Real-Coded Genetic Algorithms and Interval-Schemata

Larry J. Eshelman and J. David Schaffer
Philips Laboratories
North American Philips Corporation
345 Scarborough Road
Briarcliff Manor, New York 10510

Abstract

In this paper we introduce *interval-schemata* as a tool for analyzing real-coded genetic algorithms (GAs). We show how interval-schemata are analogous to Holland's symbol-schemata and provide a key to understanding the implicit parallelism of real-valued GAs. We also show how they support the intuition that real-coded GAs should have an advantage over binary coded GAs in exploiting local continuities in function optimization. On the basis of our analysis we predict some failure modes for real-coded GAs using several different crossover operators and present some experimental results that support these predictions. We also introduce a crossover operator for real-coded GAs that is able to avoid some of these failure modes.

1 INTRODUCTION

A growing number of researchers in the genetic algorithm (GA) community have come to champion real-coded (or floating-point) genes as opposed to binary-coded genes, in spite of the fact that there are theoretical arguments purporting to show that small alphabets should be more effective than large alphabets. Although a few theorists have taken on this argument (Antonisse, 1989; Wright, 1991), the standard defense has been the practical one that experience shows that real-coded genes work better (Davis, 1991a). In this paper we take on the task of giving a theoretical defense of real-coded GAs. In the past such GA-heretics have been a small

minority, who were largely ignored, but recently their numbers have been growing. Furthermore, in the last few years the GA community has begun to pay attention to the work of the Evolutionary Strategy approach in Europe which has from the beginning used real-coded genes (Bäck, Hoffmeister & Schwefel, 1991). Finally, descendents of the Evolutionary Programming approach have recently published a critique of GAs using binary coding and crossover, arguing that gaussian mutation on real-coded genes can be counted on to work well for a larger class of problems (Fogel & Atmar, 1990). (For more details on the history of real-coded GAs see Goldberg (1990).)

Although a number of advantages have been offered, we believe that three have been the primary motivation for real-coded GAs. First, real-coding of the genes eliminates the worry that there is adequate precision so that good values are representable in the search space. Whenever a parameter is binary coded, there is always the danger that one has not allowed enough precision to represent parameter values that produce the best solution values. Second, the range of a parameter does not have to be a power of two. Third, GAs operating on real-coded genes have the ability to exploit the gradualness of functions of continuous variables (where gradualness is taken to mean that small changes in the variables correspond to small changes in the function) (Wright, 1991).

In this paper we will concentrate on the third feature. Strictly speaking, the first point does not distinguish real-coded GAs from binary coded GAs. All computer solution methods require a discretization and hence a Nyquist-like assumption (i.e., no large variations between sample points). Given that computers have limited precision and "real-coded" (i.e., floating-point) values can be mapped onto integers, we will focus on what might be more properly called *integer-coded* GAs. (Several methods have been suggested for dealing with the precision problem without giving up a binary representation (Schraudolph & Belew, 1991; Shaefer, 1987).) We will not address the second alleged advantage, but will assume that the parameters of an integer-coded GA range over the same set of values as binary coded parameters—powers of two. In other words, it will be assumed that GAs that we will be comparing will use representations that have the same range and precision for any given function. The only difference is that the integer-coded GA will create new individuals by operating on strings of integers rather than bit strings. This approach allows us to use the same functions for comparing the performance of a GA that uses integer coding with a traditional GA that uses binary coding.

2 INTERVAL-SCHEMATA AND CROSSOVER

Most of the theoretical objections to real-coded GAs assume that the crossover operator operates at parameter boundaries (e.g., Goldberg, 1990, 1991). But many implementors of real-coded GAs use more vigorous forms of crossover. Davis combines parameter-bounded crossover with a crossover operator that averages (some of) the parameters (Davis, 1991a). Wright's linear crossover operator creates three offspring: Treating the two parents as two points, p_1 and p_2, one child is the midpoint of p_1 and p_2, and the other two lie on a line determined by p_1 and p_2: $1.5p_1 - 0.5p_2$ and $-0.5p_1 + 1.5p_2$ (Wright, 1991). Radcliffe's flat crossover chooses parameters for an offspring by uniformly picking parameter values between (inclusively) the two

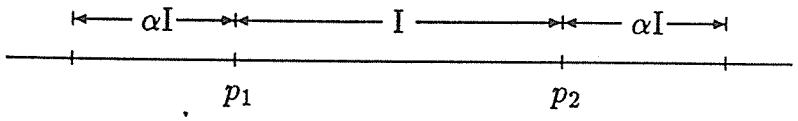

Figure 1: BLX-α

parents parameter values (Radcliffe, 1990). We use a crossover operator that is a generalization of Radcliffe's which we call blend crossover (*BLX-*α). It uniformly picks values that lie between two points that contain the two parents, but may extend equally on either side determined by a user specified GA-parameter α (see Figure 1). For example, BLX-0.5 picks parameter values from points that lie on an interval that extends $0.5I$ on either side of the interval I between the parents. (These are the extrema used by Wright.) BLX-0.0, on the other hand, is equivalent to Radcliffe's flat crossover. (Of course, there are many other possible versions of crossover that operate on integer or real-coded parameters.)

What all these crossover operators have in common is that they exploit the parameter intervals determined by the parents rather than the patterns of symbols they share. Holland's language of schemata was developed for strings of symbols. We feel that this terminology is too restrictive for analyzing real-coded GAs. Something analogous, yet distinct, is needed for GAs that manipulate interval values rather than bit values. We suggest that the relevant concept is an *interval-schema.*

Let $n = 2^L$ be the size of the range for integers that could be coded as L-bit strings. The number of interval-schemata (including all possible subranges) that can be defined over this range of integers is:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Thus, 36 interval-schemata can be defined for a parameter whose range is $[0, 7]$. There are two interval-schemata of length 7, $[0, 6]$ and $[1, 7]$. At the other extreme are the 8 short interval-schemata, $[0, 0]$, $[1, 1]$, etc. Every specific parameter value is a member of at least n interval-schemata (points that lie at the extrema) and up to a maximum of $\lfloor (n+1)^2/4 \rfloor$ interval-schemata (points that lie in the center of the interval). In particular, a value of k for the parameter that ranges from $[0, n-1]$ is a member of $(k+1)(n-k)$ interval-schemata. Furthermore, the two points at positions k_1 and k_2 $k_1 < k_2$ have $(k_1+1)(n-k_2)$ interval-schemata in common.

It should be noted that interval-schemata are similar to Wright's connected schemata (Wright, 1991). Both interval-schemata and connected schemata are concerned with intervals. However, connected schemata are like ordinary schemata except the "don't-cares" are restricted to the lower order bits. Thus, 01### corresponds to the interval $[8, 15]$ (assuming binary coding). The problem with this way of analyzing intervals is that some intervals are not representable with symbol-schemata—for example, no single symbol-schema describes the interval $[7, 10]$. Thus, according to Wright's analysis for a parameter of range $[0, n-1]$ there are $2^{L+1} - 1$ connected schemata, far fewer than the $n(n+1)/2$ interval-schemata (where $n = 2^L$) identified by our analysis.

Of course, if this is to be more than an abstract counting exercise it must be related to how the algorithm samples interval-schemata. Clearly any selection mechanism that allocates exponentially increasing trials to the observed best individuals will also allocate exponentially increasing trials to the observed best interval-schemata. So the important question is how crossover preserves and explores interval-schemata.

Parameter-bounded crossover, Davis's averaging operator, and Radcliffe's flat crossover (BLX-0.0) all have the property that the offspring are members of the same interval-schemata of which the parents are common members. (In Radcliffe's terms, these operators are "respectful.") These algorithms differ, however, as to how many new interval-schemata are potentially reachable in a single crossover event. Parameter-bounded crossover and averaging crossover are both strongly biased toward certain interval-schemata over others. BLX-0.0, on the other hand, is much less biased in this respect, although it does have a bias toward points near the center of the interval. It should be noted that this difference in reachability has a parallel for symbol-schemata using either two-point crossover (2X) or uniform crossover (UX). Because UX eliminates the positional bias of 2X, many more schemata are potentially reachable via a single crossover event using UX than 2X (Eshelman, 1991).

The negative side of greater reachability is that repeated applications of such a crossover operator are also more likely to disrupt the schemata that make the two parents better than average. As we have argued elsewhere, however, for a GA to be effective, it must not simply preserve schemata, it must test them in new contexts, and this entails disrupting schemata (Schaffer, Eshelman, & Offutt, 1991). Furthermore, we have shown that by combining a disruptive crossover operator with a conservative selection mechanism that maintains a population of the best M individuals seen so far (where M is the population size) one often gets the best of both worlds—vigorously testing schemata in new contexts while preserving the best schemata discovered so far.

The way an interval-processing GA (IPGA) processes interval-schemata is analogous to the way a symbol-processing GA (SPGA) processes symbol-schemata. To understand the parallel, it is important to note that long interval-schemata correspond roughly to low order symbol-schemata. Both are characterized by not being very specific. As search progresses a SPGA will progressively focus its search on higher order schemata whereas an IPGA will progressively focus on shorter interval-schemata. In the former case, the SPGA has narrowed the search down to certain partitions, whereas in the latter case the IPGA has narrowed the search to certain contiguous regions. The interval-schemata that are being searched are those bounded by the parameter extrema contained in the population. As these values narrow, the search becomes more and more focused, taking its samples from a smaller and smaller region of the parameter range. In this way an IPGA exploits the local continuities of the function.

Finally, it should be noted that the above analysis is easily extendible to multi-parameter functions. If the function has two parameters, for example, then each instance (point in a two dimensional space) is a member of a set of rectangle-schemata. The number of rectangle-schemata that a particular instance is a member of will be the product of the number of intervals the first parameter is a member of times the number of intervals the second parameter is a member of.

3 FAILURE MODES OF AN IPGA

Every successful search algorithm exploits some biases allowing it to favor some samples over others. Every bias also has an Achilles' heal—a problem can always be devised that will mislead a search method depending on a special bias. An IPGA is no exception. If the problem has no local (Euclidean) continuities, or if they lead away from the optimal solution, then an IPGA is likely to have difficulties. In this section we are concerned, however, with failure modes that arise even for problems that on the surface appear as natural for an IPGA as a SPGA. We shall concentrate on two failure modes: failure to propagate good schemata and premature convergence.

3.1 FAILURE TO PROPAGATE GOOD SCHEMATA

There are a number of situations where an IPGA will have difficulty propagating good schemata, but it is instructive to consider an extreme case—a needle on a plateau. In other words, there is only a single value in the interval that is good, and all other values are equally bad. Of course, finding the optimum will be equally hard for a SPGA as an IPGA. However, suppose that the function consists of a number of independent needle-on-a-plateau genes, or suppose the genes have some structure, but there is a plateau in the region of the optimum. The successful algorithm requires a crossover operator that has a fairly high likelihood of passing on to the offspring those genes that are by chance the optimum allele. Clearly, 2X has this property, since it has a relatively high probability in a many-gene problem of passing on any single gene intact. Perhaps less obvious, UX will also be more successful than BLX-0.0 at propagating optimal values when surrounded by a plateau. If we look at the extreme case where an optimum value is crossed over with its complement, the cases appear the same. Suppose the optimum lies at one of the extrema of a parameter that ranges over 2^L values—e.g., it is coded as the integer 0 for an IPGA and a string of L zeros for a SPGA with binary coding. Then if the parameter is crossed over with its complement (i.e., $2^L - 1$ in the case of the IPGA and L ones in the case of a SPGA) to produce a single child, there is only a $1/2^L$ probability of the allele surviving in the child. Although the probability of propagating the allele is the same in the worst case, this is not the typical situation. To see this, the important thing to note is that if there is no structure around the optimum, then the mate is likely to be a random individual in this range.[1] In the case of BLX-0.0 the expected value of a randomly generated gene will differ from the optimum by one half the range, and in the case of UX one half the bits. In other words, the probability of propagating the optimum when mated with a randomly chosen individual is $2 \times 1/2^L = 1/2^{L-1}$ for BLX-0.0, whereas it is $1/2^{L/2}$ for UX. If the optimum lies in the center rather than an extrema, the probability of propagating the optimum doubles for BLX to $1/2^{L-2}$, since the expected distance of a randomly chosen value from the optimum is now 1/4 rather than 1/2, but for intervals coded

[1] This needs to be qualified. Initially, when the optimum is first discovered, the values on the plateau will tend to be random. But as this optimum value is repeatedly crossed over with other values, to the extent that some offspring survive these matings, there is a tendency for the nonoptimum values to pick up "fragments" from the optimum allele and to drift in the "direction" of the optimum for both a SPGA and an IPGA.

with more than four bits, UX will still have a higher likelihood of propagating the optimum than BLX-0.0.

3.2 PREMATURE CONVERGENCE

In order for a GA to make progress it must focus its search. Convergence of the population is the necessary consequence of this. Unfortunately, it is highly likely that some of the observed correlations will be spurious (due to sampling error) so that the population may converge to a suboptimal region, discarding schemata that contain the optimum. No search algorithm that uses its prior samples to bias its future samples is immune to sampling error. As we have argued elsewhere, however, the less disruptive a crossover operator is, the more susceptible the algorithm will be to sampling error (Schaffer, Eshelman & Offutt, 1991). Furthermore, some functions are more likely to present misleading samples (e.g., deceptive functions) than others. The strength of BLX-0.0 is that it produces its samples in the contiguous regions defined by the points contained in the population. This means that BLX-0.0 is less likely to prematurely converge to the values that correspond to the lower order bits. (We will make an important qualification to this below.) 2X, on the other hand, is much more likely to prematurely converge on the lower order bits. 2X is good at preserving contiguous chunks of the chromosome intact, whereas BLX-0.0 is good at testing small variations of contiguous chunks (at least for the lower order bits). Unlike 2X, UX has no positional bias. It will be better at searching the lower order bits than 2X, but not as good as BLX-0.0.

BLX-0.0 pays a price, however, for this ability to exploit local information. To understand this, we need to look at what it means for the population to converge for an IPGA. Again, it is helpful to first examine the parallel phenomenon in a SPGA. In discussing schemata it is typically noted that a population of size M consisting of bit strings of length L contains between 2^L and $M * 2^L$ schemata. As a counting exercise this is true, but what is critical is how many new schemata potentially can be sampled in the population by crossover. If all the bits are converged, then the population consists of identical members, each of which has the same 2^L schemata. However, since there is nothing to cross over, no new schemata are being sampled. More generally, the number of schemata that are relevant to search is between 2^D and $M * 2^D$ where D is the number of bits of L that are not completely converged to the same alleles. The same holds for interval-schemata. The number of interval-schemata being searched by a GA using BLX is limited by the maximum and minimum values of the parameters represented in the population. Just as 2X or UX cannot introduce new alleles, BLX-0.0 cannot extend the interval ranges.

IPGAs that operate on large intervals share with SPGAs that use a large cardinality alphabet a particularly serious failure mode—premature convergence in the first generation. If the range of the parameters (or cardinality of the alphabet) is large relative to the population size, then the algorithm is quite likely to start its search without some values represented. This is a fatal weakness for an IPGA if the optimal point is at one of the extrema of the interval, for a crossover operator bounded by the two points determined by the parents (as in the case of BLX-0.0) will never be able to find the optimum unless it is enveloped by the original population. More generally, unless the extrema in the initial population envelop the optimal point, it cannot be reached via BLX-0.0 (or parameter-bounded or average crossover). This

fact is consistent with our analysis of interval-schemata. Of course, even if the initial population contains extrema that straddle the optimal point, this may be of no value if the points on one side of the optimal point are very poor so that they don't survive. (Imagine the optimum is at the bottom of an incline that abuts a cliff, so that it can be approached from only one side—any point sampled on the other side of the optimum won't survive.)

This problem of optimal-extrema can be overcome by letting the range from which an offspring is chosen extend on either side of the interval defined by the parents' parameter values (i.e., let $\alpha > 0$). We note that in the absence of selection pressure all values for $\alpha < 0.5$ will exhibit a tendency to population convergence toward allele values in the center of their ranges. Only when $\alpha = 0.5$ does the probability that an offspring will lie outside its parents become equal to the probability that it will lie between its parents. In other words, $\alpha = 0.5$ balances the convergent and divergent tendencies in the absence of selection pressure.

Although it is no longer the case that the child will be a member of the same interval-schemata that the parents share, the algorithm still narrows its focus as the search progresses, and thus differs from mutation.[2] But this process is no longer monotonic. Whenever a point is sampled that lies outside the population extrema and survives, the extrema are expanded. In the long run the extrema will narrow (assuming that there is local structure to the function), but in the process of narrowing and widening the algorithm now can also shift its focus laterally.

4 EMPIRICAL COMPARISONS

We have two goals in this section: (1) to test whether BLX-α is susceptible to the failure modes predicted in the previous section, and (2) to test how well a GA using BLX-α does on a range of functions in comparison to other crossover operators.

4.1 FAILURE MODE TESTS

We devised four functions to test BLX-α's predicted failure modes. Our purpose in choosing these functions was not to come up with challenging problems, but to choose a set of simple functions that will enable us to test the failure modes predicted in the previous section.

We tested each function using both a traditional GA and CHC (Eshelman, 1991). In order to place the emphasis on the effects of crossover, no mutation was used in either algorithm. CHC differs from the traditional GA in several respects: (1) *C*ross generational elitist selection: the parent and child populations are merged and the best M individuals are chosen, where M is the population size. (2) *H*eterogeneous recombination (incest prevention): only individuals who are sufficiently different (in terms of Hamming distance) are mated. (3) *C*ataclysmic mutation (restarts): only crossover is used to produce new offspring, but when the population converges, massive mutations are applied, preserving the best individual intact, and the search is resumed using only crossover. We tested four crossover operators: BLX-0.0, BLX-0.5, 2X, and HUX. (HUX is like UX, except exactly half the differing bits

[2] See section 5 for a discussion of BLX-α's relationship with mutation.

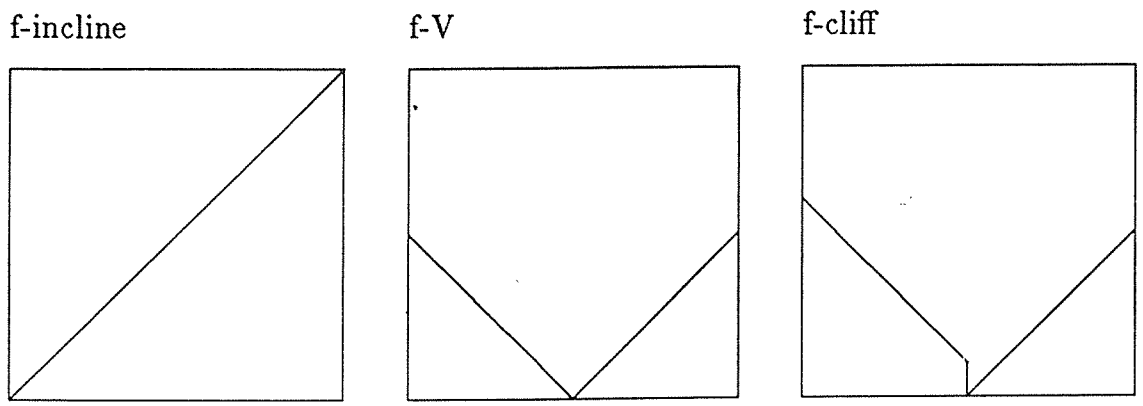

Figure 2: Failure mode test functions

Table 1: Failure Mode Tests

Number of times optimum found and trials to convergence								
	f-needles		f-incline		f-V		f-cliff	
	opt	trials	opt	trials	opt	trials	opt	trials
BLX-0.0	21	1597	0	1895	50	734	0	1926
BLX-0.5	20	1751	50	1418	50	968	49	1795
HUX	35	1409	47	1177	44	1153	47	1154
2X	29	831	9	1175	15	1175	8	1188
Tr-GA	11	1513	3	2111	0	1939	1	1563

are swapped at random.) Each of the four operators produces two children per mating. In each case, we used a population size of 50 and halted the search when either the minimum was found or the population converged (with no restarts). The traditional GA used proportional selection, the elitist strategy, a population size of 50, no mutation, and 2X with a crossover rate of 1.0.

The first function, f-needles, was devised to test the hypothesis that BLX-α would have difficulty in certain circumstances propagating good schemata. f-needles consists of five needles on five plateaus: for each of the five 6-bit genes a value of zero is given if the pattern is 111111 (i.e., 42 in gray code), and a value of one for all other patterns. The next three functions were devised to test the hypothesis concerning premature convergence (see Figure 2). The first, f-incline, is a simple incline problem with the minimum (the optimum) at one extreme: $f(x) = x$. The second, f-V, is a double incline or V function with the minimum at the center: $f(x) = |range/2 - x|$. The third, f-cliff, is similar to f-V except that the left incline has been raised so that there is a cliff on one side of the minimum: $f(x) = x - range/2$ $if(x \geq range/2)$ $otherwise$ $f(x) = (6 \times range/10) - x$. For these three functions, x ranges from 0 to $2^{30} - 1$. For the SPGA tests the 30 bit string was interpreted as gray coded.

The results, averaged for 50 replications, are reported in Table 1. Our prediction that needles-on-plateaus would be relatively harder for BLX-α than HUX or 2X is confirmed by BLX-0.0's and BLX-0.5's worse performance on f-needles. Also

as predicted, BLX-0.0 has difficulties on f-incline and f-cliff, but does quite well on f-V where the optimum lies in the center. The good performance of BLX-0.5 on these problems indicates that extending the interval outside the extrema determined by the parents overcomes a major shortcoming of BLX-0.0. Finally, the poor performance of 2X (for both a traditional GA and CHC) may come as somewhat of a surprise. It should be kept in mind, however, that no mutation was used with any of these runs, and a relatively small population size (50). The main failure mode for 2X on these functions is premature convergence—and in the case of the latter three functions, premature convergence on the lower order bits. Uniform crossover (HUX) is much less susceptible to this, and BLX-0.5 is even less so.

4.2 PERFORMANCE TESTS

We have seen that BLX-0.0 is susceptible to several failure modes as predicted, but that one of these shortcomings, the inability to sample parameter values that lie outside the extrema represented in the population can be overcome by using BLX-0.5. Our goal in this section is to discover how well BLX-0.5 performs relative to a crossover operator that operates on bit strings.

As our test suite we used a set of 11 functions for which we have extensive performance data (Schaffer, Caruana, Eshelman & Das, 1989; Eshelman & Schaffer, 1991; Eshelman, 1991). There are twelve problems in the test suite, but one of them (f10, a graph partitioning problem) is by nature a binary problem and so was not suitable for BLX-α. To these 11 we added two functions that other researchers have reported pose challenges to binary coded GAs—f13 and f14.

f13 is a function studied by Fogel and Atmar requiring the solving of a system of 10 linear equations with a 1.0 probability of the off-diagonal coefficient being non-zero (Fogel & Atmar, 1990). f14 is a 45 variable dynamic control problem studied by Janikow and Michalewicz (1991). For f13 we used 8-bit parameters and for f14, 10-bit parameters. (The binary representations of all functions were interpreted as gray coded.)

Table 2 summarizes some of the features of the thirteen functions. We used several variants of Davis's random bit-climber (i.e., an iterative, bit-wise hillclimber) as a measure of the difficulty of these functions (Davis, 1991b). Functions for which some version of the bit-climber significantly outperformed all versions of the GA were given an Easy rating, whereas functions for which some version of the GA significantly outperformed the bit-climber were given a Hard rating.[3] Functions for which the best version of the bit-climber and the GA performed about the same were given a Moderate rating.

We used CHC (with HUX as the crossover operator) as our benchmark since in two previous studies we showed that CHC outperformed a traditional GA for functions f1-f12 (Eshelman, 1991; Eshelman & Schaffer, 1991).[4] We ran CHC using BLX-0.5

[3]No version of the bit-climber was able to find the optimum for functions f8, f9, f11, f12 and f13. Furthermore, by shifting both axes by a small amount so that the optimum is no longer at the origin, functions f6 and f7 become hard for the bit-climber relative to the GA.

[4]For 10 of the 12 functions we were not able to find any parameter settings for a

Table 2: Function Summary

fnc	np	bpp	len	ep	bcr	description
f1	3	10	30	N	E	parabola
f2	2	12	24	Y	H	Rosenbrock's saddle
f3	5	10	50	N	E	stair steps
f4	30	8	240	N	H	quadratic with noise
f5	2	17	34	Y	E	Shekel's foxholes
f6	2	22	44	Y	M	sine envelope sine wave
f7	2	22	44	Y	M	stretched V sine wave
f8	16	4	64	Y	H	FIR filter
f9	30	5	150	Y	H	30-city TSP, sort representation
f11	20	5	100	N	H	needle on a plateau
f12	20	5	100	N	H	deceptive
f13	10	8	80	Y	H	10 linear equations
f14	45	10	450	Y	M	dynamic control problem

fnc function
np number of parameters
bpp bits per parameter
len string length
ep epistasis among parameters (Yes, No)
bcr bit-climber rating (Easy, Moderate, Hard)

and HUX on the 13 functions, halting each run when either the optimum was found or 50,000 evaluations had been completed. (Since f4 is noisy, it was required to be only "close" (two standard deviations) to the minimum.) Unlike the failure mode tests, restarts were enabled for these runs. The results are summarized in Table 3 based on 50 replications. BLX-0.5 did significantly better than HUX for f1, f2, f4, f13, and f14, whereas HUX did significantly better for functions f3, f5, f6, f11 and f12. For the remaining three functions there is no significant difference.

The five problems for which BLX-0.5 is the winner are the kind of functions that one might expect BLX-0.5 to do well on. They are all smooth, continuous functions (except for the discretization introduced by the representation). f1 and f4 are continuous and monotonic (in Euclidean space) with independent parameters. In the other three functions, f2, f13, and f14, there is interaction among the parameters. f2 is continuous and monotonic, but the discretization produces local minima in the region near the optimum. Based on the fact that a bit-climber can do quite well on f14, it seems that f14 is also monotonic. f13, on the other hand, seems to have many local minima. (We tried a variety of hillclimbers on f13, but none of them did very well.)

traditional GA (tr-GA) so that it outperformed CHC-HUX. The two exceptions were f1 (the easiest problem in the suite) and f12 (a deceptive problem). For seven of the functions, f5-f11, CHC performed significantly better. We tested the tr-GA using 840 different combinations of parameter settings for f1-f10, and 20 different combinations for f11 and f12, and picked the best settings for each function when comparing to CHC. CHC, on the other hand, was tested using its default parameter settings (population size of 50 and a restart mutation rate of 0.35).

Table 2: Function Summary

fnc	np	bpp	len	ep	bcr	description
f1	3	10	30	N	E	parabola
f2	2	12	24	Y	H	Rosenbrock's saddle
f3	5	10	50	N	E	stair steps
f4	30	8	240	N	H	quadratic with noise
f5	2	17	34	Y	E	Shekel's foxholes
f6	2	22	44	Y	M	sine envelope sine wave
f7	2	22	44	Y	M	stretched V sine wave
f8	16	4	64	Y	H	FIR filter
f9	30	5	150	Y	H	30-city TSP, sort representation
f11	20	5	100	N	H	needle on a plateau
f12	20	5	100	N	H	deceptive
f13	10	8	80	Y	H	10 linear equations
f14	45	10	450	Y	M	dynamic control problem

fnc function
np number of parameters
bpp bits per parameter
len string length
ep epistasis among parameters (Yes, No)
bcr bit-climber rating (Easy, Moderate, Hard)

and HUX on the 13 functions, halting each run when either the optimum was found or 50,000 evaluations had been completed. (Since f4 is noisy, it was required to be only "close" (two standard deviations) to the minimum.) Unlike the failure mode tests, restarts were enabled for these runs. The results are summarized in Table 3 based on 50 replications. BLX-0.5 did significantly better than HUX for f1, f2, f4, f13, and f14, whereas HUX did significantly better for functions f3, f5, f6, f11 and f12. For the remaining three functions there is no significant difference.

The five problems for which BLX-0.5 is the winner are the kind of functions that one might expect BLX-0.5 to do well on. They are all smooth, continuous functions (except for the discretization introduced by the representation). f1 and f4 are continuous and monotonic (in Euclidean space) with independent parameters. In the other three functions, f2, f13, and f14, there is interaction among the parameters. f2 is continuous and monotonic, but the discretization produces local minima in the region near the optimum. Based on the fact that a bit-climber can do quite well on f14, it seems that f14 is also monotonic. f13, on the other hand, seems to have many local minima. (We tried a variety of hillclimbers on f13, but none of them did very well.)

traditional GA (tr-GA) so that it outperformed CHC-HUX. The two exceptions were f1 (the easiest problem in the suite) and f12 (a deceptive problem). For seven of the functions, f5-f11, CHC performed significantly better. We tested the tr-GA using 840 different combinations of parameter settings for f1-f10, and 20 different combinations for f11 and f12, and picked the best settings for each function when comparing to CHC. CHC, on the other hand, was tested using its default parameter settings (population size of 50 and a restart mutation rate of 0.35).

Table 3: Performance tests

Mean number of trials to find optimum				
	BLX-0.5	sem	HUX	sem
f1	*874*	20	1089	25
f2	*4893*	357	9065	591
f3	2005	119	*1169*	27
f4	*933*	24	1948	97
f5	5561	588	*1396*	38
f6	14736	1998	*6496*	725
f7	3425	68	3634	291
f8	5822	522	7279	515
Mean performance				
f9	424.6	0.3	429.2	3.5
f11	1.5	0.2	*0.0*	0.0
f12	9.0	0.3	*1.2*	0.1
f13	*21.7*	2.4	61.8	11.2
f14	*16241.2*	30.1	38272.4	1039.0

The five cases for which BLX-0.5 lost are more instructive. BLX-0.5's poor performance on f11 and f12 is no surprise, since neither one has continuous variables with the sort of gradualness that BLX-0.5 can exploit. f11 consists of a 20 independent 5-bit genes, each of which is a needle on a plateau lying at one extrema. f12 consists of 20 independent 5-bit genes, each of which is deceptive. f3, like f11, also contains plateaus. Each of its 5 independent 10-bit genes has more structure than those of f11, but next to the optimum value there is a small plateau that gives BLX-0.5 some problems. (Each gene consists of 12 plateaus or steps, the optimum step being at one extrema, and only 10% as large as the previous step.) f5 consists of evenly spaced wells with sloped floors sunk in a plateau. Thus, the optimum value and the 24 suboptima are up against cliffs.

BLX-0.5's poor performance on f6 is harder to explain. It is a noiseless, continuous function without any plateaus or cliffs. On examination, it turns out that f6 illustrates not a defect in BLX-0.5 so much as a fortuitous advantage presented to HUX by the representation chosen. Since we believe that this is an important phenomenon, we will examine f6 in some detail.

Figure 3-a shows a 2D cross section through the origin of f6. f6 is cylindrically symmetric about the z axis. Figure 3-b shows a small region of f6 around the origin as seen from "above". The point in the center is the global optimum, and the concentric circles marked with dashed lines are the regions of the second, third, and fourth best local optima (counting from the center). The concentric circles marked with solid lines are the peaks of the ridges that separate the local optima.

Figure 3-c plots the points generated and evaluated during a single run of CHC using HUX and Figure 3-d plots the subset of points generated that are accepted into the parent population by replacing the worst members. Figures 3-e and 3-f show corresponding plots for BLX-0.5. Figures 3-c and 3-e dramatically illustrates the difference in how schemata are sampled via a SPGA and an IPGA—patterns

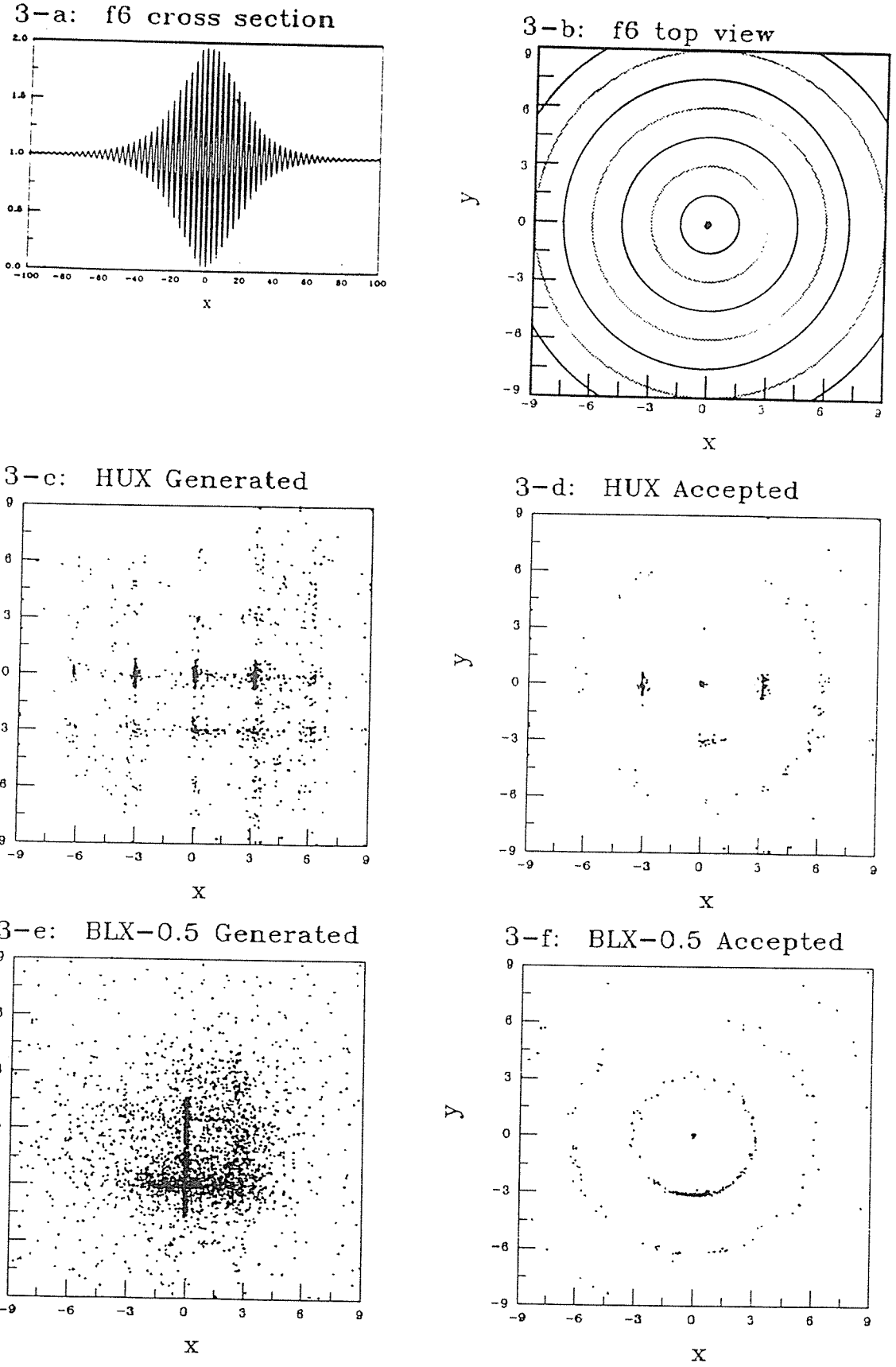

Figure 3: Function f6

vs intervals. One can see the outline of a grid-like structure filling much of Figure 3-c but not 3-e. (Keep in mind that the source (i.e., parents) of the points shown in Figures 3-c and 3-e are some of the points in figures 3-d and 3-f.)

To really understand the differences between the two search algorithms a dynamic dimension is needed. Both algorithms tend to get trapped in the best suboptimal region indicated by the inner, dashed circle in Figure 3-b. And both tend to favor points in this inner circle that intersect the x and y axes, although this tendency seems to be much stronger with HUX.[5] Given that good points tend to cluster in these areas, all crossover needs to do to put a point in the central region, where the global optimum resides, is to recombine a point in the (0, 3) region with a point in the (3, 0) region (for example). This would be easy to do with parameter-bounded crossover, but it turns out that in this case it isn't that hard with uniform crossover (HUX) either. The reason is that the function was fortuitously discretized so that the spacing between the concentric circles is nearly a power of 2. (The circle marking the best suboptima crosses the axes 65820 units from the center which is very close to $2^{16} = 65536$.) Since Gray coded values that are powers of two apart differ by only two bits (except when they are neighbors), the Gray coded values of the points where the inner suboptimum circle crosses either of the axes will differ by only two higher order bits from the optimum point in the center. However, some of the neighboring points in this suboptimal region will differ from the optimum by only one bit. By shifting both the axes by an amount that is not a multiple (or a near multiple) of the distance between concentric circles, e.g., 2^{14}, the points in the best suboptimal region will always differ from the optimum by at least two bits. This makes the problem somewhat harder for HUX. The grid-like pattern generated by HUX for the shifted problem is similar to that shown in figure 3-c except the spacing between "lines" is half as much as before (see Figure 4). BLX-0.5's performance is not affected, but HUX's performance deteriorates to that of BLX-0.5. We can make f6 even harder for HUX by rescaling the problem so that the distance between "valleys" is one-third of that of the original problem. This does not affect the performance of BLX-0.5, but HUX now does significantly worse than BLX-0.5.

In summary, the experimental data corresponds nicely with the theory. The functions for which BLX-0.5 does worse than HUX are either predicted by theory to be hard for BLX-0.5, or are cases where HUX has an advantage due to the periodicity (natural or fortuitous) of the problem.[6]

[5]The reason why points tend to cluster along the axes is that in these regions many new good points can be generated by changing one parameter value by a small amount whereas in regions that intersect lines at 45° to the axes good neighboring points can be generated only by changing both parameters by nearly equal and small amounts.

[6]L. Davis has discovered a similar phenomenon where a choice of representation (e.g., symmetrical about the origin) can make the problem easy for a bit-climber (Davis, 1991b). A SPGA, however, is much much more robust in this respect—there are many more ways to fortuitously introduce patterns which it can exploit.

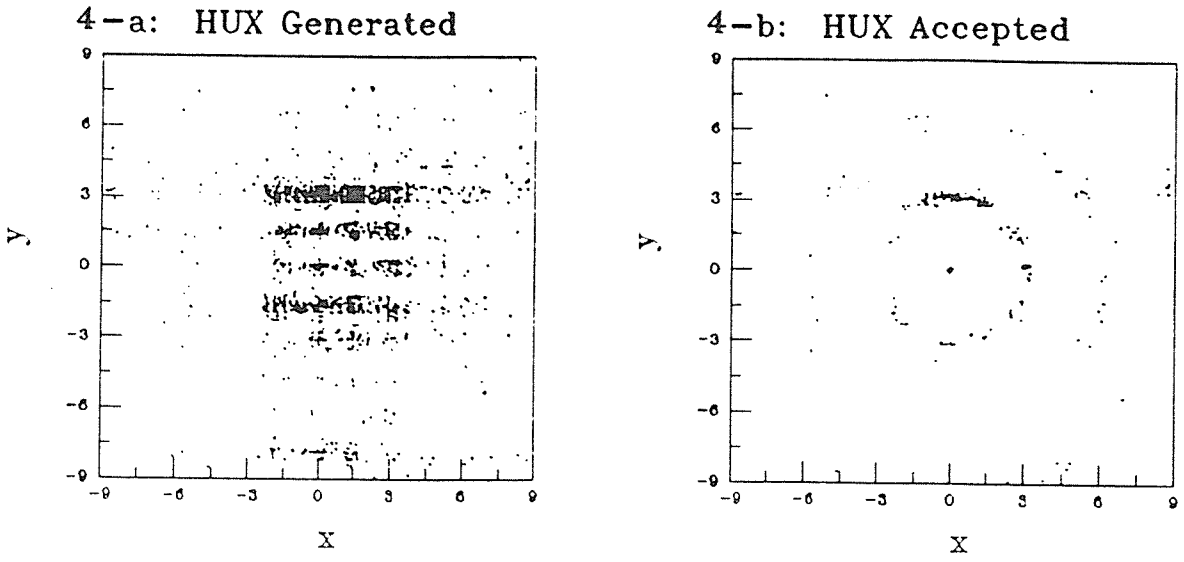

Figure 4: Function f6-shifted (output re-centered)

5 CROSSOVER VERSUS MUTATION

Although we are not making any claims about the general superiority of blend crossover, or BLX-0.5 in particular, the results of the empirical tests we have conducted have convinced us that BLX-0.5 is a powerful operator. It might be questioned, however, as to what kind of operator it really is. In particular, is BLX-0.5 really a crossover operator or is it more like a mutation operator? If one insists upon looking at BLX-0.5 from the symbol-schemata point of view it will not appear to qualify as a crossover operator. Even from the point of view of our interval-schemata framework, it fails to fully qualify as crossover—for unlike BLX-0.0, not all the interval-schemata commonly containing the parents are preserved in the offspring. In Radcliffe's terminology, BLX-0.5, unlike BLX-0.0, is not strictly a *respectful* recombination operator (Radcliffe, 1991). But before excommunicating BLX-0.5 from the growing communion of crossover operators, it is important to ask the converse question as to how it differs from mutation.

Unlike the typical mutation operator used with a real-coded GA, BLX-0.5's "step-size" is self-adjusting, and is a function of the extent to which the population is converged. If it is a mutation operator, it is a very special mutation operator that shares with crossover the property of increasingly focusing search. There are search algorithms in the literature that do use a dynamically adjusting step-size, so this won't necessary put BLX-0.5 in the crossover category (Beale & Bentley, 1984; Bäck, Hoffmeister & Schwefel, 1991). But unlike these other operators BLX-0.5 does not use aggregate information to determine the step size, but like crossover, uses the specific information contained in the parents being paired, i.e., in Radcliffe's terminology, it preserves some of the *locality formae* common to both parents (Radcliffe, 1991). This is an important feature that can easily be illustrated by an example. Suppose that the function has two parameters coded as two intervals. Furthermore, suppose that there are two good regions in the search space—one where both

parameters have values at the low extreme and the other where both parameters have values at the high extreme. If the population is about evenly divided between instances from both these regions, then any mechanism that bases the mutation step-size on aggregate statistics is going to have a difficult time focusing in on the good regions. On the other hand, BLX-0.5 will have difficulty only when the parents are from different regions. This will only happen half the time. The other half the time parents will be chosen from the same region and search will progress.

The important point is that BLX-0.5, like all true crossover operators, but unlike mutation operators, including ones that are dynamically adjusted, implicitly exploits higher order correlations. Genes are not adjusted simply on the basis of the aggregate value of other instances of the same gene. Crossover implicitly takes into account the interaction among the genes when generating new instances. This is the source of crossover's power as a search operator.

6 CONCLUSION

With the new tool of interval-schemata, the reasons behind the empirical successes reported for real-valued GAs can now be better understood. Both IPGAs and SPGAs have the property of implicit parallelism. They differ in their biases. IPGAs exploit local continuities, whereas SPGAs exploit discrete similarities. It is natural to expect that for different problems different biases will provided a competitive advantage.

One future line of research is to explore an algorithm that uses both crossover operators, creating half the children using BLX-0.5 and half using HUX. Preliminary results suggest that the performance of an algorithm using both operators is better than the average of the performances of each operator used separately, and on some problems, e.g., f6-shifted, performs better than either alone.

References

H. J. Antonisse. (1989) A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint, *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 86-91.

T. Bäck, F. Hoffmeister & H. Schwefel. (1991) A Survey of Evolution Strategies, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 2-9.

G. O. Beale & S. E. Bentley. (1984) Parameter Estimation Using Microprocessors and Adaptive Random Search Optimization, *IEEE Transactions on Industrial Electronics IE-31*, 1 85-89.

L. Davis. (1991a) Hybridization and Numerical Representation, in *The Handbook of Genetic Algorithms*, L. Davis (editor), Van Nostrand Reinhold, New York, 61-71.

L. Davis. (1991b) Bit-Climbing, Representational Bias, and Test Suite Design, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 18-23.

L. J. Eshelman. (1991) The CHC Adaptive Search Algorithm: How to Have Safe Search When Engaging in Nontraditional Genetic Recombination, in *Foundations of Genetic Algorithms*, G. J. E. Rawlins (editor), Morgan Kaufmann, San Mateo, CA, 265-283.

L. J. Eshelman & J. D. Schaffer. (1991) Preventing Premature Convergence in Genetic Algorithms by Preventing Incest, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 115-122.

D. B. Fogel & J. W. Atmar. (1990) Comparing Genetic Operators with Gaussian Mutations in Simulated Evolutionary Processes Using Linear Systems, *Biological Cybernetics 63*, 111-114.

D. E. Goldberg. (1990) Real-Coded Genetic Algorithms, Virtual Alphabets, and Blocking, IlliGAL Report 90001, Illinois Genetic Algorithms Laboratory Dept. of General Engineering University of Illinois at Urbana-Champaign, Urbana, IL.

D. E. Goldberg. (1991) The Theory of Virtual Alphabets, In *Parallel Problem Solving from Nature*, H. P. Schwefel and R. Männer (editors), Springer-Verlag, Berlin, 13-22.

C. Z. Janikow & Z. Michalewicz. (1991) An Experimental Comparison of Binary and Floating Point Representations in Genetic Algorithms, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 31-36.

N. J. Radcliffe. (1990) Genetic Neural Networks on MIMD Computers, Ph.D. Dissertation, Dept. of Theoretical Physics, University of Edinburgh, Edinburgh, UK.

N. J. Radcliffe. (1991) Forma Analysis and Random Respectful Recombination, *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 222-229.

J. D. Schaffer, R. A. Caruana, L. J. Eshelman & R. Das. (1989) A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization, *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, 51-60.

J. D. Schaffer, L. J. Eshelman & D. Offutt. (1991) Spurious Correlations and Premature Convergence in Genetic Algorithms, in *Foundations of Genetic Algorithms*, G. J. E. Rawlins (editor), Morgan Kaufmann, San Mateo, CA, 102-112.

N. N. Schraudolph & R. K. Belew. (1991) Dynamic Parameter Encoding for Genetic Algorithms, Technical Report LAUR90-2795, Los Alamos National Laboratory, Los Alamos, NM.

C. G. Shaefer. (1987) The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, Hillsdale, NJ, 50-58.

A. Wright. (1991) Genetic Algorithms for Real Parameter Optimization, in *Foundations of Genetic Algorithms*, G. J. E. Rawlins (editor), Morgan Kaufmann, San Mateo, CA, 205-218.