Building Better Test Functions

D. Whitley, K. Mathias, S. Rana and J. Dzubera
Department of Computer Science
Colorado State University
Fort Collins, Colorado 80523 USA
(303) 491-5373
whitley,mathiask,rana,zube@cs.colostate.edu

Abstract

We introduce basic guidelines for developing test suites for evolutionary algorithms and examine common test functions in terms of these guidelines. Two methods of designing test functions are introduced which address specific issues relevant to comparative studies of evolutionary algorithms. The first method produces representation invariant functions. The second method constructs functions with different degrees of nonlinearity, where the interactions and the cost of evaluation scale with respect to the dimensionality of the search space.

1 Introduction

This paper explores methodologies for designing and evaluating parameter optimization problems which can be used for comparing the effectiveness of evolutionary algorithms. The end result is not a *new* test suite but rather new methods for designing test functions that address specific issues relevant to comparative studies.

Current evolutionary algorithm test suites contain several problems that are *separable*; problems are separable if there are no nonlinear interactions between variables. Separable functions may be nonlinear in that the objective function may involve nonlinearities when determining the contribution of a single variable to the overall evaluation. Nevertheless, the optimal value for each parameter can be determined independently of all other parameters. This is problematic in that separable functions are often readily solved by local search methods. For this reason we would argue that test problems should not be separable.

One of the reasons that separable functions are commonly used is that they are scalable. Scalability is indeed desirable, but the nonlinear interactions in a test function should also be sensitive to scaling. We introduce simple methods for constructing test functions with N parameters that allow nonlinear interactions between variables to be selectively scaled between $O(N)$ and $O(N^2)$ as the dimensionality of the problem is increased.

A final consideration addressed here is the use of BCD (Binary Coded Decimal) and binary reflected Gray encodings as discrete problem representations. We show that it is possible to generate functions that are invariant under both BCD and Gray encodings.

2 Evaluating Evolutionary Algorithms

DeJong's (1975) test suite has continually been used as a standard for measuring the performance of various genetic algorithms. Although other test functions have been introduced over the years (Ackley 1987; Schaffer 1989; Muhlenbien 1993) none has received the attention of the DeJong test suite. Several of these test functions are listed in Table 1.

2.1 Minimal Guidelines for Test Problems

The following principles represent ideas that most evolutionary algorithm researchers might accept as desirable properties for test functions. Ironically, most commonly used test problems fail to satisfy these criteria. We use these guidelines to construct new functions with the goal of doing a better job of evaluating and comparing evolutionary algorithms.

1. Test Suites Should Contain Problems Resistant To Hill-Climbing. Any test suite used for comparison purposes should be composed largely of problems that are resistant to hill-climbing strategies. Additionally all problems used for comparison purposes should be benchmarked using hill-climbing strategies. When hill-climbing strategies are successful, they are typically faster than evolutionary algorithms and have less algorithmic overhead. On the other hand, functions which may be hill-climbed are valuable for some types of comparisons, particularly for testing evolutionary algorithms to ascertain the existence of possible hill-climbing components.

2. Test Suites Should Contain Nonlinear, Nonseparable Problems. Many of the test functions currently found in the genetic algorithm literature are "separable" functions. Comparative testing on separable functions can produce misleading results and may lead experimental researchers to draw dubious conclusions about the relative merits of various types of evolutionary algorithms. Nonlinear *separable* problems can be decomposed into a linear combination of independent nonlinear subfunctions. An example of a separable function is:

$$F(x, y, z) = S(x) + S(y) + S(z)$$

Given a function F, which decomposes into n separate subfunctions, S, where each subfunction is coded using k bits, the total search space has a size of 2^{nk}. However, one can find the global optimum of F by searching the $n2^k$ points that define the n different subfunctions. Therefore, only $O(n)$ function evaluations are needed to solve F. A problem with 100 parameters coded using 10 bits each has a total search space of 2^{1000}; however, to generate the globally optimal solution by enumerating the individual subfunctions requires at most $100 * 2^{10}$ evaluations. Each subspace can be completely enumerated, thereby avoiding local optima. This characteristic often allows stochastic search methods to move the search into the basin of attraction of the global optimum of that subfunction.

3. Test Suites Should Contain Scalable Functions. Functions in current test suites that are scalable also tend to be separable functions. Problems that are not separable tend to be non-scalable and are relatively small. One should be able to variably scale up problems over a large number of parameters. Additionally, the difficulty of the problem and the function evaluation time should also scale up as the dimensionality of the problem increases.

4. Test Problems Should Have a Canonical Form. Close examination of past comparative studies indicates that problem representations often vary. First, the problems may be represented as binary or real-valued strings. Second, some researchers convert the BCD representations into a Gray coded representation. Finally, even if two representations use the same binary form, the method for mapping the bit strings onto real-valued numbers often differs. Such changes in representation have a potentially dramatic impact on search algorithms. Different problem representations induce different numbers of local optima with different sized attraction basins. If one is trying to solve a particular problem then changing the representation may be reasonable and advantageous. However, for comparison purposes, these differences alter the difficulty of the problem.

2.2 Testing Strategies and Baseline Comparisons

To evaluate our test suite construction strategies, we consider three forms of hill-climbers and two forms of evolutionary algorithms. The hill-climbers include next-ascent random bit climbing (RBC) as defined by Davis (1991) , random mutation hill-climbing (RMHC) defined by Forrest and Mitchell (1993) and what we refer to as a *line search* algorithm.

RBC starts by changing 1 bit at a time beginning at a random position. The sequence in which the bits are tested is also randomly determined. Each change that results in an improvement is kept. If RBC checks every bit in the string and no improvement is found, RBC is restarted from a new random point in the space.

RMHC uses a "mutation" operator to make random changes to a single string. Every bit in the string is mutated with a low probability; any improvements are accepted. RMHC was run with a fixed mutation rate on all problems (i.e. 2/L, where L is the length of the string). In every case RMHC yielded poorer results than RBC; therefore, only results for RBC are reported.[1]

The line search algorithm is designed to exploit the separability of functions by searching over all values of each parameter, one parameter at a time. Given a function which accepts n variables that are coded using k bits, line search checks each of the 2^k points that are associated with each of the n variables. The representation need not be binary; any discretization of the variables will suffice.

The two evolutionary algorithms that we applied were the CHC adaptive search algorithm (Eshelman, 1991) and an elitist simple genetic algorithm with tourna-

[1]We have found that the performance of RHMC can be improved by spending a significant amount of time to tune the mutation rate, but this was impractical for the large number of experiments which we ran.

$F1: f(x_i \mid_{i=1,3}) = \sum_{i=1}^{3} x_i^2$	$x_i \in [-5.12, 5.11]$
$F2: f(x_i \mid_{i=1,2}) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$x_i \in [-2.048, 2.047]$
$F6: f(x_i \mid_{i=1,N}) = (N * 10) + \left[\sum_{i=1}^{N} \left(x_i^2 - 10cos(2\pi x_i)\right)\right]$	$x_i \in [-5.12, 5.11]$
$F7: f(x_i \mid_{i=1,N}) = \sum_{i=1}^{N} -x_i sin(\sqrt{\lvert x_i \rvert})$	$x_i \in [-512, 511]$
$F8: f(x_i \mid_{i=1,N}) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N}(cos(x_i/\sqrt{i}))$	$x_i \in [-512, 511]$
$F9: f(x_i \mid_{i=1,2}) = 0.5 + \frac{sin^2\sqrt{x_1^2+x_2^2} - 0.5}{[1.0+0.001(x_1^2+x_2^2)]^2}$	$x_i \in [-100, 100]$
$F10: f(x_i \mid_{i=1,2}) = (x_1^2 + x_2^2)^{0.25}[sin^2(50(x_1^2 + x_2^2)^{0.1}) + 1.0]$	$x_i \in [-100, 100]$

Table 1: Common Test Functions.

ment selection (ESGA-T) (Goldberg et. al, 1990). The population size for the ESGA-T was 50. Recombination is accomplished using a 2-point reduced-surrogate crossover operator (Booker, 1987) applied with probability 0.7. Mutation is applied to each bit with a probability of 1/L. CHC was run with a population of 50.

All of the parameter optimization problems used here are posed as minimization problems with optimal solutions of 0. Algorithms are compared for significant differences in terms of the value of the mean solution using a 2-tailed Student's t-test with a 95% confidence interval. Algorithms are also compared for significant differences in terms of how often the global optimum is found using a Chi-square (χ^2) test with a 99% confidence interval. These measures may be sensitive to the amount of time that algorithms are allowed to search, particularly when it is possible to quickly locate globally competitive solutions but more difficult to locate the optimal solution.

2.3 Limitations of Existing Test Problems

The DeJong test functions are typically labeled F1 through F5. The Rastrigin (F6), Schwefel (F7) and Griewank (F8) functions (Mühlenbien, 1993) , shown in Table 1, can be scaled to use any number of variables. Functions F9 and F10 are the sine envelope sine wave and the stretched V sine wave functions (Schaffer, 1989). These test problems have often been used to tune and refine variants of a single algorithm and to argue the superiority of one approach over another. Such an approach is potentially dangerous in that algorithms can become customized for a particular set of test problems; this is especially disconcerting if the test problems are not characteristic of the types of problems for which evolutionary algorithms would be appropriate.

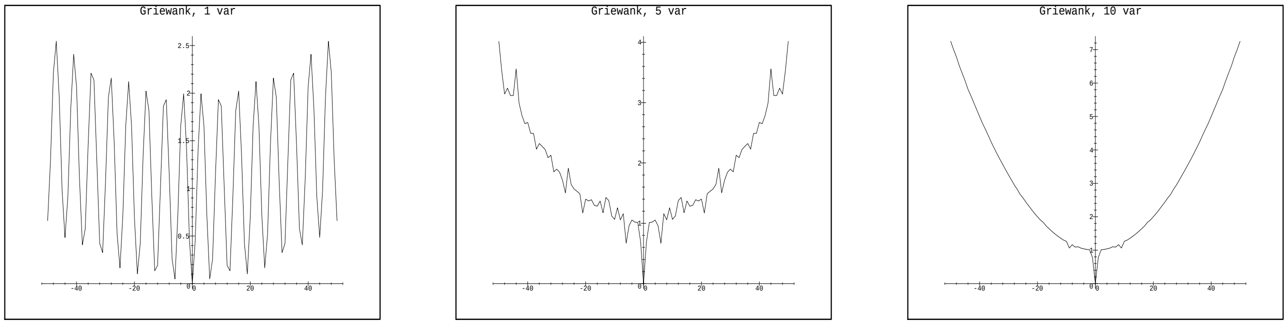
Davis (1991) has shown that many of the DeJong functions are quickly solved by RBC and that the performance of the RBC is sensitive to problem representation. Functions F1, F3 and F5 are separable functions that are always solved by line search.[2] Additionally, Mühlenbien et. al., (1993) have used empirical evidence based on test functions F6, F7 and F8 to argue that the "Breeder Genetic Algorithm" scales such that $O(n\ ln(n))$ function evaluations are needed to locate the global optimum, where n is the number of parameters used by these functions. However, line search requires only $O(n)$ function evaluations to solve F6 and F7 since these functions are also separable.

This leaves only F2, F8, F9 and F10 among the problems that are both nonseparable and nonlinear. Problems F2, F9 and F10 are not solved by a single pass of line search and multiple passes do not yield competitive solutions as illustrated in Table 2. All of these problems are solved by the CHC algorithm using dramatically fewer evaluations (Eshelman, 1991; Mathias and Whitley 1994).

[2] F4 is also separable, although the addition of noise might prevent an algorithm from locating an optimum.

Function	Mean Soln	Mean Sigma	No. Solved
f2	0.2647818	0.3033391	0
f9	0.1215778	0.0773996	0
f10	3.807974	1.777144	0

Table 2: Line Search Performance on Nonlinear Nonseparable Functions.



(1-Dimension) (5-Dimensions) (10-Dimensions)

Figure 1: The graphs represent slices of the Griewank's function for 1, 5 and 10 dimensional versions of this problem. These graphs clearly illustrate that as the dimensionality increases the local optima induced by the cosine decrease in number and complexity.

Figure 1 illustrates the Griewank function (F8) for 1, 5 and 10 variables. These figures are 1-dimensional slices of the function taken along the diagonal of the hypercube. The function becomes simpler and smoother in numeric space, and thus easier to solve, as the dimensionality of the search space is increased. The function becomes easier for Gray coding representations as well.

Of all of these common test problems, F8 is the only scalable, nonlinear and nonseparable function. Nevertheless, F8 exhibits a serious flaw when scaled. The summation term of F8 induces a parabolic shape while the cosine function in the product term creates "waves" over the parabolic surface creating local optima. As the dimensionality of the search space is increased the contribution of the product term is reduced and the local optima become smaller.

Since Gray coding preserves the adjacency contained in numeric space (Mathias and Whitley, 1994) the following observation is obviously true for all functions. *Any path that walks the adjacency neighborhood that corresponds to the discretized numeric representation of the search space exists as a subset of the paths that traverse the Gray space representation.* In a sense, the Gray space contains the discretized numeric representation. Thus, if the numeric representation offers opportunities for hill-climbing, these opportunities are preserved under Gray coding. This is not true for the BCD representation. This may explain why Gray coding often offers a representation of the search space more conducive to search than BCD, *especially on simpler test problems.*

These observations lead us to believe that the use of these test functions (i.e., F1 - F10) in comparative studies may lead to suspect conclusions. Therefore, we introduce new methods for constructing more complex test functions that satisfy the guidelines set forth in Section 2.1. The issues that these new methods address specifically are representation invariance and dimensionality/evaluation cost scaling. All of the methods discussed here for building test functions employ a strategy whereby more complex functions are built from simpler primitive functions. We illustrate these methods using functions taken from the existing common test suites. Not all of these problems are good candidates for primitive functions, but the process of identifying what makes a good primitive is instructive.

Before looking at the construction methods proposed in this paper we should make note of Kauffman's (1989) N-K landscapes . These problems are expressed as bit strings of length N where each bit interacts with K other bits to determine its contribution to the fitness function. While these functions have several nice properties, they are different from the types of problems that have typically been used in parameter optimization and are not scalable in the sense of being able to scale a function with known properties to a higher dimensional space. It is possible, however, that principles used in the construction of N-K landscapes could be combined with the construction methods proposed in this paper.

3 Invariance under BCD and Gray

It is possible to construct a function that is invariant to BCD and binary reflected Gray code representations. The transformation between BCD and Gray space is performed using binary matrix multiplication. There exists an $n \times n$ matrix G_n that maps a string of length n from the BCD to the Gray representation. There also exists a matrix D_n that maps the Gray string back to its BCD representation. The G_4 and D_4 matrices are:

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

In higher dimensions the G_n matrix continues to have 1 bits along the diagonal and the upper minor diagonal, and D_n has 1 bits in the diagonal and the upper triangle. The result of $x^T G_n$ produces a Gray coding of x and $x^T D_n$ produces a DeGray coding of x, where x is an n-bit column vector. For example, if $x^T = |1001|$, then $(x^T G_4)^T = |1101|$. Similarly, if $x^T = |1101|$, then $(x^T D_4)^T = |1001|$.

There exists a set of equivalences for the G_n and D_n matrices such that:

$$\text{for } k = 1..(n-1), (G_n)^k = (D_n)^{(n-k)}$$
$$\text{and} \quad (G_n)^n = (D_n)^n = I_n$$

where I_n is the $n \times n$ identity matrix. For simplicity, $(D_n)^k = D_n^k$. For a 4-bit representation, $G_4^1 = D_4^3$, $G_4^2 = D_4^2$, $G_4^3 = D_4^1$ and $G_4^4 = D_4^4 = I_4$; also note that $G_4 \cdot D_4 = I_4$. Using these equivalences, it is possible to construct a function that results in coding *similar* evaluations for both BCD and Gray coded strings. We define functions to be *similar* when the corresponding objective functions are identical except that the set of parameters are rearranged by a simple shift operation. Using an evaluation function $F(x)$ which takes a single parameter x of length n, we can generate a new function S_n that is *similar* in Gray and BCD space. We define S_n to have t parameters and t terms where $t = 2^q = 2^{(\lceil(\log_2(n))\rceil)}$. The general form of S_n is:

$$\begin{aligned} S_n(p_1, .., p_t) &= F(p_1) + F(p_2^T D_n) + \\ &\quad ... + F(p_{(t-1)} D_n^{(t-2)}) + F(p_t D_n^{(t-1)}) \end{aligned}$$

For example, suppose n=4, S_4 is defined as:

$$\begin{aligned} S_4(w,x,y,z) &= F(w) + F(x^T D_4) + \\ &\quad F(y^T D_4^2) + F(z^T D_4^3) \end{aligned} \qquad (1)$$

If the same input strings are Gray coded, then

$$\begin{aligned} S_4(w,x,y,z) &= F(w^T G_4) + F((x^T G_4) D_4) + \\ &\quad F((y^T G_4) D_4^2) + F((z^T G_4) D_4^3) \\ &= S_4(x,y,z,w) \end{aligned}$$

The BCD coded parameters in equation 1 can be shifted once to the left to produce the similar Gray coded parameter set. This does not change the evaluation as long as S is insensitive to the order of the parameters. Most methods discussed in this paper are insensitive to parameter order.

We applied this technique to F1 and F2 of the DeJong test suite. F1 uses bit strings of length 10. Thus, S_{F1} takes sixteen 10-bit parameters where each parameter is applied to the x^2 subfunction. For F2, which takes two 12-bit parameters, the S_{F2} expansion involves 16 terms where each term consists of two 12-bit parameters:

$$\begin{aligned} S_{F2}(ps_1, ..., ps_{16}) &= F2(ps_{1_x}, ps_{1_y}) + \\ &\quad F2(ps_{2_x}^T D, ps_{2_y}^T D) + ... + \\ &\quad F2(ps_{16_x}^T D^{15}, ps_{16_y}^T D^{15}) \end{aligned}$$

where ps_{i_x} refers to the x variable of parameter set i and ps_{i_y} refers to the second y variable of parameter set i. These expanded functions are not only insensitive to Gray coding, but are also more difficult to solve. The overall difficulty of the problem depends directly on which mapping of the problem is most difficult.

Table 3 shows the results of executing RBC, ESGA-T, and CHC on coding similar expansions of DeJong's F1 and F2. All algorithms were allowed up to 2.5 million evaluations for F1 and up to 5 million evaluations for F2. All results are averaged over 30 independent experiments. As expected, none of the BCD results are significantly different from the Gray results. In comparisons between algorithms, CHC was able to locate the optimum solution to the F1 test function significantly more often than all of the other algorithms. Additionally, CHC found significantly better mean solutions for the F2 function than those found by the other algorithms tested.

4 Constructing Nonseparable Scalable Functions

Many of the common test problems discussed here are separable functions. One way to introduce nonlinear interactions and retain scalability is to use a nonlinear function of two variables, $F(x,y)$, as a starting function. The function can then be scaled to three variables, by constructing a new function $F'(x,y,z)$ where:

$$F'(x,y,z) = F(x,y) + F(y,z) + F(z,x).$$

We will refer to F' as an *expanded* function. The expanded function F' induces nonlinear interactions across multiple variables. Furthermore, this strategy can be extended to provide different degrees of nonlinearity. A function with $O(N)$ subterms or with $O(N^2)$ terms can be easily constructed. Consider the following matrices:

		F1			F2		
ALG	Coding	Mean Solution	σ	Nbr Slv	Mean Solution	σ	Nbr Slv
RBC	BCD	1.133e-4	4.342e-5	1	6.538e+0	1.403e+0	0
	Gray	1.200e-4	6.644e-5	4	6.827e+0	1.585e+0	0
ESGA-T	BCD	4.833e-4	1.464e-4	0	2.187e+0	0.938e+0	0
	Gray	5.233e-4	1.569e-4	0	2.410e+0	1.290e+0	0
CHC	BCD	6.667e-6	2.537e-5	28	1.030e-1	7.228e-2	0
	Gray	6.667e-6	2.537e-5	28	1.093e-1	6.213e-2	0

Table 3: Performance on F1 and F2 Coding Similar Functions. The ESGA-T used a population of 200, mutation rate 1/L, where L is the length of the string, and 0.9 probability of crossover.

	ESGA-T			CHC			RBC		
No. Var	Mean Soln	Standard deviation	No. Solved	Mean Soln	Standard deviation	No. Solved	Mean Soln	Standard deviation	No. Solved
Full Matrix Excluding Diagonal (Gray Code)									
6	25.737	10.268	4	24.748	11.256631	5	0.008454	0.004507	0
8	42.501	23.845	0	51.739	14.063457	1	0.017842	0.007974	1
10	83.152	22.601	0	83.155	22.591471	0	0.033502	0.008282	0
Minor Diagonal of Matrix (Gray Code)									
6	5.345	1.812	3	5.939	< 0.000001	0	0.000181	0.000282	21
8	6.071	3.407	7	7.391	2.009167	2	0.000188	0.000347	23
10	7.919	4.027	6	9.569	1.807304	1	0.000134	0.000349	26

Table 4: Matrix Complexity for the Expanded F2 Function (Gray).

$$\begin{array}{c|cccc} & q & x & y & z \\ \hline q & & qx & & \\ x & & & xy & \\ y & & & & yz \\ z & zq & & & \end{array} \qquad \begin{array}{c|cccc} & q & x & y & z \\ \hline q & & qx & qy & qz \\ x & xq & & xy & xz \\ y & yq & yx & & yz \\ z & zq & zx & zy & \end{array}$$

where the variables q, x, y, z are labeled along the edges and appear as variable pairs in the matrix. The *minor diagonal* scaling strategy (left matrix) picks argument pairs along the upper minor diagonal of the matrix (i.e., (q,x), (x,y) and (y,z)) and the pair of arguments from the lower left corner of the matrix (i.e., (z,q)). This results in an $O(N)$ evaluation cost; every variable interacts with two other variables and appears in both parameter positions. The *full matrix* scaling strategy (right matrix) excludes the pairs along the main diagonal. Note that the cost of executing the evaluation function also scales: the cost of the full matrix evaluation grows as a function of $O(N^2)$, while the minor diagonal scaling strategy has an evaluation cost of $O(N)$.

DeJong's F2 provides a convenient example of a function that has nonlinear interactions between two variables. After scaling up the *expanded* F2 function (E-F2) to 6, 8, and 10 variables we compared the performance of the two evolutionary algorithms using BCD and Gray problem encodings. Gray coding does not significantly improve the performance of the genetic algorithms on E-F2. However, it does significantly improve the mean solution found by the RBC. (See Table 4). There was no significant difference between the results produced by CHC and ESGA-T for both the full matrix and minor diagonal evaluation strategies. Statistical significance tests indicated that RBC performed best in every case for both the full and minor diagonal evaluation methods when performance was measured using the mean evaluation. The Gray coded version of E-F2 using the minor diagonal evaluation appears to get easier for RBC as the problem is scaled up to higher dimensions; the average error is reduced at higher dimensions and the number of times the global optimum is found is increased. F2 may offer an advantage to the RBC because there are

	ESGA-T			CHC			RBC		
No. Var	Mean Soln	Standard deviation	No. Solved	Mean Soln	Standard deviation	No. Solved	Mean Soln	Standard deviation	No. Solved
Full Matrix Excluding Diagonal–BCD									
6	0.832	1.579	0	0.243	0.070	0	0.0933	0.057	5
8	1.583	3.761	0	0.406	0.156	1	0.2300	0.121	2
10	1.261	1.846	0	0.691	0.214	0	0.5718	0.169	0
Full Matrix Excluding Diagonal–Gray									
6	5.512	15.680	18	0.000	0.000	30	11.235	8.550	0
8	15.233	36.616	14	0.000	0.000	30	32.906	21.885	0
10	10.848	18.169	7	0.000	0.000	30	52.404	42.726	0

Table 5: Matrix Complexity for the Expanded F10 Function.

large regions of the search space that represent solutions competitive with the global optimum. Thus, this problem appears to be better solved by hill-climbing than by the evolutionary algorithms tested here.

Table 5 presents the results generated for the *expanded* F10 function, E-F10. Gray coding makes it much easier for CHC to locate the optimal solution on this problem, while RBC performs significantly worse when Gray coding is used. Gray coding produces mixed results for ESGA-T; the mean solutions become generally worse, but ESGA-T finds the optimal solution significantly more often. Both evolutionary algorithms locate the optimal solution significantly more often than RBC. Furthermore, the performance of RBC is dramatically worse when Gray coding is used. Additional experiments on E-F10 at 20, 50 and 100 dimensions indicates that this appears to be a difficult problem for all of these algorithms.

4.1 Composite Functions

Another method which can be used to produce scalable, nonseparable problems is function composition. Using F, a nonlinear separable function (e.g., F6 or F7), defined as:

$$F(x,y,z) = S(x) + S(y) + S(z)$$

and a two variable transformation T, we can create a nonseparable function using function composition:[3]

$$F'(x,y,z) = S(T(x,y)) + S(T(y,z)) + S(T(z,x)).$$

In a simple case, T might average the pair of input variables. Alternatively, T might be a nonlinear transformation of the pair of variables onto a single numerical value (e.g., F2). This technique, when combined with the matrix expansion methods discussed previously, provides a method for scaling nonlinearity and can also result in complex fitness landscapes.

Several compositions were explored using the Griewank function (F8). The scaling problem and the role of the cosine term in F8, as discussed in Section 2.3, become critical considerations when composing F8 with other functions. Table 6 shows a baseline set of experiments for F8 using 6, 8 and 10 variables. The results show that when F8 is Gray coded, CHC is able to locate the optimal solution with a high degree of regularity; RBC solves F8 significantly more often at 10 dimensions than at 8, and again significantly more often at 8 dimensions than at 6. Our empirical experiments show this same trend holds for 20, 50 and 100 variables: F8 becomes easier to climb as the function is scaled up.

The simplest composition we tested treated F8 as a 1-dimensional function with $T(x,y) = (x+y)/2$. the inner transformation function. This problem is more resistant to hill-climbing by RBC at higher dimensions that F8. (Table 6). The following compositions of F8 and F2 were also tested.

$$F8F2-A: F8(F2(w,x)_1, F2(x,y)_2, \; \dots \; F2(z,w)_m)$$
$$F8F2-B: F8(F2(w,x)_1 + F2(x,y)_2 + \; \dots \; + F2(z,w)_m)$$
$$F8F2-C:$$
$$F8(F2(w,x))_1 + F8(F2(x,y)_2) + \; \dots \; + F8(F2(z,w)_m).$$

As expected, the F8F2-A composition was the easiest to solve because of the scaling problem in the original F8 function. RBC provided significantly better mean solutions than either of the evolutionary algorithms on this problem, yielding solutions of less than 0.00001 at 6, 8 and 10 dimensions. The F8F2-B and F8F2-C compositions proved to be more difficult than F8F2-A, but not as difficult as the composite problem in Table 6 where the pairs of input parameters are averaged.

[3]This method requires that the range of $T(x,y)$ maps onto the domain of $S(x)$.

	ESGA-T			CHC			RBC		
Nb Var	Mean Soln	Standard deviation	No. Solved	Mean Soln	Standard deviation	No. Solved	Mean Soln	Standard deviation	No. Solved
Scaling the Simple Griewank (F8) Function									
4	0.0090	0.0156	22	0.0019	0.0072	28	0.0234	0.0180	9
6	0.0253	0.0207	10	0.0000	0.0000	30	0.0309	0.0161	4
8	0.0449	0.0289	5	0.0022	0.0083	28	0.0204	0.0153	10
10	0.0858	0.0425	0	0.0019	0.0073	28	0.0138	0.0167	17
Full Matrix Complexity for F8($(x+y)/2$) + ... + F8($(z+x)/2$)									
4	0.19282	0.15618	11	0.00651	0.03566	29	0.23459	0.07399	2
6	0.58311	0.37555	6	0.21439	0.32360	20	1.04029	0.17267	0
8	1.53744	0.54526	0	0.88686	0.67784	8	2.64662	0.60897	0
10	3.03061	0.78302	0	2.00293	0.85400	0	5.43582	1.26885	0

Table 6: Comparisons on two variants of F8 using Gray coding.

5 Conclusions

The results presented in this paper highlight several weaknesses in the parameter optimization problems commonly used to compare the performance of evolutionary algorithms. Comparative studies aimed at evaluating the performance of evolutionary algorithms against other search methods should include test problems displaying characteristics that make evolutionary algorithms an appropriate choice of search method.

Test suites should contain problems that are resistant to hill-climbing, as well as problems that are nonlinear and nonseparable. The scalability of both dimensionality and evaluation cost should be a consideration in the construction of new functions. Finally, to reduce ambiguity, a canonical form for each function should be defined. The problem construction methods proposed here address the issues of invariance under Gray and BCD representations, as well as scalability. These guidelines and function construction methods should be useful in the development of more robust test functions and facilitate cross-study comparisons of evolutionary algorithms.

Acknowledgements

This work was supported in part by NSF grant IRI-9312748 and IRI-9503366.

References

[1] Ackley, D. (1987). *A Connectionist Machine for Genetic Hillclimbing.* Kluwer Academic Publishers.

[2] Booker, L. (1987). Improving Search in Genetic Algorithms. In Davis, L., editor, *Genetic Algorithms and Simulated Annealing*, chapter 5, pages 61–73. Morgan Kaufmann.

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

[4] DeJong, K. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Department of Computer and Communication Sciences Ann Arbor, Michigan.

[5] 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.

[6] Forrest, S. and Mitchell, M. (1993). Relative Building-Block Fitness and the Building Block Hypothesis. In Whitley, L. D., editor, *FOGA - 2*, pages 109–126. Morgan Kaufmann.

[7] Goldberg, D. (1990). A Note on Boltzmann Tournament Selection for Genetic Algorithms and Population-oriented Simulated Annealing. Technical Report Nb. 90003, Department of Engineering Mechanics, University of Alabama.

[8] Mühlenbein, H. and Schlierkamp-Voosen, D. (1993). Predictive Models for the Breeder Genetic Algorithm. *Journal of Evolutionary Computation*, 1(1):25–49.

[9] Kauffman, S. (1989). Adaptation on Rugged Fitness Landscapes. In Stein, D., editor, *Lectures in the Science of Complexity*, pages 527–618. Addison-Wesley.

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

[11] Schaffer, J. D., Caruana, R. A., Eshelman, L. J., and Das, R. (1989). A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization. In Schaffer, J. D., editor, *Proc. of the 3rd Int'l. Conf. on GAs*, pages 51–60. Morgan Kauffman.