ELSEVIER

Artificial Intelligence 85 (1996) 245-276

Artificial Intelligence

Evaluating evolutionary algorithms

Darrell Whitley [a,*], Soraya Rana [a], John Dzubera [a], Keith E. Mathias [b]

[a] *Computer Science Department, Colorado State University, Fort Collins, CO 80523, USA*

[b] *Philips Research Labs, Amherst, MA 01003, USA*

Received January 1995; revised October 1995

Abstract

Test functions are commonly used to evaluate the effectiveness of different search algorithms. However, the results of evaluation are as dependent on the test problems as they are on the algorithms that are the subject of comparison. Unfortunately, developing a test suite for evaluating competing search algorithms is difficult without clearly defined evaluation goals. In this paper we discuss some basic principles that can be used to develop test suites and we examine the role of test suites as they have been used to evaluate evolutionary search algorithms. Current test suites include functions that are easily solved by simple search methods such as greedy hill-climbers. Some test functions also have undesirable characteristics that are exaggerated as the dimensionality of the search space is increased. New methods are examined for constructing 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

Numerous empirical studies have attempted to show the effectiveness of some particular search algorithm. Empirical and experimental approaches to comparing algorithms have many disadvantages, especially when the algorithms are designed to be robust, general purpose optimization and search tools. One obvious danger with empirically evaluating search algorithms is that the resulting conclusions depend as much on what problems are used for testing as they do on the algorithms that are being compared. This can have the side effect that algorithms are designed and tuned to perform well on a particular test suite; the resulting specialization may or may not translate into improved performance on other problems or applications. It is therefore important that test suites be both challenging and diverse.

* Corresponding author. E-mail: whitley@cs.colostate.edu.

SSDI 0004-3702(95)00124-7

In this paper we examine new and existing methodologies for constructing test functions for comparing the effectiveness of evolutionary algorithms on parameter optimization problems. The end result is not a new test suite, but rather principles for designing test functions to be used for different evaluation purposes. We propose guidelines concerning the types of problems that should be used for comparative studies of evolutionary algorithms, how these studies should be carried out and some of the limitations of such studies. Methods for constructing scalable test functions are also introduced. These methodologies and guidelines should make it possible to perform more critical comparisons in the future between different evolutionary algorithms as well as facilitate comparisons with other heuristic search methods.

First, we consider some of the limitations of current test suites, particularly as related to the evaluation of evolutionary algorithms for parameter optimization. We argue that problems should not be *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. Surprisingly, almost all of the functions in current evolutionary search test suites are separable. Such test problems have been used to demonstrate the effectiveness of algorithms such as simulated annealing over evolutionary search algorithms. This is problematic in that separable functions can be solved by exact methods. Such functions are also often readily solved by local search methods and hence may be easily solved by any algorithm that explicitly builds on local search, such as simulated annealing [25] or TABU search [17].

Test functions can also display symmetries which may make them easier to solve by some methods. For two-dimensional functions, symmetry exists if $F(x, y) = F(y, x)$. In higher dimensions, up to $N!$ equivalent solutions may exists for a function of N variables. We also show that higher order symmetries can exist which may make some types of genetic algorithms an inappropriate method of search.

Separable functions are commonly used as test problems because they are scalable. This allows search algorithms to be tested on problems with progressively higher dimensionality [32]. Scalability is indeed desirable, but the nonlinear interactions in a test function should also be sensitive to scaling. We show that simple methods can be used for constructing test functions that allow nonlinear interactions between variables to be selectively scaled as the dimensionality of the problem is increased. We also consider how scaling impacts the cost of evaluation.

The use of BCD (binary coded decimal) and binary reflective Gray encodings as discrete problem representations is another major consideration when applying evolutionary algorithms to parameter optimization problems with bit encodings. We explore the relationship between Gray and BCD representations, how they relate to real-valued representations and how these representations relate to search behavior.

We do not consider combinatorial optimization problems in this paper. Well-known test cases exist for problems such as the traveling salesman problem. The inherent difficulty of these problems and their status as NP-complete problems is more thoroughly documented than the difficulty of most parameter optimization problems [7]. Furthermore, researchers often use specialized representations and operators when applying

evolutionary algorithms and other heuristic search methods to this class of problems. Parameter optimization problems have simple representations (e.g. bit strings or real-valued strings) that are manipulated by a general set of operators. Given the specialized representations and operators, combinatorial optimization problems are not often used for general comparative purposes, perhaps because the results are seen as being application dependent. In the long term, such problems should make up a specialized component of a thorough test suite.

The design principles for parameter optimization problems proposed in this paper cannot solve the general problem of discriminating between search algorithms in terms of their effectiveness. However, the principles developed here will help to establish guidelines for comparative studies and focus the evaluation effort on classes of test problems that are most likely to be of relevance to basic evaluation goals.

2. Evaluating evolutionary algorithms

In recent years, the terms *evolutionary algorithms* and *evolutionary computation* have come to refer to a set of algorithms that use evolutionary principles to build adaptive systems. Genetic algorithms, as introduced by Holland in the 1970s [21], are perhaps best known. Around the same time in Germany, researchers such as Rechenberg [37] and Schwefel [41] were developing algorithms known as *evolution strategies*. Work in the 1960s by Fogel, Owens and Walsh [13] define yet another set of methods referred to as *evolutionary programming*.

Evolutionary algorithms are population-based search methods that employ some form of selection to bias the search toward good solutions. Mutation and recombination are applied to strings representing candidate solutions to some optimization or search problem. Genetic algorithms tend to emphasize recombination of string pairs, while evolutionary programming tends to emphasize a mutation driven search, where mutation acts on single strings. Evolution strategies place more emphasis on mutation than genetic algorithms, but do not exclude recombination to the same degree normally associated with evolutionary programming. Genetic algorithms as defined by Holland have also been associated with binary encodings and the notions of schema processing and hyperplane sampling, whereas real-valued encodings tend to be used in evolution strategies. Several publications provide detailed descriptions of these algorithms and their relationship to one another [2, 3, 14, 15, 20, 43].

2.1. The limitations of the existing test problems

For almost twenty years, De Jong's test suite [10] has continually been used as the standard for measuring the performance of various genetic algorithms. The De Jong test suite (Table 1, $F1$–$F5$) includes a variety of characteristics that may affect algorithmic performance. This test suite was never meant to serve as a "gold standard", but rather was designed to illustrate the broad efficacy of genetic algorithms for different basic types of parameter optimization problems [4]. These functions include a unimodal function ($F1$), a nonlinear function over two variables ($F2$), a discontinuous function ($F3$), a noisy function ($F4$) and a multi-modal function with several local optima ($F5$).

Table 1
Common test functions for evaluating evolutionary algorithms

$F1$:	$f(x_i \mid_{i=1,3}) = \sum_{i=1}^{3} x_i^2$	$x_i \in [-5.12, 5.11]$
$F2$:	$f(x_i \mid_{i=1,2}) = 100(x_1^2 - x_2)^2 + (1 - x_1)^2$	$x_i \in [-2.048, 2.047]$
$F3$:	$f(x_i \mid_{i=1,5}) = \sum_{i=1}^{5} \lfloor x_i \rfloor$	$x_i \in [-5.12, 5.11]$
$F4$:	$f(x_i \mid_{i=1,30}) = \left[\sum_{i=1}^{30} i x_i^4 \right] + Gauss(0,1)$	$x_i \in [-1.28, 1.27]$
$F5$:	$f(x_i \mid_{i=1,2}) = \left[0.002 + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6} \right]^{-1}$	$x_i \in [-65.536, 65.535]$
$F6$:	$f(x_i \mid_{i=1,N}) = (N * 10) + \left[\sum_{i=1}^{N} \left(x_i^2 - 10\cos(2\pi x_i) \right) \right]$	$x_i \in [-5.12, 5.11]$
$F7$:	$f(x_i \mid_{i=1,N}) = \sum_{i=1}^{N} -x_i \sin(\sqrt{\lvert x_i \rvert})$	$x_i \in [-512, 511]$
$F8$:	$f(x_i \mid_{i=1,N}) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} (\cos(x_i/\sqrt{i}))$	$x_i \in [-512, 511]$
$F9$:	$f(x_i \mid_{i=1,2}) = 0.5 + \frac{\sin^2 \sqrt{x_1^2 + x_2^2} - 0.5}{[1.0 + 0.001(x_1^2 + x_2^2)]^2}$	$x_i \in [-100, 100]$
$F10$:	$f(x_i \mid_{i=1,2}) = (x_1^2 + x_2^2)^{0.25} [\sin^2(50(x_1^2 + x_2^2)^{0.1}) + 1.0]$	$x_i \in [-100, 100]$

Other test sets have been introduced over the years [1, 8, 16, 32, 39]. Some of the best known of these problems are illustrated in Table 1. Functions $F6$–$F8$ are known as the Rastrigin ($F6$), Schwefel ($F7$) and Griewangk ($F8$) functions and can be scaled to any number of variables [32]. The functions labeled $F9$ and $F10$ are known as the sine envelope sine wave and the stretched V sine wave functions [39].

These test problems have often been used to tune and refine variants of a single evolutionary algorithm and to argue the superiority of one approach over another. The danger in this practice is that algorithms can become customized for a particular set of test problems; this is troubling if the test problems do not represent the types of problems that evolutionary algorithms are best suited for in practice.

Davis [9] has shown that many of these functions are quickly solved by a random bit climber. Davis has also shown that the performance of a random bit climber is sensitive to the representation of the problem. Additionally, Mühlenbein et al. [33]

Table 2
Results of one pass of line search on nonlinear nonseparable functions; the optimal solution for these problems is 0

Function	Mean solution	Mean sigma	Number solved
$F2$	0.2647818	0.3033391	0
$F9$	0.1215778	0.0773996	0
$F10$	3.807974	1.777144	0

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, we show that search methods exist that require only $O(n)$ function evaluations to *exactly* solve $F6$ and $F7$. In addition, we show that $F8$ becomes easier as the dimensionality of this function is increased.

One can immediately identify problems $F1$, $F3$, $F5$, $F6$ and $F7$ as separable functions. $F4$ is also separable, although the addition of noise might prevent an algorithm from locating the optimal solution. The *line search* algorithm exploits separability by solving for each parameter independently through enumeration. Given a separable function which accepts n variables that are coded using k bits, the total search space has a size of 2^{nk}. Line search checks each of the 2^k points that are associated with each of the n variables. Thus one complete iteration of line search has a cost of $n(2^k)$. Assuming k is constant, the result is an $O(n)$ exact method for solving discretized separable functions. For example, many of the test problems are encoded using 10 bits per parameter. For a 10 parameter problem, the effective size of the search space is not 2^{100}, but rather only $10(2^{10})$, which is easily enumerated. For general nonlinear functions, line search is not an exact method but rather serves as a heuristic form of local search which can be run multiple times with random restarts. The representation need not be binary; given any discretization of the variables, line search can be applied without regard to problem encoding.

This leaves only $F2$, $F8$, $F9$ and $F10$ as nonseparable, nonlinear problems. Of these problems, $F2$, $F9$ and $F10$ are not scalable; the results obtained after one pass of line search for these three problems are given in Table 2. All of these problems are also solved by various evolutionary algorithms using dramatically fewer evaluations [12,29].

It should be noted that line search is not an effective *heuristic* for $F2$, $F9$ and $F10$ in part because the number of values assigned to each parameter is large: $F9$ and $F10$ are coded using 22 bits per parameter and $F2$ is coded using 12 bits per parameter. All the other test functions are coded using 10 bits. Thus, a single iteration of line search requires more than 8 million evaluations for $F9$ and $F10$. Compare this to line search on a problem with 10 bits per parameter and 10 parameters (i.e., a search space of 2^{100}): line search can enumerate all 10 parameters 800 times given 8 million evaluations. If $F9$ and $F10$ are sampled at a rate of 2^{10} per parameter, they are also solved by line search using multiple iterations.

$F2$ is also known as Rosenbrock's function [38] in the optimization literature. Solutions to this function can be obtained using minimization methods that do not require derivatives and which employ linear search [6].

Of all the test problems in Table 1, only $F8$ (Griewangk's function) is scalable, nonlinear and nonseparable. Nevertheless we have found that $F8$ exhibits undesirable properties as the dimensionality of the function is increased. The summation term of the $F8$ function induces a parabolic shape while the cosine function in the product term creates "waves" over the parabolic surface; these waves create local optima. It has been shown by enumeration of low-dimensional versions of this function that the basin of attraction containing the global optimum appears to encompass a larger percentage of the total space as the search space grows [29].

We now note that the product term involving the cosine is such that as the dimensionality of the search space is increased the contribution of the product term becomes smaller and the local optima induced by the cosine term become smaller. This suggests that this function becomes easier as the dimensionality of the search space is increased for numeric real-valued representations. Since Gray coding preserves the adjacency contained in numeric space [29] *any path that walks the adjacency neighborhood that corresponds to the discretized numeric representation of the search space also exists as a subset of the paths that traverse the Gray space representation.* (See Section 3.2.) Unlike the BCD representation, the Gray space contains the discretized numeric representation. Therefore, we can conclude that Gray coded representations of this function also become easier as the dimensionality of the search space is increased.

Fig. 1 illustrates Griewangk's function for 1, 3, 5 and 10 variables. These figures are one-dimensional slices of the function taken along the diagonal of the hypercube. The effects of increasing the dimensionality of the problem with respect to the product term that includes the cosine are clearly illustrated. The function becomes simpler and smoother as the dimensionality of the search space is increased.

2.1.1. Symmetry

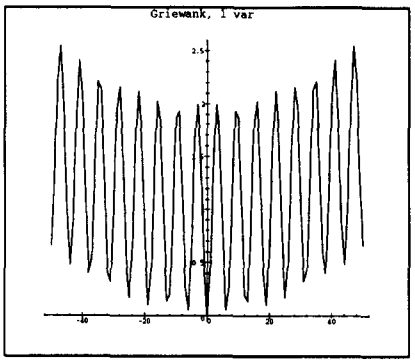
Another property that many of these functions exhibit is symmetry. Functions $F9$ and $F10$ are symmetric as can be seen by examining the evaluation functions (also see Fig. 2 for $F10$). Two-dimensional versions of the type of separable functions found in Table 1 are also symmetric. Separable functions can also display increased symmetry at higher dimensions.

Observation. Given a vector a representing the parameter values 1 to N of any potential solution to a separable function of the form $F(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} S(x_i)$ constructed using subfunction S, all $N!$ permutations of a represent equivalent solutions.

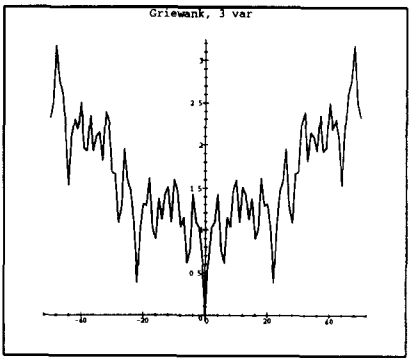
Proof. The evaluation of each component of a is independent of all other components of a, and so the order of evaluation is irrelevant. Since the same subfunction is applied to each component of a, the $N!$ permutations of a yield equivalent evaluations. □

A corollary of this observation is that if the components of a are unique (i.e., no two components of a are equal) then the $N!$ unique permutations are all distinct but equivalent solutions.

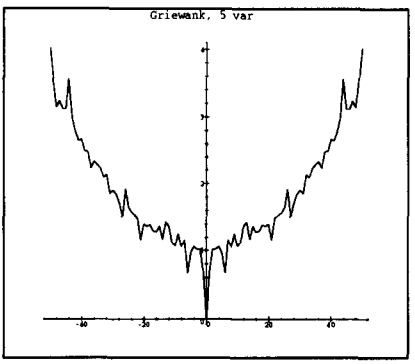
For the separable functions in Table 1, the global optimum is unique because at the global optimum each component of a has the same value and all permutations of a represent exactly the same solution. Thus, there is a single global optimum regardless of



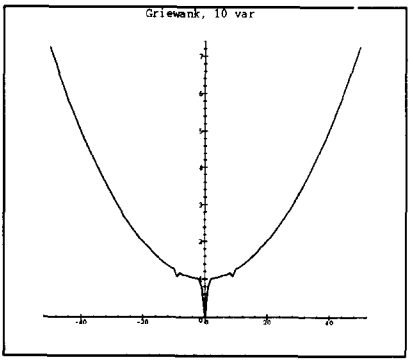
(1-Dimension)



(3-Dimensions)



(5-Dimensions)



(10-Dimensions)

Fig. 1. The graphs represent slices of the Griewangk's function for one-, three-, five- and ten-dimensional versions of this problem. As these graphs clearly illustrate, as the dimensionality increases the local optima induced by the cosine decrease in number and complexity.

the dimension of the function. Nevertheless, this symmetry partitions much of the search space into large equivalence classes. In general, if there are $\mathcal{Q}$ values per parameter and N parameters, ($\mathcal{Q} > N$), there are $\binom{\mathcal{Q}}{N}$ combinations where all parameter values are unique and there are $N!$ equivalent permutations for each of these combinations. The set of combinations containing a single duplicate is given by $\binom{\mathcal{Q}}{N-1}(N-1)$ of which $N!/2!$ combinations are equivalent.

Applications with multiple equivalent solutions are not unknown. Such problems are also not necessarily easy. Multiple equivalent solutions exist for some classes of neural networks. Consider a neural network with $\mathcal{H}$ hidden nodes, all of which are fully connected to an input layer and an output layer. Let the vector a represent the set of weights in the neural networks. Furthermore, let the weights of a be organized so that all weights that connect to any given hidden unit (i.e., all fan-in and fan-out connections) are adjacent on vector a. This partitions a into $\mathcal{H}$ pieces corresponding

to the hidden units. For every possible vector there are then $\mathcal{H}!$ reorderings that are equivalent solutions, since reordering the $\mathcal{H}$ partitions on a moves the positions of the hidden nodes in the network without changing the neural network's functionality. In this case a set of weights which minimizes error is also likely to have different weights for different connections in the networks, and thus there are potentially $O(\mathcal{H}!)$ multiple equivalent solutions.

The existence of multiple symmetric solutions induces a known mode of failure for certain forms of genetic algorithms. Assume that there are two symmetric solutions to a neural network optimization problem,

$$a_1, a_2, \ldots, a_{N-1}, a_N \quad \text{and} \quad b_1, b_2, \ldots, b_{N-1}, b_N,$$

where $a_1 = b_N, a_2 = b_{N-1}, \ldots, a_N = b_1$. Instead of a single parameter, assume component a_i represents a set of weights that attach the ith hidden node to the input and output layer of the neural network. Strings a and b represent a different ordering of the hidden units, but result in identical functionality. Recombining a and b, however, will mean that certain hidden nodes contained in both parents will be duplicated in the offspring, while other hidden nodes shared by both parents will be lost. If the parents represent good solutions, the offspring is likely to lose functionality. This problem has been noted by several researchers [31, 36, 40, 44]. Goldberg [18, p. 189] refers to offspring produced by dissimilar near-optimal parents as "lethals". The issues of symmetry and of lethals are significant for the new test problems introduced in Section 4.

For the separable functions in Table 1 two factors mitigate the negative effects of having $O(N!)$ symmetric equivalent solutions for an extremely large number of points in the search space. First, there is still a single global optimum. Second, each subfunction is independent from each other subfunction. Thus recombining a vector of parameters a and its inverse b poses no particular problem: if the individual components are good, the offspring is good. At the same time, because such problems have no nonlinear interaction across variables, recombination operators that preserve interacting subsets of variables in the form of "schemata" or "building blocks" [18] have no particular advantage and may be at a disadvantage since they less vigorously explore the search space.

2.2. *General guidelines for test suite problems*

Test suites should include problems which are representative of the types of applications for which the algorithm is appropriate. For example, it would be inappropriate to test heuristic search algorithms on a test suite made up of only linear functions, since other methods are generally more appropriate. Ideally, test suites should include problems which are representative of real world applications. However, given more powerful algorithms, the range of problems that are of practical interest is likely to expand. Thus, test suites should also include some problems that push the limits of the methods that are being tested. In addition, test suites should be open ended: testing should be hypothesis driven and different comparative goals may demand different test problems that may not be well served by a fixed test suite.

If evolutionary algorithms are to be of practical interest, it should be established that there exist functions where evolutionary algorithms outperform simpler methods.

In particular, if application problems can be solved by simple local search and line search methods then heuristic search methods such as evolutionary algorithms, simulated annealing and TABU search are unnecessarily complex and costly. A related goal would be to compare different types of evolutionary algorithms and heuristic search methods. Comparisons based on test problems solved by simpler methods might lead to different conclusions than comparisons based on more difficult problems. The following are guidelines which we propose for evaluating evolutionary algorithms.

1. Test suites should contain problems that are resistant to hill-climbing

When measuring the relative performance of evolutionary algorithms we would argue that the test suite used for comparison purposes should be composed largely of problems that are resistant to simple search strategies. To validate a test suite, all problems used for comparison purposes should be benchmarked using various hill-climbing strategies (including line search) for those representations that are to be used in the comparisons. [1] We are mainly interested in identifying problems that are readily *solved* by hill-climbing. This does not mean that all problems which are solved by hill-climbing should be automatically removed from a test suite. It is also important not to disallow problems that are difficult, but where certain hill-climbing methods may still yield competitive solutions. If problems are solved by hill-climbing, this should be well documented and comparative results should be interpreted accordingly.

When hill-climbing strategies are successful, they are typically faster than evolutionary algorithms and have less algorithmic overhead. Other forms of heuristic search which use strategies to escape local optima do so at additional computational cost. If evolutionary algorithms have advantages over hill-climbing algorithms and other stochastic search methods, these advantages may be lost if algorithm designers customize their evolutionary algorithms by adding mechanisms that promote hill-climbing.

It can be proven that for any given problem there are multiple problem representations that can be easily hill-climbed [26]; however, the space of all possible representations is dramatically larger than the search space. Therefore, if standard representations such as real-valued, BCD or Gray encodings are not hill-climbable, then finding a representation that is easily hill-climbed is likely to be far more difficult than solving the optimization problem. Here, we focus our attention on binary encoded problems using either BCD or Gray encodings; however, a number of the concepts and methods developed in this paper apply to real-valued encodings of problems as well.

2. Test suites should contain problems that are nonlinear, nonseparable and nonsymmetric

These issues have been shown to be relevant in light of the limitations of current test problems. Test suites should contain functions that have nonlinear interactions across variables and which are not easily solved by decomposing the problem and solving the individual parts. Similarly, not all functions should be symmetric. Having some

[1] Hill-climbing may be defined as any local search method that defines a neighborhood, then moves to the first position found in that neighborhood that offers improvement (e.g., next ascent) or to the position offering the best improvement in the neighborhood (e.g., steepest ascent).

problems with known symmetries is acceptable as long as comparative studies interpret data accordingly.

3. Test suites should contain scalable functions

Scalability is an important characteristic for test functions as it forms the basis for predicting the performance of algorithms as the search space becomes larger. This is often relevant to real world applications. Additionally, the difficulty of the problem should also scale up as the dimensionality of the problem increases and some mechanism should be provided for controlling the nonlinearity of the function.

If one considers the class of combinatorial optimization problems, it is clear that the scale of such problems is critical. Exact methods, such as branch and bound algorithms [23] exactly solve many NP-complete problems when these problems are relatively small. For example, Padberg and Rinaldi [34] have solved 500-city traveling salesman problems using exact methods; it is also relatively easy to solve knapsack problems with up to 250,000 variables [27]. It is only as these problems are scaled that the inherent difficulty of the problem is expressed.

4. Test suites should contain problems with scalable evaluation cost

For most common test functions, evaluation is extremely fast; thus the overhead of the search method is often a significant part of the total computation cost. The nature of these test functions stands in sharp contrast to some real world applications. For example, for some problems in seismic data interpretation, changing one parameter changes the partial evaluations associated with every other parameter and the cost of the full evaluation function grows as a function of $O(N^2)$, where N is the number of parameters [30]. This represents a significant computational challenge where the number of variables in large seismic data interpretations may be 600 variables. It is sometimes desirable for the cost of evaluation in test problems to increase as the size of the problem is scaled up.

On the other hand, for some objective functions evaluation can be relatively fast. Many NP-complete problems have simple objective functions, where the cost of evaluation scales in a linear fashion [7]. Therefore, the designer of a test suite should consider how the cost of evaluation scales with respect to the dimensionality of the search problem.

5. Test problems should have a canonical form

While the test functions in Table 1 have been widely used, under closer examination one is often likely to find that the *representations* used to solve the problems are actually different. First, the problems may be represented as binary or real-valued strings. Furthermore, even if two representations are both in BCD form, the method for translating binary strings into real-valued parameters may differ. In the simplest case, the degree of discretization may be different. But there are more insidious ways in which representations can differ. For example, it is common to transform the binary string into a positive integer and then shift and scale the integer value to map onto the range associated with a particular parameter. Alternatively, representations such as two's complement might also be used. Finally, even if parameters are translated into binary strings in the same fashion,

some researchers convert the BCD representations into a Gray coded representation. The practice of using different problem representations has created a great deal of confusion in the comparative literature. Such changes in representation have potentially dramatic impacts on search algorithms.

If one is trying to solve a particular problem, or even a particular class of problems, then changing the representation of the problem is reasonable and valuable. But for general comparative purposes, these differences change the difficulty of the problem. Different problem representations induce different numbers of local optima with different sized attraction basins. Solving a problem using two different representations equates to solving two different problems. Thus, results obtained using one representation of a problem for algorithm A may not be valid for comparing the performance of algorithm B if a different representation has been used.

The only solution to this representation problem is to be precise not only about how the problem is defined, but also about how it is represented. One way to adequately achieve this is to use an electronic archive and to design test problems so that representation is part of the problem specification.

2.3. Building new test problems

The methods discussed here for building test functions employ a strategy whereby more complex functions are built from simpler 2-D primitive functions. These 2-D functions have the advantage that they can be visualized and even enumerated. The construction methods also allow one to directly determine the global optimum in the higher order constructed functions.

3. Testing strategies and baseline comparisons

3.1. Selected algorithms

To illustrate our test suite construction strategies, we initially tested three forms of hill-climbers and two forms of evolutionary algorithms. The hill-climbers include a next ascent random bit climber (RBC) as defined by Davis [9], a random mutation hill-climber (RMHC) defined by Forrest and Mitchell [16] as well as the *line search* algorithm presented in Section 2.1. The first two search strategies are typically identified with binary encodings of search problems, but any of these search methods can be redefined in conjunction with any discretization of a parameter optimization problem.

Davis' random bit climber (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. When an improvement is found, it is accepted. After the climber has flipped every bit in the string, a new random sequence for testing the bits is chosen 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 optima has been located and RBC is restarted from a new random point in the space by generating a new random string.

Random mutation hill-climbing (RMHC) uses a "mutation" operator to make random changes to a single string. Every bit in the string is mutated with a low probability

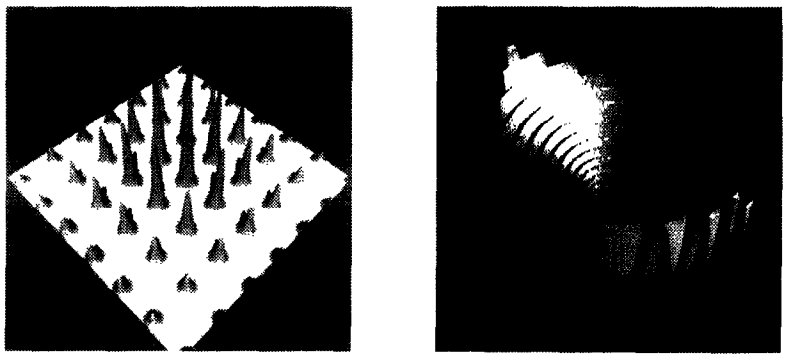


Fig. 2. Three-dimensional renderings of two functions displaying global structure. Both functions are viewed from the top looking down. The function on the right is $F10$.

(we used $2/L$ where L is the length of the string); any improvements are accepted. One motivation for testing RMHC was that this algorithm does not define a fixed neighborhood and thus can potentially reach every point in the search space; local optima are not encountered. However, we ran RMHC and RBC on dozens of functions at different dimensions. In every case the performance of RMHC was poorer than RBC using multiple restarts. Thus, we have elected to use only RBC for comparative illustrations.

Line search, on the other hand, often outperformed the other local search algorithms as well as the evolutionary algorithms. As long as the number of values assigned to each parameter is relatively small (e.g. 1024), it is practical to run line search multiple times. In the spirit of RBC, line search enumerates the individual parameters in a randomly determined order. If it enumerates all the parameter values twice and arrives at the same value, a local optima has been reached. Line search is then restarted from a new random point in the space with a new random ordering of the parameters.

While line search does encounter local minima, it is not sensitive to gradient information in the same way as is a simple gradient descent algorithm. Line search is not sensitive to local minima encountered along the line which is enumerated. It has the advantage of a greedy search while at the same time it can exploit global structure: line search has a distinct advantage when the best point in the dimension currently being searched remains in a relatively good region of the search space as the other parameters are also enumerated. For example, Fig. 2 illustrates two functions given as examples by Ackley [1] of functions with global structure. The first function,

$$F(x, y) = e^{-0.2\sqrt{x^2+y^2}+3(\cos 2x+\sin 2y)},$$

is a maximization problem that is solved by a single pass of line search. The best point along the first line will be at the center of the space which will take the second line through the global optimum. The second function, $F10$, is posed as a minimization problem and the global structure is slightly harder to exploit. If the initial cut through the space made by line search is near the outer edge of one of the concentric channels,

then the next cut through the space will pass through the global optimum. However, if the first cut does not lie near the outer edge of one of the concentric channels, then line search will become stuck without reaching the optimum. In this case, multiple iterations of line search may be required to find the global optimum. The practicality of running line search multiple times depends on the number of variables as well as the discretization of the variables. Compared to RBC, RMHC, CHC and ESGAT, line search is the least affected by representation, since it relies on enumeration of individual parameter values and thus is not affected by intraparameter neighborhood connectivity or how individual parameters are coded.

Along with the various local search methods, we utilized the CHC adaptive search algorithm [12], as well as an elitist simple genetic algorithm with tournament selection (ESGAT). The elitist simple genetic algorithm is meant to be representative of Holland's genetic algorithm. Elitist genetic algorithms date back to De Jong [10]. The use of tournament selection [19] is somewhat nonstandard, but it is well known that genetic algorithms that do not use some form of fitness scaling quickly lose selective pressure, thus making the genetic algorithm ineffective for optimization purposes [18]. Most fitness scaling methods also have the disadvantage that the scaling algorithm impacts the effectiveness of the search and are difficult to tune. Tournament selection is self scaling, simple to understand and implement, and effective.

Tournament selection is a stochastic form of rank-based selection. Instead of duplicating strings directly according to fitness, tournament selection randomly picks two strings and keeps the best of the two [19]. This is done $N-1$ times to create an intermediate population of $N-1$ strings. Recombination and mutation are then probabilistically applied to the $N-1$ strings to create the population of $N-1$ offspring. The algorithm is *elitist*, which means that the best string from the previous generation is then copied to the offspring population, restoring it to size N.

The population size for the elitist simple genetic algorithm was 200. Recombination was accomplished using a 2-point reduced surrogate crossover operator [5] applied with probability of 0.9. Mutation was applied to each individual bit with a probability of $1/L$, where L is the length of the string.

The CHC adaptive search algorithm [12] has many features in common with genetic algorithms, such as its strong emphasis on recombination. But it also has characteristics that would classify it as a $(\mu + \lambda)$ evolution strategy. CHC employs a parent population of size μ 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 λ offspring. The algorithm then chooses the μ best strings from the combined parent and offspring populations as the next generation of reproducing parents. Thus, the CHC algorithm maintains the best μ strings that have been encountered over the course of the search.

CHC is run here with $\mu = 50$, which is typical of most comparative work using this algorithm [12,28]. Although the target value for λ is also 50, the full set of offspring may not be produced due to threshold mating conditions which attempt to prevent "incest" (i.e., mating of similar strings). If two potential parents do not differ in more positions than specified by an adaptive threshold value they are not mated.

CHC also implements a form of "heterogeneous recombination" using HUX, a spe-

cial recombination operator. HUX exchanges half of the bits that differ between parents, where the bit positions to be exchanged are randomly determined. CHC uses only recombination to execute search; search terminates when no new offspring can be inserted into the new population. At this point, a restart mechanism known as *cataclysmic mutation* [12] is used to introduce new diversity to the search. The string representing the best solution found over the course of the search is used as a template to re-seed the population. Re-seeding of the population is accomplished by randomly changing 35% of the bits in the template string to form each of the other $\mu - 1$ new strings in the population. Search is then resumed.

3.2. Gray and BCD encodings

Before building and testing new functions, the issue of representation must be considered. The most commonly used representations among evolutionary algorithms that use bit representations are BCD (i.e. standard binary) and Gray encodings. The conversion from BCD to Gray can be performed using a simple conversion matrix. There exists an $n \times n$ matrix G_n that maps a string of length n from BCD to the binary reflective Gray representation. There also exists a matrix D_n that maps the binary reflective Gray string back to its original representation.[2] The G_4 and D_4 matrices (for use with a 4-bit string) are given below.

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

In higher dimensions the G_n matrix continues to have 1 bit along the diagonal and the upper minor diagonal, and D_n has 1 bit in the diagonal and the upper triangle. Using binary matrix multiplication (i.e. matrix multiplication mod 2), $x^{\mathrm{T}}G_n$ produces a Gray coding of x and $x^{\mathrm{T}}D_n$ reverses this process, where x is an n-bit column vector. For example, if $x^{\mathrm{T}} = |1001|$, then $(x^{\mathrm{T}}G_4)^{\mathrm{T}} = |1101|$. Similarly, if $x^{\mathrm{T}} = |1101|$, then $(x^{\mathrm{T}}D_4)^{\mathrm{T}} = |1001|$.

While the matrix shown here that converts BCD to binary reflective Gray coding represents the most common form of Gray encoding, all reorderings of the columns of the matrix produce a matrix that is also a Gray transformation. Furthermore, all rotations of any Gray representation produce a Gray representation. This produces a very large number of possible Gray codings; rotations of the space, however, do not change the structure of the space with respect to genetic algorithms or neighborhood search. The exact number of possible Gray representations is an open question.

Gray coding is often used in the genetic algorithm literature because it removes *Hamming cliffs*. A Hamming cliff corresponds to a pair of adjacent numbers in numeric

[2] Note that a transformation using the "DeGray" matrix does not necessarily produce a BCD encoding; if a string is "Grayed" and "DeGrayed" it returns the encoding to the original representation, whatever that might be. The DeGray matrix can also be used as a transformation in its own right.

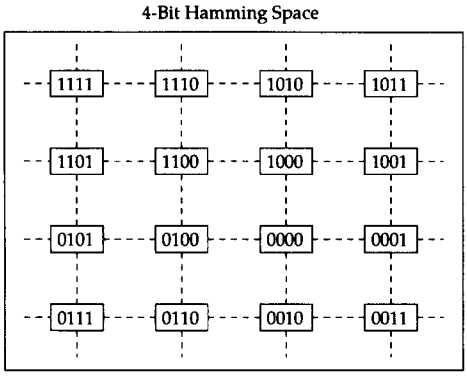


Fig. 3. Adjacency in 4-bit Hamming space.

space whose bit representations are complementary. Thus in a 4-bit space, 7 and 8 are adjacent in numeric space but their representations as bit strings are 0111 and 1000. But Gray coding does more than just remove Hamming cliffs. As noted earlier, Gray codings preserve the adjacency found in numeric representations of functions. Fig. 3 illustrates the adjacency between all 4-bit strings in Hamming space.

Note that the above space wraps around on all edges. The graph on the left in Fig. 4 shows that all the adjacency relationships found in the numeric representation are preserved in Gray space. On the right, one can see that only half of the adjacent edges found in numeric representations are preserved in BCD space; for representations of arbitrary length it continues to be true that half of the edges from the numeric representation are preserved under BCD representations.[3] The set of possible functions under Gray and BCD encodings is identical since there is an affine transform between the two representation spaces. It has been shown that there are many other ways in which Gray and BCD encodings are equivalent [29]. However, these representations are clearly different in terms of adjacency.

3.2.1. Invariance under Gray and BCD encodings

It is possible to construct functions that are insensitive to whether the representation is a BCD encoding or the binary reflective Gray code.

For select N, there exists a set of equivalences for the G_n and D_n matrices such that:

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

[3] The degree of adjacency for strings of length 2, 3 and 4 is half by inspection. Given an ordered set of strings of length $L-1$, to increase the ordered set of strings of length L in the BCD encoding, append a 0 to all the strings in the $L-1$ set to create the first half of the new set of strings and append 1 to the same ordered set of strings to create the second half. Since the degree of adjacency is half in both the first set of new strings and the second set of new strings, it also remains half in the newly created set. By induction, all BCD encodings preserve half of the adjacency relations contained in the numeric space.

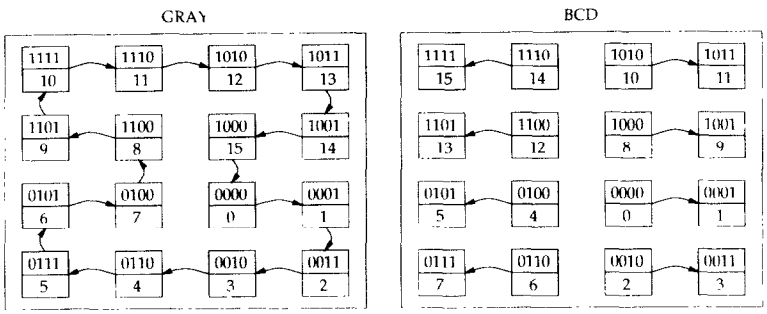


Fig. 4. Adjacency in 4-bit Hamming space for Gray and BCD encodings.

where I_n is the $n \times n$ identity matrix. For simplicity, $(D_n)^k = D_n^k$. In 4-bit space, $G_4^1 = D_4^3$, $G_4^2 = D_4^2$, $G_4^3 = D_4^1$ and $G_4^4 = D_4^4 = I_4$; also note that $G_4 \cdot D_4 = I_4$. Using these equivalences, it is possible to construct a function that results in the same evaluation for both BCD and Gray coded strings. Thus a BCD-Gray coding equivalent function $CE_4(x)$ exists for evaluation function $F(x)$, where x is a 4-bit vector in column form, such that

$$CE_4(x) = F(x) + F(x^{\mathrm{T}}D_4) + F(x^{\mathrm{T}}D_4^2) + F(x^{\mathrm{T}}D_4^3). \tag{1}$$

Now assume that the input to CE_4 is Gray coded:

$$CE_4(x^{\mathrm{T}}G_4) = F(x^{\mathrm{T}}G_4) + F((x^{\mathrm{T}}G_4)D_4) + F((x^{\mathrm{T}}G_4)D_4^2) + F((x^{\mathrm{T}}G_4)D_4^3).$$

Simplifying this expression yields,

$$CE_4(x^{\mathrm{T}}G_4) = F(x^{\mathrm{T}}D_4^3) + F(x) + F(x^{\mathrm{T}}D_4) + F(x^{\mathrm{T}}D_4^2).$$

For bit strings of length 5–8, the corresponding matrix D_n^4 is *not* an identity matrix; rather D_n^8 is the first occurrence of the identity matrix. In general for strings (or substrings) of length n where $2^{q-1} < n \leqslant 2^q$ the first corresponding D_n^k matrix that is an identity matrix is $D_n^{2^q}$ and $D_n^{(2^q-1)} = G_n$.

Whitley et al. [45] show how to use these principles to build functions that are insensitive to binary reflective Gray coding or BCD. At the same time, the resulting functions would not be insensitive to other forms of Gray coding or other transformations of the space. These coding insensitive problems also require that a function be solved simultaneously in multiple representations; it is unclear what relationship these functions would have to actual applications. Thus, we note the potential for constructing coding insensitive functions, but in the current study we use both BCD and Gray codings instead. We also argue that Gray coding should be the default given its relationship to the numeric representation.

4. Constructing nonseparable scalable functions

One way to introduce nonlinear interactions and still 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 N variables, for example, by constructing a new function $E\text{-}F(x_1, x_2, \ldots, x_N)$ where:

$$E\text{-}F(x_1, x_2, \ldots, x_N) = F(x_1, x_2) + F(x_2, x_3) + \cdots + E\text{-}F(x_{N-1}, x_N) + F(x_N, x_1).$$

We will refer to E-F as an *expanded* function. The expanded function E-F is no longer separable and induces nonlinear interactions across multiple variables. A function with $O(N)$ subterms or with $O(N^2)$ terms can be easily constructed. Consider the following matrices:

	x_1	x_2	x_3	x_4
x_1		x_1, x_2		
x_2			x_2, x_3	
x_3				x_3, x_4
x_4	$x_4 x_1$			

	x_1	x_2	x_3	x_4
x_1		x_1, x_2	x_1, x_3	x_1, x_4
x_2	x_2, x_1		x_2, x_3	x_2, x_4
x_3	x_3, x_1	x_3, x_2		x_3, x_4
x_4	x_4, x_1	x_4, x_2	x_4, x_3	

where the variables x_1, x_2, x_3, x_4 are labels along the left and top edges and appear as variable pairs in the matrix.

The *minor diagonal* scaling strategy (shown on the left) injects paired arguments into the subfunction F by choosing those variable pairs along the upper minor diagonal of the matrix (i.e. (x_1, x_2), (x_2, x_3), and (x_3, x_4)) and the pair of arguments from the lower left corner of the matrix (i.e., (x_4, x_1)). This results in $O(N)$ evaluations; every variable interacts with two other variables and appears in both parameter positions of F. We will also refer to this as a "wrap" scaling strategy. A similar use of paired variables appears in Powell's singular function [35]:

$$F(\boldsymbol{x}) = (x_1 + 10x_2)^2 + (x_2 - 2x_3)^4 + 5(x_3 - x_4)^2 + 10(x_1 - x_4)^4.$$

The expanded evaluation function can also use either the upper or lower triangle of the matrix to choose variable pairs, or the full matrix (including or excluding the main diagonal). Scaling the test functions using this pairing technique provides a method for scaling the cost of evaluation. Furthermore, expanded functions are similar to application problems such as the "statics" problem in seismic data interpretation mentioned earlier. In this case, the evaluation is summed over a matrix representing the cross-correlation terms associated with each pair of parameters [30]. The parameters represent time adjustments to seismic signals and maximizing the cross-correlations between signal pairs aligns the signals to reveal geological features. This problem appears to be similar to other visualization and signal processing problems in fields such as magnetic resonance imaging.

4.1. Properties of separable and expanded functions

If the primitive function F is *symmetric* along the diagonal of the search space then it may be easier to find values for expanded functions that simultaneously reduce error in both the x and y dimensions. Note that $E\text{-}F(x, y) = F(x, y) + F(y, x) = 2F(x, y)$ when

F is symmetric. This collapse of symmetric functions at higher dimensions occurs in other contexts as well. For example, the upper and lower triangles of the full evaluation matrix have identical pairs of variables, except that the order of the variables is reversed. If a function F is symmetric, then any expanded function E-F has identical evaluations for the expanded upper and lower matrix expansions. Thus, in general evaluation of a full matrix expansion (not including the diagonal) is equal to 2 times the evaluation of the lower or upper triangle matrix expansions. As will be seen, this has significant implications for search algorithms. This kind of simple symmetry can be avoided by using nonsymmetric primitive functions.

As with the separable functions in Section 2.1, another form of symmetry occurs as functions undergo expansion. Again consider

$$E\text{-}F(x,y) = F(x,y) + F(y,x).$$

Clearly, if the optimum of $F(x,y)$ is such that $x = y$, then the optimum of E-$F(x,y)$ has the property $x = y$. This pattern holds for higher-dimensional versions of E-F, thus making it possible to infer the global optimum of E-F from F. If the optimum of a symmetric function $F(x,y)$ is such that $x \neq y$, then there are two equal global optima for the 2-D E-$F(x,y)$ at $x = a, y = b$ and at $x = b, y = a$. This is due to symmetry in E-$F(x,y)$ as well as $F(x,y)$. Even if $F(x,y)$ is not symmetric, this duplication of global optima can potentially occur.

At higher dimensions symmetry problems become more extreme. Consider E-$F(x_1, x_2, x_3, x_4)$. If for $F(p_1, p_2)$, $p_1 \neq p_2$ at the global optimum of F, then it is possible that $(x_1 = a, x_2 = b, x_3 = c, x_4 = d)$ for E-F such that each parameter value at the global optimum is unique. In this case, all shifts of this sequence of parameter values are also distinct yet equivalent global optima for both the minor diagonal scaling strategy and the full matrix scaling strategy. For the full matrix scaling strategy, other equivalence classes also exist.

Theorem 1. *Given any vector a representing the parameter values 1 to N of any potential solution to an expanded function constructed with full matrix evaluation, all $N!$ permutations of a are equivalent solutions.*

Proof. For the expanded full matrix evaluation constructed using $F(x,y)$, each parameter value a_i appears at the x value in combination with all other components a_j. In addition, each a_i appears as the y value in combination with all other components a_j. Thus, the same set of $F(x,y)$ evaluations occurs for all $N!$ permutations of vector a. This holds regardless of whether the full matrix expansion include the diagonal or not, since the diagonal itself is symmetric. □

If the components of a are unique (i.e., no two components of a are equal) then the $N!$ unique permutations are all equivalent solutions. If a is a global optimum, then all unique permutations of a are equivalent global optima. Thus, there exists up to $N!$ global optima.

By placing the global optimum of the primitive function F on the diagonal such that at the global optimum $F(x,y)$ has $x = y$, the problem of multiple global optima can

be avoided and the optimum of E-F can be determined by enumerating the space for $F(x, y)$. But this does not change the fact that there are many equivalent solutions in general. The same counting arguments apply to expanded functions using the full matrix evaluation as applied to the separable functions in Section 2.1. In this case, if vectors a and b are parameter values, where b and a are inverse vectors, then recombining a and b may indeed result in a "lethals" problem. Unlike simple separable functions, there are nonlinear interactions between variables for expanded functions. Thus, this case is more analogous to the problem that can occur when recombining neural networks.

There are several ways to avoid the lethals problem associated with having numerous equivalent solutions. One solution is to weight the various calls to the subfunctions differently. Another way to deal with the problem is to use only the lower triangle of the matrix for subfunction evaluation. In addition, primitive functions should also be nonsymmetric.

One final property of expanded functions is that they allow for partial incremental evaluation. When a single variable is changed, it is possible to update only those subcomputations that are affected. For a matrix expansion with $O(N^2)$ subterms, only $O(N)$ subterms change; for a wrap expansion, only 2 subterms change. In the current study we did not distinguish between partial and full evaluations, but many applications allow for partial evaluation. The use of partial evaluation has been exploited for combinatorial optimization problems such as the TSP when comparing algorithms (e.g., [12, 32]) but have not been considered for parameter optimization problems.

4.1.1. Expanded functions and N-K landscapes

Expanded functions that use pairs of variables provide a limited degree of nonlinearity. Nonlinearity implies potential interactions over the power set of input variables. Functions could be built that more fully exploit interactions over the power set, or over some K combinations of variables. This idea appears in a related form in Kauffman's N-K landscapes [24]. These problems are expressed as bit strings of length N where each bit interacts with K other randomly chosen bits to determine its contribution to the fitness function. While these functions have several desirable 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 specific 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. One could build primitive functions of K variables and apply the N-K landscape expansion principles in an analogous fashion. At the same time, it might be difficult to characterize the types of functions which result from randomly selecting K variables. In this paper we limit our attention to expanded functions built from nonsymmetric primitive functions of two variables.

4.2. New primitive functions

The following two-dimensional functions were constructed to be nonsymmetric, to have many local minima and to have a unique global solution on the diagonal.

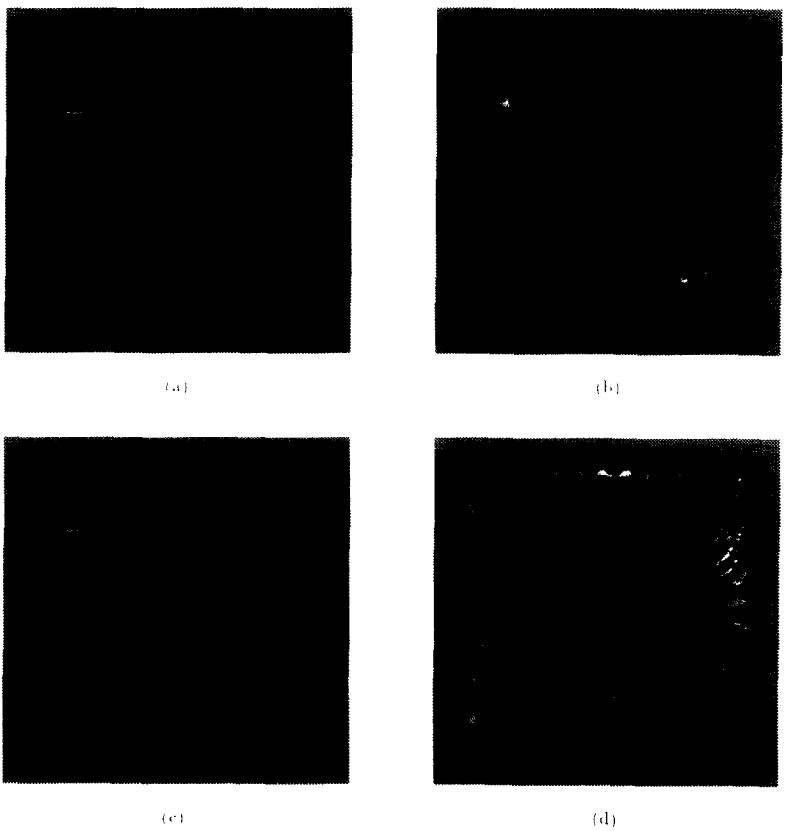


Fig. 5. Illustration (a) is a rendering of the primitive function $F102$, while (b) is E-$F102$. Image (c) represents a weighted version of E-$F102$. Image (d) is a weighted version of E-$F103$. All of these views are from below the functions.

$$F101(x,y) = -x\sin(\sqrt{|x-(y+47)|}) - (y+47)\sin(\sqrt{|y+47+x/2|}),$$

$$F102(x,y) = x\sin(\sqrt{|y+1-x|})\cos(\sqrt{|x+y+1|}) + (y+1)\cos(\sqrt{|y+1-x|})\sin(\sqrt{|x+y+1|}),$$

$$F103(x,y) = 0.5 + \frac{\sin^2\sqrt{100.0x^2+y^2x^2} - 0.5}{[1.0+0.001(x^2-2xy+y^2)]^2}.$$

$F101$ was constructed to be analogous to the structure of Schwefel's functions. Fig. 5(a) plots *Rana's* function: $F102(x,y)$. For $F101$ and $F102$, $\{x,y\} \in [-512,511]$ using 10 bits. It is instructive to examine the 2-D expansion for E-$F102(x,y) = F102(x,y) + F102(y,x)$ in Fig. 5(b). This simple 2-D expansion causes local minima to coalesce. To help to combat this problem we weighted the subfunctions for $F102$.

Fig. 5(c) plots $E\text{-}F102(x,y) = 3 \cdot F102(x,y) + F102(y,x)$ and illustrates how weighting can also ameliorate coalescence. Fig. 5(d) plots $E\text{-}F103(x,y) = 3 \cdot F103(x,y) + F103(y,x)$. $F103$ was constructed to be analogous to $F9$, the sine envelope sine wave. For $F103$, $\{x,y\} \in [-100,100]$ using 10 bits.

4.2.1. Composite functions

Many of the test functions that have been introduced into the literature have interesting properties, but may have limited usefulness in their current form. The proposed composition starts with a primitive function of one variable and composes it with an inner function that takes in two variables and outputs a single value which falls into the domain of the outer primitive function. A separable function such as Rastrigin ($F6$) or Schwefel ($F7$) might be expanded as follows:

$$F(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} S(x_i)$$

becomes

$$E\text{-}F(x_1, x_2, \ldots, x_n) = S(T(x_n, x_1)) + \sum_{i=1}^{n-1} S(T(x_i, x_{i+1})),$$

where S is the subfunction used in the original function and the transformation function T maps two variables onto the domain of S. If the resulting function is to have a single global optimum then T must uniquely map a set of input parameters to the value which yields the global optimum of S. For example, if an input of z yields the global optimum of S, then using a transformation function such as $T(x,y) = (x+y)/2$ may yield many solutions if for most inputs in the domain of x, there is a value in the domain of y such that $(x+y)/2 = z$.

We composed the Griewangk function ($F8$) with De Jong's $F2$ in the following manner (here illustrated with wrap expansion):

$$F8F2(x_1, x_2, x_3, \ldots, x_n) = F8(F2(x_1, x_2)) + F8(F2(x_2, x_3)) + \cdots + F8(F2(x_{n-1}, x_n)) + F8(F2(x_n, x_1)).$$

The resulting function is nonsymmetric (because $F2$ is nonsymmetric) with many local minima. The expanded functions also avoid the scale-up problems associated with the simple version of $F8$, since $F8$ is used in its one-dimensional form. Fig. 6(a) shows a three-dimensional view of the fitness landscape for the test function $F2(x,y)$. Fig. 6(b) shows a three-dimensional view of the fitness landscape for the function composition $F8(F2(x,y))$. By inspection the two fitness landscapes are similar, with the "horn of the saddle" in the $F2$ landscape being smoothed somewhat in the landscape of the function composition. Upon closer examination a great deal of texture exists over much of the new function. Clipping all fitness values above the value of 10.0 reveals a crescent shaped canyon illustrated in Fig. 6(c). Clipping all fitness values of the landscape above 1.0 highlights the bottom of the canyon (Fig. 6(d)) and reveals numerous local minima.

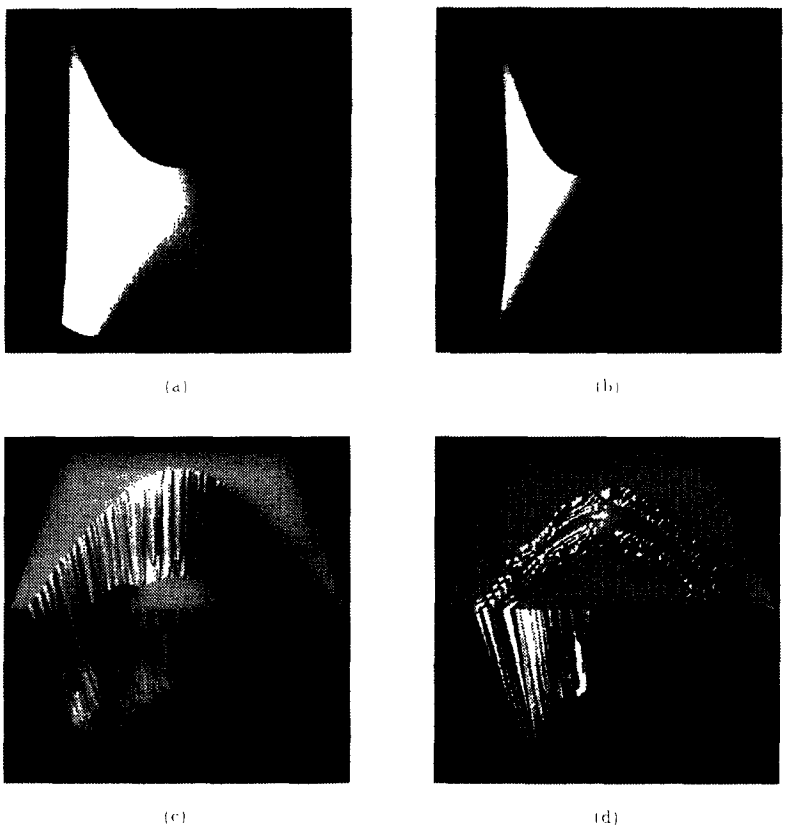


Fig. 6. Illustration (a) is a rendering of the original De Jong $F2$ test function view from the top looking down. Image (b) represents the same view of the simple function composition of the Griewangk and $F2$ test functions (i.e., $F8(F2(x,y))$). The $F8(F2(x,y))$ composition with all fitness values above 10.0 clipped is shown in Fig. (c), while all fitness values above 1.0 are clipped in illustration (d).

5. Some example tests

The comparisons offered in this paper are designed to illustrate characteristics of the test problems and their interactions with specific algorithms. The comparisons made here are not intended to definitively compare one algorithm against another, but rather to motivate hypotheses for future research. All of the test problems are posed as minimization problems. The best mean solutions are compared for various algorithms after 500,000 evaluations. These comparisons are affected by the amount of time that algorithms are allowed to search (e.g., the number of evaluations). This is particularly true for functions where it may be possible to quickly locate globally *competitive* solutions, but much more difficult to make subsequent progress or to locate the global optimum.

We begin by illustrating the effects of the scaling problem associated with the simple $F8$ function. Results are presented only for Gray coding. The results in Table 3 show

Table 3
Simple $F8$-Gray coded

Algrithm	Nb var	Griewangk's $F8$ Mean soln.	σ	Succ	Mean trials	Restarts
			CHC			
CHC	10	0.00		30	51015	
	20	0.00		30	50509	
	50	0.00104	0.00569	29	182943	
	100	0.0145	0.0231	20	242633	
			RBC			
RBC	10	0.00212	0.00808	28	184105	313.3
	20	0.00		30	82215	56.9
	50	0.00		30	33789	7.4
	100	0.00		30	34119	3.0
			ESGAT			
ESGAT	10	0.0515	0.0381	6	354422	
	20	0.0622	0.0400	5	405068	
	50	0.0990	0.0564	0		
	100	0.262	0.0459	0		
			Line search			
Line	10	0.0137	0.0174	18	175587	
	20	0.0286	0.0252	11	259231	
	50	0.0814	0.0640	8	266500	
	100	0.1899	0.1503	6	217983	

that CHC is able to locate the optimal solution with a high degree of regularity at 10 and 20 dimensions. At 50 and 100 dimensions the problem becomes easier for RBC. Note that while the dimensionality of the space is increasing, the number of restarts required to locate the global optimum is decreasing for RBC. Surprisingly, line search does not solve this problem as reliably as RBC or CHC. These results illustrate that not all search methods exploit the same features of the search space. It is possible that the additional work of parameter enumeration carried out by line search actually hinders the algorithm: RBC checks 10 neighbors for each parameter instead of the 2^{10} possible values per parameter, and thus exploits the simplicity of the problem structure. More complex algorithms may therefore be at a disadvantage on such problems.

Of the three new two-dimensional nonsymmetric primitive functions we developed, E-$F101$ proved to be the easiest. We tested this function for both the lower triangle and wrap expansions *without weighting*; both CHC and line search consistently solved the unweighted problem in all of its various forms, although CHC did so up to an order of magnitude faster. The weighted versions of E-$F101$ proved to be more difficult. Table 4 presents results for the various algorithms at 10 and 20 dimensions for E-$F101$ using a *weighted* lower triangle expansion as well as a weighted wrap. CHC clearly gives the best performance here, solving all variants of the problem every time. At 20 dimensions,

Table 4
Results for $F101$; this function was quickly and consistently solved by CHC when Gray coded; the optimum evaluation is −939.880

Algorithm	Nb var	Mean solution	σ	Succ	Mean evaluations	σ	Succ
		$F101$-LowerTriangle-GRAY			$F101$-Wrap-GRAY		
	10	−939.880	0.000	30	−939.879	0.0	30
CHC	20	−939.880	0.000	30	−939.879	0.0	30
	10	−924.753	39.284	24	−747.814	35.924	0
RBC	20	−806.934	68.243	2	−663.248	21.494	0
	10	−826.415	179.785	21	−816.147	96.457	10
ESGAT	20	−907.947	98.324	0	−816.497	78.416	0
		$F101$-LowerTriangle-BCD			$F101$-Wrap-BCD		
	10	−939.880	0.0	30	−939.879	0.0	30
CHC	20	−869.196	155.826	19	−939.862	0.048	24
	10	−939.643	10.346	0	−807.982	44.149	0
RBC	20	−840.329	45.019	0	−696.061	26.085	0
	10	−834.478	178.832	3	−809.973	74.127	0
ESGAT	20	−783.533	210.115	0	−820.326	70.475	0
		$F101$-LowerTriangle			$F101$-Wrap		
	10	−939.879	0.00383	29	−817.231	48.204	3
Line	20	−929.057	59.271	28	−767.276	45.821	1

ESGAT has lower mean solutions compared to RBC and line search and thus is at least competitive with these algorithms.

Table 5 shows results for Rana's function, E-$F102$, using a weighted lower triangle expansion as well as a weighted wrap expansion at 10 and 20 variables. An analysis of variance (ANOVA) comparing line search to CHC across the 4 variants of this problem also indicates that significant differences exist (F = 91.144). Line search typically generates the best results, although a one-tailed t-test indicated that CHC produced the best results at 10 dimensions for the wrap expansion.[4] For 20 dimensions a one-tailed t-test suggests that line search produced superior results compared to CHC for the wrap expansion of $F102$.

An ANOVA was used to test whether there were significant differences in performance based on a comparison of results for BCD and Gray encodings across the genetic algorithms and RBC; line search was not included since it is not sensitive to coding. The results were significantly different (F = 18.750). The results for E-$F102$ using Gray coding were better than the results using the BCD coding for all algorithms in every individual case except one: the results for ESGAT at 10 dimensions yielded very similar results for both BCD (−460.171) and Gray (−459.992).

Table 6 shows results for $F103$ using a weighted lower triangle expansion as well as a weighted wrap expansion to 10 and 20 variables. Of the three new primitive functions, $F103$ has the most local optima. Line search typically generated the best

[4] Results of the ANOVA and t-tests were considered significant if a p value less than 0.05 was generated.

Table 5
Rana's function at 10 and 20 variables using Gray and BCD encodings; the optimal evaluation is −511

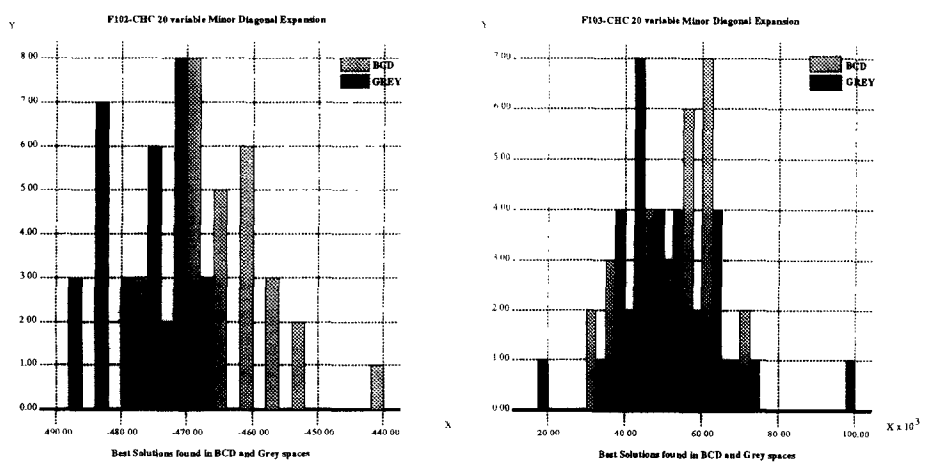
Algorithm	Nb var	Mean solution	σ	Succ	Mean solution	σ	Succ
		*F*102-LowerTriangle-Gray			*F*102-WRAP-Gray		
	10	−491.571	6.845	0	−494.927	4.392	0
CHC	20	−473.131	11.420	0	−477.594	6.812	0
	10	−403.569	15.912	0	−445.163	9.3987	0
RBC	20	−336.284	21.536	0	−404.399	8.7111	0
	10	−443.238	45.353	0	−459.992	17.787	0
ESGAT	20	−334.712	31.556	0	−407.920	13.553	0
		*F*102-LowerTriangle-BCD			*F*102-WRAP-BCD		
	10	−433.786	41.050	0	−481.951	8.638	0
CHC	20	−412.461	37.715	0	−463.813	7.715	0
	10	−395.076	14.5324	0	−438.556	11.321	0
RBC	20	−323.742	12.8602	0	−399.368	10.493	0
	10	−409.919	45.0622	0	−460.171	16.019	0
ESGAT	20	−300.282	22.1167	0	−402.922	15.082	0
		*F*102-LowerTriangle			*F*102-WRAP		
	10	−507.330	10.292	20	−490.517	7.25	0
Line	20	−487.277	36.662	13	−481.586	5.95	0

results, although CHC produced the best results at 10 dimensions for the wrap. A t-test indicated that line search produced the best results at 20 dimensions for the wrap expansion, but at 10 dimensions a t-test failed to show a significant difference between line search and CHC for the wrap expansion of *E-F*103. Gray coding made less of a difference in this case. Fig. 7 gives histograms for the mean solution distributions across problems *E-F*102 and *E-F*103 at 20 variables when using CHC on the wrap expansion. For *E-F*103 the Gray and BCD distributions are very similar while for *E-F*102 the differences between the distributions are more evident. An ANOVA failed to demonstrate a significant difference between results using BCD and Gray encodings for the genetic algorithms. One conjecture which deserves more exploration is that if a function has a numeric representation that is very complex then using a coding that is more closely related to the numeric representation could actually be a disadvantage.

*E-F*102, Rana's function, and *E-F*103 proved to be more difficult than *E-F*101. Line search did particularly well on lower triangle expanded functions. Note that the distributions of mean solutions are often skewed for the lower triangle results since the global optimum was often located more than half the time by both CHC and line search for functions such as *E-F*103. We conjecture that the coalescence of local minima may explain why some functions such as *E-F*102 and *E-F*103 appear to be more difficult for line search when using the minor diagonal "wrap" evaluation function as opposed to the lower triangle matrix evaluation. With fewer expansion terms coalescence may be less of a problem.

Table 6
Results for E-$F103$ using weighted lower triangle and weighted wrap

Algorithm	Nb var	Mean solution	σ	Succ	Mean solution	σ	Succ
		E-$F103$-LowerTriangle-Gray			E-$F103$-Wrap-Gray		
	10	0.00079	0.0027	17	0.0289	0.021	0
CHC	20	0.00408	0.0191	25	0.050	0.014	0
	10	0.14101	0.0295	0	0.0814	0.013	0
RBC	20	0.25323	0.4017	0	0.1255	0.010	0
	10	0.01941	0.0352	12	0.0571	0.021	0
ESGAT	20	0.15689	0.0228	0	0.1754	0.022	0
		E-$F103$-LowerTriangle-BCD			E-$F103$-Wrap-BCD		
	10	0.00149	0.0061	22	0.0289	0.018	1
CHC	20	0.00178	0.0079	12	0.0533	0.011	0
	10	0.07391	0.0240	0	0.0816	0.124	0
RBC	20	0.14471	0.0402	0	0.1306	0.008	0
	10	0.02433	0.0301	0	0.0560	0.021	0
ESGAT	20	0.16185	0.0299	0	0.1856	0.022	0
		E-$F103$-LowerTriangle-Gray			E-$F103$-Wrap-Gray		
	10	0.00005	0.0002	27	0.0227	0.011	0
Line	20	0.00023	0.0006	25	0.0297	0.015	0



(a) (b)

Fig. 7. Histograms for mean solutions of E-$F102$ and E-$F103$ at 20 variables plotted with respect to Gray and BCD encoding across algorithms.

Finally we look at results using the composite function $F8F2$. However, $E\text{-}F8F2$ was evaluated using a full unweighted matrix expansion. The full unweighted matrix expansion has the symmetry properties discussed in Section 4.1 and this may represent a difficulty to the algorithms that use recombination, particularly at higher dimensions as $N!$ grows extremely large. This is also more true for ESGAT than CHC because ESGAT uses the reduced surrogate operator which is designed to preserve blocks of bits representing hyperplane samples during recombination. CHC on the other hand typically "chops" the parents strings by randomly assorting individual bits from the two parents. Only Gray coded results are presented.

To further explore the hypothesis that symmetries makes this problem difficult for certain types of genetic algorithms at higher dimensions we ran all of the algorithms at 10, 20, 50 and 100 dimensions and included one additional algorithm: the Genitor algorithm. This is a steady state genetic algorithm, which like CHC, maintains the best strings in the population. Genitor selects pairs of parents for recombination and produces one offspring which immediately displaces the worst member of the population. Selection was done using an explicit linearly biased ranking of a population of size 1000. Genitor uses the same recombination operator as ESGAT. The reduced surrogate operator [5] is especially designed to promote unbiased hyperplane sampling and to preserve linkage between interacting parameters.

In other studies, Genitor's search behavior has more closely tracked CHC than ESGAT [28]. As can be seen in Table 7, Genitor's behavior on this problem is more similar to that of ESGAT. While these results are not conclusive, there is sufficient evidence to warrant further investigation of the hypothesis that this particular class of functions are not well suited to search by genetic algorithms which attempt to preserve linkage between blocks of interacting parameters during recombination.

Line search yields the best performance at 20 and 50 dimensions. CHC produced the best performance at 10 dimensions and a t-test indicates that CHC produced significantly better results at 100 dimensions. The poor performance of line search at 100 dimensions can be attributed in part to the fact that at 12 bits per parameter, one full iteration of line search requires 409,600 evaluations for 100 variables. Thus, little more than one full iteration of line search is possible given 500,000 evaluations.

6. Discussion

We do not want to overly interpret the current set of results from a comparative point of view. Our goal was to run the algorithms on problems with known properties to motivate the need for a better understanding of how problem characteristics can potentially interact with different search algorithms. We wish to promote the idea that specific types of functions should be used to test specific hypotheses about the behavior of search algorithms. Do nonseparable problems with multiple symmetric solutions induce a predictable mode of failure for evolutionary algorithms that use recombination? Do evolutionary algorithms that rely on mutation rather than recombination exhibit behaviors that are more correlated with hill-climbing algorithms such as RBC? Conjectures similar to these appear in the literature, but have not been adequately tested—in part,

Table 7
The $F8F2$ composition using an unweighted full matrix evaluation; this particular function has properties which may make it a poor test function for some genetic algorithms

Algorithm	Nb var	$F8F2$ full matrix GRAY		
		Mean solution	σ	Succ
CHC	10	1.344	0.921	0
	20	5.630	2.862	0
	50	75.0995	49.644	0
	100	670.223	377.59	0
RBC	10	0.139	0.422	0
	20	7.243	11.289	0
	50	301.561	72.745	0
	100	1655.557	605.268	0
ESGAT	10	4.077	2.742	0
	20	47.998	32.615	0
	50	527.100	176.988	0
	100	2991.89	596.470	0
Genitor	10	4.365	2.741	0
	20	21.452	19.459	0
	50	398.120	220.284	0
	100	2844.389	655.159	0
Line	10	3.0294	5.1526	0
	20	2.5025	3.2801	0
	50	2.6519	2.5583	0
	100	1606.4	1583.5	0

because test functions have not been designed that would allow one to perform adequate experiments.

The experiments presented here also further motivate the guidelines given in Section 2.1. It is critical to know whether a function is solved by simple hill-climbing. Scaling can make a significant difference: different problems display different interactions with algorithms at different dimensions. Problem representation can sometimes be critical.

Line search did surprisingly well in the current set of comparisons. However, in some cases (e.g. $F101$), CHC decisively outperformed line search. Wrap expanded functions using $F102$ and $F103$ were not actually *solved* by line search, but line search still sometimes found solutions with lower average means than the other algorithms. It should also be pointed out that comparing performance after 500,000 evaluations is somewhat arbitrary and different results might be obtained after 200,000 or 1 million evaluations. Brent [6] suggests another approach to comparing algorithms: measure the number of evaluations required to obtain a solution within $\pm\delta$ of the optimal solution. Such a metric could provide a different view of algorithm performance.

In the current study we intentionally gave line search a sporting chance by using only 10 bits per parameter when discretizing most of these functions. Using 15 bits per parameter would allow line search approximately 1.5 iterations over a set of 10 parameter values given 500,000 evaluations. Also, keeping the number of bits per parameter at 10 and increasing the number of variables to 100 or more has a similar effect. Thus line

search may not be appropriate if parameters must be represented at a fine level of discretization or for problems with hundreds of variables.

The relatively poor performance of ESGAT compared to CHC also demands discussion. Because ESGAT replaces the parent population after each generation (except for the best string) it does not display the greedy behavior of CHC. Many steady state genetic algorithms and $(\mu + \lambda)$ evolution strategies are similar to CHC in that they always maintain the best strings found so far in the population. Such algorithms tend to be better optimizers than simple genetic algorithms. On the other hand, part of Holland's original theoretical analysis for genetic algorithms was developed to show that genetic algorithms minimized loss of potential fitness payoff in terms of allocating trials to regions in the search space that on average contain good solutions. Loosely interpreted, the aggregate fitness values of *all* strings sampled each generation was considered as a performance metric rather than just the value of the best solution found at the end of the search. The relevance of this to real biological populations is obvious. In addition, real biological systems have fitness landscapes that dynamically change over time, so monotonically saving the best solutions can be a poor strategy in such situations. Some of the performance criteria discussed by Holland conflicts with a high risk search strategy that might examine many strings with poor evaluation in the hope of finding a string with a good evaluation. It also conflicts with enumerative methods and local search methods that look at all strings in a fixed neighborhood in order to find some improvement. There are also other ways in which simple genetic algorithms are more stable than other search methods: the trajectory of the *population* through a search space appears to be less affected by the composition of the initial population than algorithms such as CHC or Genitor (e.g. [42]). Thus Holland suggests in the preface to the 1992 edition of *Adaptation in Natural and Artificial Systems* that canonical genetic algorithms are suited to the study of complex adaptive systems whose "behavior is not well described by the trajectories around global optima". De Jong [11] has also recently pointed out that Holland's canonical genetic algorithms were not necessarily designed as function optimizers.

Theory and empirical studies have often been at odds in the genetic algorithms community. Theory lead to the development of recombination operators such as Booker's reduced surrogate operator, while empirical studies lead to the use of operators such as HUX and uniform crossover, which both randomly assort the individual bits found in the parent strings. However, in hindsight the existing empirical studies are suspect: if there are no nonlinear interactions between variables, then there is no penalty for chopping up the parents during recombination and there is no benefit to preserving building blocks of interacting parameters. Thus, better empirical studies are needed to better understand the relative merit of different approaches to genetic recombination.

Finally, we have said little about the canonical form of the new test functions. For example, bit strings were mapped onto a positive integer corresponding to the discretization of the function, then the integers were shifted onto the actual domain. The results are also not reproducible without the weighting factors used to weight the expanded functions. Because independent implementations can invite differences in the resulting functions, the test problems used in this paper can be obtained electronically by contacting the authors.

7. Conclusions

Several problems have been exposed with existing test functions and guidelines have been proposed for constructing parameter optimization test problems. The guidelines address many shortcomings of test suites currently being used in the evolutionary algorithms community.

We also introduced methods that should be useful in constructing more robust test functions for comparing evolutionary algorithms. Understanding the critical features of different test problems is difficult. The ability to visualize and analyze primitive functions proved to be helpful in the current study, but it is not sufficient for determining the complexity of the function. In order to provide a baseline for test functions, local search and hill-climbing algorithms should be included in any comparative study. There is also a body of literature on optimization without derivatives (e.g. [6]) that seems to have been largely ignored by the evolutionary algorithms communities and perhaps by other heuristic search communities.

We would also argue that the use of test suites should be hypothesis driven. Hooker [22] has also made similar arguments for more "scientific testing" and less "competitive testing" in the heuristic search community. Only after a testable hypothesis has been posed can researchers choose test problems in an informed fashion. It should also be possible to develop and test basic hypotheses about the relationship between test function characteristics and the computational behavior of local search and evolutionary algorithms when applied to constructed functions of higher dimensionality.

Finally, the results presented in this paper should alert researchers using common test suites for experimental evaluation on evolutionary algorithms, or for comparisons of evolutionary algorithms to other methods such as TABU search [17] and simulated annealing [25]. Evolutionary algorithms are best applied where simpler methods fail. If satisfactory solutions can be obtained for application problems using simpler optimization methods, then there may be no advantage in using evolutionary algorithms. It follows, therefore, that comparative studies aimed at evaluating the performance of evolutionary algorithms with that of other methods should include test problems that display characteristics that make evolutionary algorithms an appropriate choice of search method.

Acknowledgements

This research was supported by NSF grants IRI-9312748 and IRI-9503366 and by the Colorado Advanced Software Institute (CASI). CASI is sponsored in part by the Colorado Advanced Technology Institute for purposes of economic development. Thanks to Larry Eshelman for providing the original code for the sine envelope sine wave ($F9$) and stretched V sine wave ($F10$) test functions and for answering our questions about CHC. Thanks to Dave Houlton for providing the tools used to generate the fitness landscape ray-shade renderings used in this paper. Two other members of the GENITOR research group sat in on numerous discussions of this work and offered guidance from time to time: Larry Pycatt and Frederic Gruau. Adele Howe also offered

useful comments on this work. The reviewers also contributed useful suggestions for additional experiments that strengthened the paper.

References

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

[2] T. Bäck, F. Hoffmeister and H.P. Schwefel, A survey of evolution strategies, in: L. Booker and R. Belew, eds., *Proceedings Fourth International Conference on Genetic Algorithms* (Morgan Kauffman, Los Altos, CA, 1991) 2–9.

[3] T. Bäck and H.P. Schwefel, An overview of evolutionary algorithms for parameter optimization, *Evolut. Comput.* **1** (1993) 1–23.

[4] R. Belew, Paradigmatic over-fitting, *GA-Digest* **6** (18) (1992).

[5] L. Booker, Improving search in genetic algorithms, in: L. Davis, ed., *Genetic Algorithms and Simulated Annealing* (Morgan Kaufmann, Los Altos, CA, 1987) Chapter 5, 61–73.

[6] R. Brent, *Algorithms for Minimization without Derivatives* (Prentice-Hall, Englewood Cliffs, NJ, 1973).

[7] T. Cormen, C. Leiserson and R. Rivest, *Introduction to Algorithms* (MIT Press, Cambridge, MA, 1990).

[8] Y. Davidor, A naturally occurring niche & species phenomenon: the model and first results, in: L. Booker and R. Belew, eds., *Proceedings Fourth International Conference on Genetic Algorithms* (Morgan Kauffman, Los Altos, CA, 1991) 257–263.

[9] L. Davis, Bit-climbing, representational bias and test suite design, in: L. Booker and R. Belew, eds., *Proceedings Fourth International Conference on Genetic Algorithms* (Morgan Kauffman, Los Altos, CA, 1991) 18–23.

[10] K. De Jong, An analysis of the behavior of a class of genetic adaptive systems, Ph.D. Thesis, Department of Computer and Communication Sciences, University of Michigan, Ann Arbor, MI (1975).

[11] K. De Jong, Genetic algorithms are NOT function optimizers, in: D. Whitley, ed., *Foundations of Genetic Algorithms* **2** (Morgan Kaufmann, Los Altos, CA, 1993) 5–17.

[12] L. Eshelman, The CHC adaptive search algorithm. How to have safe search when engaging in nontraditional genetic recombination, in: G. Rawlins, ed., *Foundations of Genetic Algorithms* (Morgan Kaufmann, Los Altos, CA, 1991) 265–283.

[13] L.J. Fogel, A.J. Owens and M.J. Walsh, *Artificial Intelligence through Simulated Evolution* (Wiley, New York, 1966).

[14] D.B. Fogel, On the philosophical differences between evolutionary algorithms and genetic algorithms, in: D.B. Fogel and W. Atmar, eds., *Proceedings 2nd Annual Conference on Evolutionary Programming* (1993) 23–29.

[15] D.B. Fogel, Evolutionary programming: an introduction and some current directions, *Stat. Comput.* **4** (1994) 113–130.

[16] S. Forrest and M. Mitchell, Relative building-block fitness and the building block hypothesis, in: L.D. Whitley, ed., *Foundations of Genetic Algorithms* **2** (Morgan Kaufmann, Los Altos, CA, 1993) 109–126.

[17] F. Glover, Tabu search, Part I, *ORSA J. Comput.* **1** (1989) 190–206.

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

[19] D. Goldberg, A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing, Tech. Rept. 90003, Department of Engineering Mechanics, University of Alabama, Tuscaloosa, AL (1990).

[20] F. Hoffmeister and T. Bäck, Genetic algorithms and evolution strategies: similarities and differences, in: H.P. Schwefel and R. Männer, eds., *Parallel Problem Solving from Nature* (Springer, Berlin, 1991) 455–469.

[21] J. Holland, *Adaptation in Natural and Artificial Systems* (University of Michigan Press, Ann Arbor, MI, 1975; MIT Press, Cambridge, MA, 1992).

[22] J. Hooker, Testing heuristics: we have it all wrong, *J. Heuristics* **1** (1) (1995) 33–42.

[23] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms* (Computer Science Press, Rockville, MD, 1978).

[24] S. Kauffman, Adaptation on rugged fitness landscapes, in: D. Stein, ed., *Lectures in the Science of Complexity* (Addison-Wesley, Reading, MA, 1989) 527–618.
[25] S. Kirkpatrick, C. Gelatt and M. Vecchi, Optimization by simulated annealing, *Science* **220** (1983) 671–680.
[26] G. Liepins and M. Vose, Representation issues in genetic algorithms, *J. Exper. Theoret. Artif. Intell.* **2** (1990) 101–115.
[27] S. Martello and P. Toth, *Knapsack Problems: Algorithms and Computer Implementations* (Wiley, New York, 1990).
[28] K.E. Mathias and L.D. Whitley, Changing representations during search: a comparative study of delta coding, *J. Evolut. Comput.* **2** (3) (1994) 249–278.
[29] K.E. Mathias and L.D. Whitley, Transforming the search space with gray coding, in: J.D. Schaffer, ed., *Proceedings IEEE International Conference on Evolutionary Computation* (1994) 513–518.
[30] K.E. Mathias, L.D. Whitley, C. Stork and T. Kusuma, Staged hybrid genetic search for seismic data imaging, in: J.D. Schaffer, ed., *Proceedings IEEE International Conference on Evolutionary Computation* (1994) 356–361.
[31] D. Montana and L. Davis, Training feedforward neural networks using genetic algorithms, in: *Proceedings IJCAI-89*, Detroit, MI (1989) 762–767.
[32] H. Mühlenbein, Evolution in time and space: the parallel genetic algorithm, in: G. Rawlins, ed., *Foundations of Genetic Algorithms* (Morgan Kaufmann, Los Altos, CA, 1991) 316–337.
[33] H. Mühlenbein and D. Schlierkamp-Voosen, Predictive models for the Breeder genetic algorithm, *J. Evolut. Comput.* **1** (1993) 25–49.
[34] W. Padberg and G. Rinaldi, Optimization of a 532 city symmetric TSP, *Optim. Res. Lett.* **6** (1987) 1–7.
[35] M.J.D. Powell, An iterative method for finding stationary values of a function of several variables, *Comput. J.* **5** (1962) 147–151.
[36] N.J. Radcliffe, Genetic neural networks on MIMD computers, Ph.D. Thesis, University of Edinburgh, Edinburgh (1990).
[37] I. Rechenberg, *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution* (Frommann-Holzboof, Stuttgart, 1973).
[38] H.H. Rosenbrock, An automatic method for finding the greatest or least values of a function, *Comput. J.* **3** (1960) 175–184.
[39] J.D. Schaffer, R.A. Caruana, L.J. Eshelman and R. Das, A study of control parameters affecting online performance of genetic algorithms for function optimization, in: J.D. Schaffer, ed., *Proceedings Third International Conference on Genetic Algorithms* (Morgan Kauffman, Los Altos, CA, 1989) 51–60.
[40] J.D. Schaffer, D. Whitley and L.J. Eshelman, Combinations of genetic algorithms and neural networks: a survey of the state of the art, in: D. Whitley and J.D. Schaffer, eds., *Proceedings International Workshop on Combinations of Genetic Algorithms and Neural Networks* (IEEE Computer Society Press, Silver Spring, MD, 1992) 1–37.
[41] H.-P. Schwefel, *Numerical Optimization of Computer Models* (Wiley, New York, 1981).
[42] G. Syswerda, A study of reproduction in generational and steady state genetic algorithms, in: G. Rawlins, ed., *Foundations of Genetic Algorithms* (Morgan Kaufmann, Los Altos, CA, 1991) 94–101.
[43] D. Whitley, A genetic algorithm tutorial, *Stat. Comput.* **4** (1994) 65–85.
[44] D. Whitley, T. Starkweather and C. Bogart, Genetic algorithms and neural networks: optimizing connections and connectivity, *Parallel Comput.* **14** (1990) 347–361.
[45] D. Whitley, K. Mathias, R. Rana and J. Dzubera, Building better test functions, in: L.J. Eshelman, ed., *Proceedings Fifth International Conference on Genetic Algorithms* (Morgan Kauffman, Los Altos, CA, 1995).