only 9 values in the entire search space that occur as local optima. However, only one local optimum occurs over all shifts of the space and that is the global optimum.

Additional experiments were run for a shifted Binary space. The number of minima under the Binary representation of Rastrigin's function is much larger than for any of the Gray codes: there are 19 optima on average. This implies that Rastrigin's function becomes harder under a Binary representation than a Gray representation or a numeric representation. We have found similar results for other common test functions: a Binary representation induces more minima than almost any Gray representation.

# 5 Experiments

We ran three search algorithms on a test suite of six functions. Each search algorithm was run with and without shifting. The test suite functions are shown in Table 1. For Rastrigin's function, Schwefel's function and Griewangk's function [M$\ddot{9}$1] the dimension of the test problems was 10. For both Rana and Whitley's functions, the Weighted-Wrap expansion method [WMRD96] was employed to expand these functions to more than two variables. The wrap method consecutively pairs off the parameters that are input into the function, wrapping around the last parameter. For a three parameter version of a wrapped function, the expansion would be: $EF(x_1, x_2, x_3) = F(x_1, x_2) + F(x_2, x_3) + F(x_3, x_1)$. Random weights were then multiplied to each term before the summation. For both Rana and Whitley's functions, a 10 parameter version of the function was used. All five of these functions used a 10 bit Gray encoded string for each of the 10 parameters. The remaining function is known as the Powell singular function [MGH81]. For our experiments, we encoded each of the four parameters using a 20-bit Gray coded string.

## 5.1 Algorithms

Three algorithms were used to study the effects of random shifting on search. The first two algorithms, RBC and Optima Linking, illustrate the effects of shifting on hillclimbing search strategies. The third algorithm, CHC, is an illustration of how search can affect one type of genetic algorithm.

The most intuitive way to apply shifting is to allow a search algorithm to converge upon a solution using one representation then change the shift to a new representation and continue searching. In the case of hillclimbing algorithms, this technique can be very effective. Once a hillclimbing algorithm converges to a local optimum, the shift can sometimes cause that point to no longer be a local optimum, which means that the new optimum it converges upon must have a better evaluation. In the case of genetic algorithms, it is possible to run several generations and change the shift periodically.

## 5.2 RBC

The simplest algorithm we tested is the random bit climber introduced by Davis [Dav91]. Davis' random bit climber (RBC) starts by changing one bit at a time beginning at a random position. When an improvement is found, it is accepted. After the climber has flipped every bit in the string, a new random sequence is chosen for testing the bits and RBC again checks every bit for an improvement. If RBC has checked every bit in the string and no improvement is found, a local optimum has been found and RBC is restarted from a new random point in the search space.

The most appropriate time to change representations is when RBC converges to a local optimum. The process of changing shifts in RBC is actually interleaved with random restarts since random restarts are still needed for most problems. RBC is allowed to converge over 10 shifts before it is restarted.

## 5.3 Optima Linking

Optima Linking is a new search technique designed by Whitley [WCR96]; the algorithm is similiar to an idea outlined by Glover [Glo94] and Reeves [Ree97]. Optima Linking utilizes a simple local search technique, such as RBC, to generate a set of locally optimal points in the search space. From this set, two points are selected and used as guides for the local search algorithm. For example, if the local optima are 110010010 and 101000101, then both strings are in the subspace that corresponds to the schema 1**0*0***. Search

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

Table 1: Test Suite used for studying the effects of shifting.

between these points is restricted to the subspace between them. Search moves from a start point to a destination point, thus creating a path between the two optima. From the start point the best available move is taken that is one step closer to the destination. Note that this will always be a nonimproving move, since the start point is a local optimum. The search continues to take the best available move from the current point toward the destination point. After the destination is reached, the start and destination points are swapped and a second linking path is generated. The best point along each path is used as an initial starting point for an unrestricted L-neighborhood search using RBC. The result of the search will be a local optimum. If the local optimum is not already in the set of known optima it is copied and saved for future pairings. Any pair of optima are linked only once.

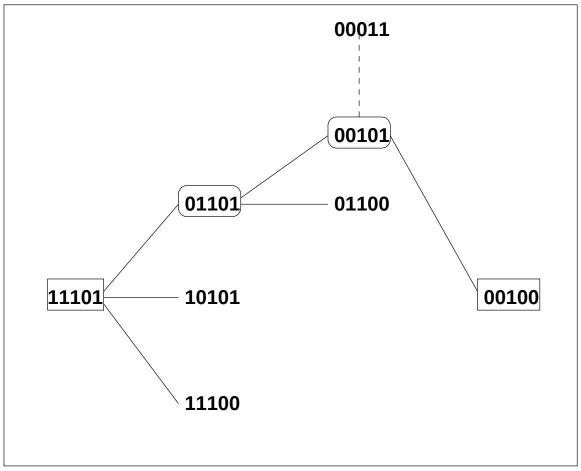


Figure 5: An example of Optima Linking working from the point 11101 to 00100.

An example of Optima Linking is given in Figure 5. In this example, two locally optimal strings 11101 and 00100 are being paired. There are three differences between the strings occurring in bit positions 1, 2 and 5, numbering the bits from left to right. Starting from 11101 a steepest descent search over all three strings that are one bit away and which fall in the subspace **10* are examined. Of these three strings, 01101 is the best string (but of course, it is not as good as 11101). Continuing from the point 01101, the subspace between 01101 and 00100 is 0*10* and the two neighbors 00101 and 01100 that lie in this subspace are compared. The string 00101 is found to be the best of the two neighbors and is actually the best string along the entire path between 11101 and 00100. The point 00101 is adjacent to the destination string 00100 so the linking path is complete. A local search is applied from the best point along the path, in this case 00101 which results in a new optimum.

Optima Linking is a way for a local search technique to escape a local optimum without using a random restart or complex Tabu mechanisms. We believe that Optima Linking has merit not only as a search technique, but also as a tool for mapping out complex search spaces. Optima Linking also offers the advantage that it does not resample as many points as some other search algorithms such as simulated annealing or even Tabu search. Once two points have been linked, they are never linked again. Since only unique pairs of optima are linked, it is unlikely that the same path will be traversed more than once.

To add shifting to Optima Linking, a random shift was applied for each path generated between two optima. Note that there are two linking paths and hence two

| Func | Shift Used? | Mean Soln | Stddev | Mean Evals | Stddev | Number Solved |
|---|---|---|---|---|---|---|
| Rastrigin | | 6.457 | 1.246 | – | – | 0 |
| | ✔ | 0.000 | – | 6334 | 6222 | 30 |
| Schwefel | | -3809 | 71.152 | – | – | 0 |
| | ✔ | -4189 | – | 2042 | 1048 | 30 |
| Griewangk | | 0.004 | 0.009 | 151472 | 127193 | 25 |
| | ✔ | 0.006 | 0.012 | 56571 | 62266 | 23 |
| Powell | | 0.0002 | $7.7 \times 10^{-5}$ | – | – | 0 |
| | ✔ | 0.0002 | $6.4 \times 10^{-5}$ | – | – | 0 |
| Whitley | | -780.53 | 38.435 | – | – | 0 |
| | ✔ | -912.40 | 39.708 | 189684 | 141191 | 18 |
| Rana | | -450.20 | 8.770 | – | – | 0 |
| | ✔ | -468.27 | 5.927 | – | – | 0 |

Table 2: Results for RBC with and without shifting on the test suite.

shifts for each pair of optima that are connected. Under a single representation, all points in the initial set of local optima obviously remain locally optimal; however, this is not necessarily the case when shifting is applied. Different shifts can induce different sets of locally optimal points. Since all points in the pool are deemed locally optimal under *some* shift, they tend to be points that have a fairly good evaluation.

### 5.4 CHC

The third algorithm we applied to the test suite was CHC. CHC employs a parent population of size $\mu$ but instead of selecting highly fit parents for recombination as is typical of most genetic algorithms, the parents are randomly and uniformly paired and conditionally mated to produce $\lambda$ offspring. The algorithm then chooses the $\mu$ best strings from the combined parent and offspring populations as the next generation of reproducing parents. Thus, the CHC algorithm maintains the best $\mu$ strings that have been encountered over the course of the search.

CHC also has a restart mechanism that is used when the parent population has converged to a solution. When CHC is restarted, the single best member of the population is used to seed the remainder of the population. Shifting is applied during the restart phase. This means that only one string needs to be decoded and recoded in a new shift – making the change in shift very simple and quick to perform.

## 6 Results

All algorithms were run 30 times with different random seeds. Each run was allowed a maximum of 500K evaluations. The purpose of these experiments is not to compare algorithms, but to illustrate how shifting can affect the performance of different algorithms on different types of functions. The Rastrigin and Schwefel functions are separable, multimodal functions with a relatively small number of local optima in numeric space. Griewangk, Powell, Whitley and Rana's functions are not separable and appear to offer varying levels of difficulty to all three algorithms.

Performance was measured using solution quality and number of evaluations. When the algorithms converge to the correct solution, performance is measured in the number of evaluations required to find the correct solutions. When the algorithms did not converge to the correct solution, performance is measured in terms of the quality of the solutions found. The performance of each algorithm run with and without shifting was compared using a Students' t-test with $p < 0.05$.

### 6.1 RBC

Two versions of RBC were run on the six test suite functions. One version of RBC used random restarts and no shifting. The second version of RBC used random restarts and shifting. Shifting was applied 10 times before a random restart could occur. This allows RBC to search from a local optimum under a variety of shifts before restarting. The results of running both versions of RBC on the six test suite functions are given in Table 2.

Without shifting, RBC was unable to solve Rastrigin or Schwefel's function; however, with the use of shifting, RBC solved both functions on every run. While RBC was able to solve Griewangk's function with and without shifting, it was solved with significantly fewer evaluations when shifting was used. The performance of RBC on Powell's function was not significantly af-

| Func | Shift Used? | Mean Soln | Stddev | Mean Evals | Stddev | Number Solved |
|---|---|---|---|---|---|---|
| Rastrigin | | 2.632 | 1.472 | 374206 | 25822 | 2 |
| | ✔ | 0.000 | 0.000 | 40891 | 26898 | 30 |
| Schwefel | | -4181 | 30.05 | 78425 | 61017 | 28 |
| | ✔ | -4189 | – | 9531 | 7095 | 30 |
| Griewangk | | 0.033 | 0.027 | 128573 | 157921 | 9 |
| | ✔ | 0.004 | 0.011 | 134940 | 122485 | 25 |
| Powell | | 0.0002 | $7.8 \times 10^{-5}$ | – | – | 0 |
| | ✔ | 0.0002 | $6.9 \times 10^{-5}$ | – | – | 0 |
| Whitley | | -860.81 | 70.108 | 233206 | 152641 | 12 |
| | ✔ | -910.38 | 51.147 | 202289 | 165207 | 22 |
| Rana | | -473.31 | 11.223 | – | – | 0 |
| | ✔ | -480.31 | 7.898 | – | – | 0 |

Table 3: Results for Optima Linking with and without shifting on the test suite.

| Func | Shift Used? | Mean Soln | Stddev | Mean Evals | Stddev | Number Solved |
|---|---|---|---|---|---|---|
| Rastrigin | | 0.000 | 0.000 | 41767 | 17711 | 30 |
| | ✔ | 0.000 | 0.000 | 17999 | 6761 | 30 |
| Schwefel | | -4189 | – | 10033 | 3519 | 30 |
| | ✔ | -4189 | – | 6716 | 2920 | 30 |
| Griewangk | | 0.000 | 0.000 | 44073 | 27561 | 30 |
| | ✔ | 0.000 | 0.000 | 22723 | 15226 | 30 |
| Powell | | $5.5 \times 10^{-9}$ | $8.2 \times 10^{-9}$ | 260991 | 123540 | 20 |
| | ✔ | 0.000 | – | 61519 | 23512 | 30 |
| Whitley | | -939.87 | – | 26204 | 11861 | 30 |
| | ✔ | -939.87 | – | 30957 | 18027 | 30 |
| Rana | | -494.60 | 6.156 | – | – | 0 |
| | ✔ | -492.04 | 7.877 | – | – | 0 |

Table 4: Results of CHC run with and without shifting on the test suite.

fected by the use of shifting. For Whitley and Rana's functions the use of shifting resulted in significant improvement. Whitley's function was able to be solved 18 of 30 runs when shifting was added while it was unable to be solved without shifting. Rana's function was unable to be solved by either version of RBC, but the solution was significantly lower for the version of RBC that used shifting.

### 6.2 Optima Linking

Optima Linking utilizes RBC as the hillclimbing search algorithm, but the performance of Optima Linking is very different from RBC. On some problems, such as Griewangk's function, the random restarts provide RBC with an advantage over Optima Linking. Figure 3 contains the results for running Optima Linking with and without shifting on the six test suite functions. Just glimpsing at the table, one can see that the set of problems that were solved by Optima Linking differ from those solved by RBC.

For Rastrigin and Griewangk's functions, shifting caused Optima Linking to perform significantly better in terms of the number of times it found the global minimum. For Schwefel's function, shifting significantly reduced the number of evaluations required to solve the problem. For Powell's function, the quality of the solutions found by Optima Linking were not affected by the use of shifting. The results for Whitley and Rana's functions were significantly affected by the use of shifting. Whitley's function was solved more often when shifting was used. Optima Linking was found better solutions on Rana's functions when shifting was used.

### 6.3 CHC

One version of CHC was run without shifting and the other version used shifting every generation (when the population was reseeded). The results are given in Table 4. For Rastrigin, Schwefel and Griewangk's functions, the shifting resulted in significantly fewer evaluations required to solve the problem. For Powell's function, CHC with shifting solved the problem significantly more often than without the use of shifting. For Whitley and Rana's functions, there was no significant difference in performance.

For all three algorithms, the addition of shifting either did not affect the overall performance of the algorithm or it significantly improved the performance. These results imply that the effects of shifting can be exploited by algorithms; the changes in basins of attraction and changes to the number and distribution of local optima can be effectively utilized by a variety of search algorithms.

## 7 Conclusion and Future Work

The goal of this paper has been twofold. The first goal has been to further encourage the use of Gray coding as a default bit representation. The second goal has been to introduce a new technique, shifting, for dynamically changing Gray codes (or any bit representation) during search. We have provided preliminary results that indicate that shifting can be used effectively by several types of algorithms on a variety of problems. The performance of the algorithms that used shifting was either significantly improved or unchanged – indicating that the use of shifting is not likely to hurt the performance of the algorithm.

Shifting can be performed with very little overhead. The shifting offset is applied during the encoding and decoding phases of function evaluation and is simply a numeric offset to the encoded or decoded value. However, changing shift can be expensive if it is performed frequently on large populations. Further study needs to be done to determine the effectiveness of shifting in genetic algorithms that utilize large populations.

**Acknowledgements**
This research was supported by NSF grants IRI-9312748 and IRI-9503366 and the Colorado Advanced Software Institute. Soraya Rana was also supported by a National Physical Science Consortium fellowship.

## References

[Dav91] L. Davis. Bit-climbing, representational bias, and test suite design. In L. Booker and R. Belew, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 18–23. Morgan Kauffman, 1991.

[Glo94] F. Glover. Tabu search fundamentals and uses. Technical report, University of Colorado at Boulder, CU Boulder, 1994.

[KJV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 1983.

[M9̈1] H. Mühlenbein. Evolution in time and space: The parallel genetic algorithm. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337. Morgan Kaufmann, 1991.

[MGH81] J. J. More, B. S. Garbow, and K. E. Hillstrom. Testing unconstrained optimization software. *ACM Transactions on Mathematical Software*, 7(1):17–41, March 1981.

[MW94] K. E. Mathias and D. Whitley. Changing representations during search: A comparative study of delta coding. *Journal of Evolutionary Computation*, 2(3):249–278, 1994.

[Ree97] C. Reeves. Personal Communication., 1997.

[RS95] N. J. Radcliffe and P. D. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In Jan van Leeuwen, editor, *Lecture Notes in Computer Science*, volume 1000. Springer-Verlag, 1995.

[RW97] S. Rana and D. Whitley. Search, binary representations and counting optima. CSU Technical Report, 1997.

[WCR96] D. Whitley, R. Cogswell, and S. Rana. A comparative study of search methods for seismic problems. Technical report, CASI, 1996.

[WM95] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, July 1995.

[WMRD96] D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Evaluating evolutionary algorithms. *Artificial Intelligence Journal*, 85, August 1996.

[WR97] D. Whitley and S. Rana. Representation, search and genetic algorithms. AAAI, 1997.