Complex Systems **5** (1991) 139–167

Real-coded Genetic Algorithms, Virtual Alphabets, and Blocking

David E. Goldberg*
Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Abstract. This paper presents a theory of convergence for real-coded genetic algorithms—GAs that use floating-point or other high-cardinality codings in their chromosomes. The theory is consistent with the theory of schemata and postulates that selection dominates early GA performance and restricts subsequent search to intervals with above-average function values, dimension-by-dimension. These intervals may be further subdivided on the basis of their attraction under genetic hillclimbing. Each of these subintervals is called a *virtual character*, and the collection of characters along a given dimension is called a *virtual alphabet*. It is the virtual alphabet that is searched during the recombinative phase of the genetic algorithm, and in many problems this is sufficient to ensure that good solutions are found. Although the theory helps suggest why many problems have been solved using real-coded GAs, it also suggests that real-coded GAs can be *blocked* from further progress in those situations when local optima separate the virtual characters from the global optimum.

1. Introduction

Real-coded or floating-point genes have a long, if controversial, history in artificial genetic and evolutionary search schemes, and their use as of late seems to be on the rise. This rising usage has been somewhat surprising to researchers familiar with fundamental genetic algorithm (GA) theory [16, 26] because simple analyses seem to suggest that enhanced schema processing is obtained by using alphabets of low cardinality, a seemingly direct contradiction of empirical findings that real codings have worked well in a number of practical problems. The debate between practitioner and theoretician over this *paradox of real codings* has risen almost to the point of schism. Theoreticians have wondered why practitioners have paid so little heed to the theory, and practitioners have wondered why the theory seems so unable to come to

*Electronic mail address: `goldberg@vmd.cso.uiuc.edu` terms with their findings. This paper seeks to resolve this apparent paradox through the introduction of a theory of operation for real-coded GAs. Specifically, the notion of *virtual alphabets* is introduced and used to shed some light on the convergence of real-coded GAs. In a sense, a problem coded with a high-cardinality alphabet is quickly reduced to searching over a much smaller virtual alphabet, and recombinative processing proceeds. The theory is consistent with known schema theory and helps explain why practitioners have obtained good results in many problems. The theory also demonstrates that real-coded GAs can be stymied by difficult problems that *block* the virtual characters from accessing (from being mutated to) the globally optimal point or points.

It should be noted that this new theory is somewhat distinct from, but a companion to, previous results on what makes a problem hard for a binary-coded GA. A previous study [17, 18] considered the conditions under which a simple, binary-coded genetic algorithm might be misled by a function-coding combination. That study also suggested the extension of the binary theory to other low-cardinality alphabets, and such extensions are directly applicable to real-coded GAs as long as deception is considered in light of the virtual alphabet. On the other hand, the blocking mechanism suggested herein has not been considered before, and is only a factor when high-cardinality alphabets are used.

Unless otherwise stated, the rest of the paper assumes that the GAs under consideration, whether binary or real-coded, are simple: they use selection, simple crossover, and simple mutation [16]. The analysis should be valid for most selection schemes that give more copies to better individuals. We assume the use of simple crossover operators, where genes (binary or floating-point) are exchanged whole, although the number of cut points and their location may vary. We assume that simple mutation changes 1s to 0s and vice versa in binary alphabets. In real-coded (and k-ary) GAs, simple mutation is assumed to respect the ordering of the alphabet and creeps up and down with either fixed- or variable-length strides. As is the case in most GA work, we ignore the need for change of representation operations, and when such operations are mentioned at all, we only consider reorderings such as inversion and recombinative reordering that attempt to get the necessary linkage between important gene combinations. These assumptions are fairly common, and they have helped promote fairly concrete analysis. But, it is important to understand that the analysis techniques presented herein extend to other known variants of crossover and mutation, and as we shall soon argue, these other operators appear to suffer the same performance limitations as simple real-coded GAs.

In the remainder, the history of using real-coded genes is considered briefly; reasons for using small and large alphabets are discussed, and the theory of virtual alphabets is set forth. This leads to an understanding of the limiting behavior of a real-coded GA, which directs our attention to the problem of inaccessible optima or blocking. The theory demonstrates that, while real-coded genes may not harm genetic processing in some problems and may even be helpful in others, there are problems where the coding places additional barriers to the search for global optima—barriers that do not exist when low-cardinality alphabets are used.

2. Past use of real-coded genes

The use of real-coded genes dates back to the earliest days of evolutionary and genetic methods. In this section, we consider both early and more recent studies that have adopted high-cardinality or real-coded genes.

One of the earliest suggestions linking artificial optimization and natural evolution came in Box's [4] evolutionary operations scheme. This suggestion was less an algorithm and more a method for systematically perturbing two or three decision variables about the current operating condition of some fixed plant. Nonetheless, the work was inspired by evolution, and the method for choosing solutions and the solution perturbation cycle may be viewed as selection-like and mutation-like operators, respectively.

Friedman [15] proposed the construction and digital simulation of what he called a *selective feedback computer*. It is unclear whether this design underwent testing, but the combination of selection and mutation mechanisms was fairly well spelled out, as was the use of real genes.

Some of the mechanisms within Selfridge's [33] Pandemonium machine learning system were inspired by evolution, and a number of these involved the manipulation of real-coded genes. Specifically, he considered the possibility of mutating weights randomly for his "demons" or processing units and then selecting the better combinations. Perhaps more interesting from the standpoint of modern genetic algorithms was his suggestion for a *conjugation* operator, where a pair of mated demons would give rise to offspring that were some logical combination of the parents. This early suggestion was very much in the spirit of more modern recombination or crossover operators.

Bledsoe [2] considered both binary and continuous (real) genes under selection and mutation. He considered both fixed-step and continuously distributed mutation operators. The paper is remarkable in that it foreshadowed many issues that did not reemerge until the 1970s and 80s, including GA-hard problems (he called them *lethally dependent*), scaling, the importance of populations, and a number of advanced operators.

Bremermann's work [5–8] considered binary, integer, and real genes. The earliest work concentrated on selection and mutation, and later studies included recombination operators (recombination by crossing over, recombination by averaging, recombination by majority vote). The problems considered included solving sets of linear equations and optimizing linear programming problems.

Contemporaneously and independently across the Atlantic, a couple of graduate students at the Berlin Technical University, Ingo Rechenberg and Hans Paul Schwefel, studied evolutionary techniques applied to the optimization of engineering systems under the name *Evolutionsstrategie* (ES). Rechenberg's early work concentrated on performing selection-mutation ESs with physical models of engineering systems. A least-drag shape, a lowest-resistance pipe bend, a minimum-weight structural truss, and a high-efficiency supersonic nozzle were among the objects considered. At about the same time, Schwefel constructed ES computer simulations that included mutation and selection, and later studies included recombination (shuffle or uniform crossover). Rechenberg's hardware-based studies are surveyed in a brief paper in English [30], although the work dates back to the early 1960s [29]. A fairly complete listing of the ES literature is available in Rechenberg's [31] bibliography. Schwefel's [32] text relates various *Evolutionsstrategien* to random search schemes and traditional optimization procedures.

Holland's contributions date back to 1962 [24], and from the start he took a discrete view of genetic processing. It wasn't until Weinberg's [34] thesis departed from this discrete tradition that a suggestion for real genes arose out of the Michigan school of thought. In that dissertation, Weinberg suggested a GA with real genes for the genetic evolution of a population of single-celled artificial organisms. The suggestion was fairly detailed but was not simulated, apparently because of the time consumed in doing the organism simulations.

Weinberg's work was followed by a flurry of real-coded gene work at Michigan [3, 14, 36]. The work included reproduction, crossover, various mutation operators, and inversion. Populations of size 40 were used and direct comparisons were made to conjugate-gradient techniques. Some analysis of individual operators was performed, but these efforts worked directly with the real codings and did not attempt to incorporate Holland's schema theory [25, 26].

Since these early efforts, the torch for real-coded genes has largely been carried by the German school, and these efforts are too numerous to mention individually. Lately, there has been a growing use of real-coded genes by researchers in the United States, although these people remain largely unaware of the long history of real-coded genes and the volume of ES work. A particularly vocal U.S. proponent of real genes has been Davis. His study of communication network design [10, 11] coded a discrete set of five communication speeds directly as a single gene and used an adjacency mutation operator (called *creep*). This flirtation with bigger-than-binary alphabets has progressed to the point where a recent study of using GAs in neural network weight adjustment used real-coded genes and a variety of mutation and recombination operators tailored to that task [28]. He also reports success in a variety of unpublished practical problems [12], and has questioned the utility of schema theory if it so badly predicts the utility of real codes. Although he makes the point without offering a theoretical explanation for his success in using high-cardinality alphabets, the point deserves a more considered response than has been offered thus far, and the primary purpose of this paper is to resolve the apparent paradox between empirical results and schema theory predictions for real-coded genes. The next section considers those aspects of schema theory that recommend low-cardinality alphabets.

Binary	Octal	Function value
000	0	22
011	3	8
101	5	11
111	7	3

Table 1: Binary vs. octal structures

3. Why small alphabets?

Fundamental theory suggests that small alphabets are good because they maximize the number of schemata available for genetic processing. The calculation is straightforward. Consider a GA coded over an alphabet of cardinality κ. Since there are $\kappa + 1$ schemata per position (the κ members of the alphabet plus a meta-wildcard character—a don't care—denoted by $*$), and each position represents $\log_2 \kappa$ bits, there are $(\kappa+1)^{1/\log_2 \kappa}$ schemata per bit of information for a code with cardinality κ. Thus we have proved the following theorem presented elsewhere [20].

Theorem 1. *There are $n_s = (\kappa + 1)^{1/\log_2 \kappa}$ similarity subsets or schemata per bit of information for strings coded using an alphabet of cardinality κ.*

This computation leads to another useful result.

Theorem 2. *For a given information content, strings coded with smaller alphabets are representatives of larger numbers of similarity subsets (schemata) than strings coded with larger alphabets.*

Proof. From Theorem 1, $n_s = (\kappa+1)^{1/\log_2 \kappa}$. Taking the log of n_s and calling this quantity r yields $r(\kappa) = \log_2 n_s = \log_2(\kappa + 1)/\log_2 \kappa = \ln(\kappa + 1)/\ln \kappa$. Evaluating the derivative with respect to κ we obtain

$$\frac{dr}{d\kappa} = \frac{[\kappa \ln \kappa - (\kappa + 1)\ln(\kappa + 1)]}{\kappa(\kappa + 1)\ln^2 \kappa}.$$

Since $\kappa \ln \kappa$ is monotonically increasing for all legitimate alphabets ($\kappa \geq 2$), the derivative of r is less than zero. Since exp is a monotonically increasing function, we have shown that smaller alphabets have larger numbers of schemata per unit of information. ∎

A simple example will help drive home these calculations. Suppose we have a choice of coding a problem in octal or binary. The examination of a small sample population will help convince us of the greater information that becomes available using smaller alphabets (table 1). Scanning the octal structures and their objective function values we are at a loss to know what to do next. Since each octal string represents itself alone, we can make no inferences regarding which of the unnamed structures might be particularly promising. On the other hand, scanning the binary structures, each string is a representative of a number of subsets of structures with similarities at one or more positions. Thus we can speculate on cause and effect relationships between good components and high fitness. For example, maybe the first string is highly fit because it has a 00 in its rightmost positions or because of its 0 at its rightmost position, or perhaps strings 000 and 101 are relatively fit because of the 0 they share in the middle. In this way, many hypotheses can be formulated regarding the association between substring values and high fitness, and it is this information that is recombined to speculate on possibly better structures during the normal course of genetic search.

This reasoning and the previous calculations are straightforward and hardly open to question, and both seem to drive us toward the conclusion that we should be using small alphabets; however, there are problems where the use of all these schemata is not necessary and may even slow down our search. For example, in linear problems, knowledge of high-order schemata is unnecessary because position-wise manipulation of the code permits us to achieve the optimum. In such cases, one can certainly count the high-order schemata, but counting them is little more than an academic exercise. This point about which schemata are important in problems of varying difficulty, and therefore should be counted when arguing over the advantage of one coding over another, has been made more forcefully and quantitatively elsewhere [19], but it is one that is often overlooked. In the next section, we consider a number of possible reasons why one might choose to ignore the schema advantage of low-cardinality codings.

4. Why large alphabets?

Although the largest numbers of schemata are obtained using the smallest alphabets, there are a number of reasons why a user of genetic-evolutionary techniques might choose to ignore this advantage:

1. comfort with one-gene–one-variable correspondence;
2. avoidance of Hamming cliffs and other artifacts of mutation operating on bit strings treated as unsigned binary integers;
3. fewer generations to population conformity;
4. reduction of the opportunities for normal-mode deception.

The first of these reasons is more psychological than technical, but many users find a one-to-one correspondence between genes and decision variables comforting and find the codings of decision variable as bit strings or other discrete codes disconcerting. As we have seen, much of the early work took the real approach almost without thinking, and Holland's suggestions for discrete structures and more effective schema processing have not always been appreciated.

Justification for the real-gene approach can be made on more technical grounds, however. Real-coded GAs usually adopt mutation operators that

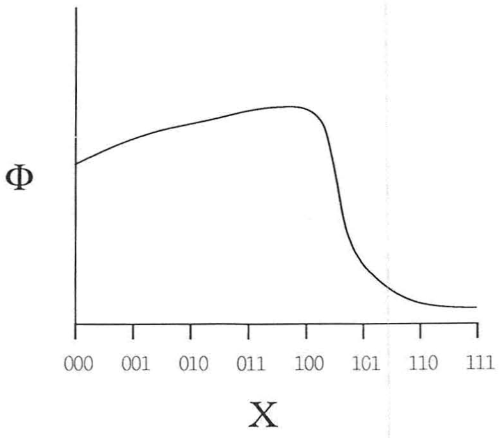

Figure 1: A three-bit function with a Hamming cliff.

perturb the current solution a little about the current value, and binary-coded GAs usually adopt a bitwise complement operator for mutation. By creeping up and down, real-coded GAs can easily do hillclimbing in the underlying decision space, but with bitwise complement mutation, binary-coded genes can become stuck on Hamming cliffs as illustrated in figure 1.

Using an octal coding over the eight points of the unimodal search space and some sort of creeping mutation, the higher cardinality GA will be able to find the best point regardless of initial population. After the population converges to some value, successive mutations will continue to correct the solution until the optimum is reached. On the other hand, depending upon initial convergence, the binary GA may or may not be able to access the best point. For example, because the points in the left half of the space are above average (because $f(0**) > f(1**)$), a GA is fairly likely to converge initially to 011. Although 011 is close to the correct solution as measured in decision space, it is quite distant in Hamming space and requires changes at all three bits to reach the optimum 100. Such changes are unlikely—they are of $O(p_m^3)$—and require long waiting times. A number of authors have suggested the use of Gray codes to overcome such problems [1, 9], but doing so introduces higher order nonlinearities with respect to recombination [18]. A simple solution to this problem exists. Simply use *both* bitwise and decision-variable-wise mutation operators in binary-coded GAs. Some thought should be given to the proper proportion of each, but doing so should permit binary-coded GAs to climb hills in the space (coding or decision) that is currently friendliest toward hillclimbing. Such a solution was recently used to good effect in Lucasius and Kateman's [27] study of chemometric applications of GAs, and others should find the technique appealing.

Another reason users may prefer to use higher cardinality alphabets is speed. Assuming a fixed population size, a fixed number of search alternatives, and serial processing of individual loci, it may be shown theoretically and empirically that higher cardinality alphabets converge to some solution more quickly than those coded over a smaller alphabet. It is important to recognize that this faster convergence is a mixed blessing as the quality of the solution degrades with increasing κ. Nonetheless, some users may be happy to let a GA converge quickly to some near-optimal solution, allowing selection and mutation or other local search procedures to polish off the search.

A final reason a user might consider a higher cardinality alphabet is because it reduces the dimensionality of the problem. This in turn reduces the opportunity for *deception* [17, 18] and reduces (or eliminates) crossover disruption within a parameter. Briefly stated, deception exists when low-order building blocks lead in one direction, but high-order building blocks containing the global optimum lie elsewhere; reducing the dimension of the problem can reduce the opportunity for deception because there are fewer low-order building blocks to confuse. This might be cause for celebration, but as we shall soon see, this is no free lunch. First, note that reducing problem dimension reduces the *opportunity* for deception. If a problem is in more than one dimension, deception can still exist and can cause convergence difficulty. Second, the introduction of higher cardinality alphabets introduces the possibility of new obstacles to convergence, obstacles that must be accounted before choosing any coding.

The reduction of crossover disruption might at first glance appear to be a blessing, undisguised. Unfortunately, it, too, can be a net advantage or disadvantage, depending upon the problem. There is no doubt that the reduction in crossover disruption increases the chances that selection will be able to pick out good alleles. On the other hand, as we've already argued, indivisibility prohibits lower order schemata from offering clues where other good (unvisited) points might be. This was the primary lesson of the counting argument of the previous section. Thus, the picture is not at all clear-cut.[1] Rather than arguing the merits of these alternatives on such primitive grounds, we try to better understand the processing of a real-coded genetic algorithm.

5. Selection dominates early genetic algorithm performance

To understand the performance of high-cardinality GAs, we must first recognize how selection dramatically cuts back on the alternatives that are considered early in a run. Two examples, first in one and then in two dimensions, will make this clearer.

[1]As Harry Truman longed for a one-armed economist, I find myself wishing for a one-armed genetic algorithmist in the hope of putting an end to all this on-the-one-hand-and-on-the-other stuff. On the other hand, economic systems share much of the interwoven complexity of evolutionary systems, and perhaps it is this commonality that drives, and must leave unfilled, requests for monodexterous individuals in both fields.

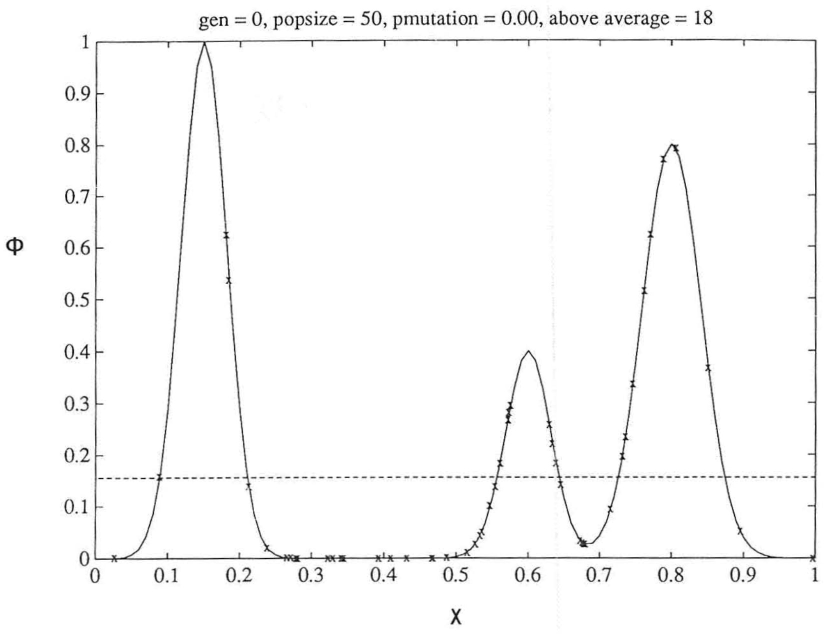

Figure 2: Trimodal function in one dimension is shown with a uniformly random initial distribution of points. The horizontal line shows the average value of the function.

5.1 A function of one variable

To understand the power of selection alone, consider the one-dimensional function with three peaks as shown in figure 2. Initially, a random population of points is chosen (as shown by the x's placed on the curve), and binary tournament selection is used in successive generations. In the initial generation, a significant proportion of the population is above average, and in succeeding generations this proportion should be expected to grow no more slowly than logistically [21]. At generation 4, all the points are above average as shown in figure 3. This leads us to wonder whether some special role is played by above-average points in the space. After all, in a very short time, through the action of selection alone, search on the full unit interval has been cut to only those subintervals that are above average. Furthermore, the addition of some small amount of creeping mutation or other genetic operators should not materially affect these results; early real-coded GA performance is dictated by (1) the points present in the initial population and (2) the action of selection to restrict consideration to the best of those points. We will develop this notion more fully in a moment, but first we must consider what happens in functions of more than one variable.

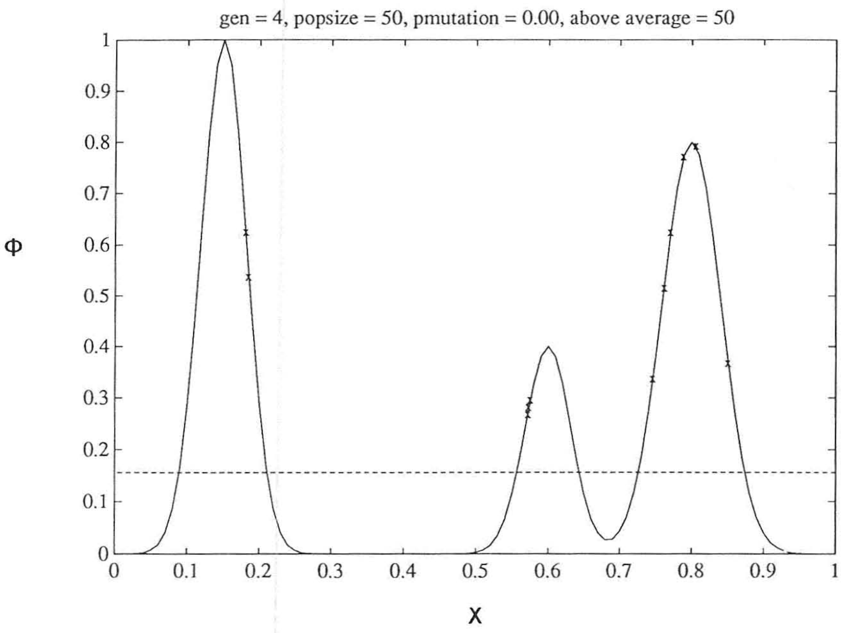

Figure 3: The distribution of points on the trimodal function at generation 4 shows that selection quickly chooses above-average points.

5.2 Functions of more than one variable

The picture for functions of more than one variable is similar to the one just painted in that above-average points are preferred by selection, except now consideration of above-average intervals must be done dimension-by-dimension, averaging out the effect of the other variables. To understand this, examine the function of figure 4. To get a meaningful view of the function with respect to the first decision variable x_1, simply average out x_2 as follows:

$$\bar{\phi}(x_1) = \frac{1}{b-a}\int_a^b \phi(x_1, x_2)dx_2.$$

Such *mean slices* will be useful in evaluating the early effect of selection on functions of more than one variable. The averaging of the mean slice may be connected to the random sampling that is actually performed initially if we think in terms of *Monte Carlo integration.* One common way to obtain approximations to definite integrals is to sample the integrand over the domain of integration randomly, and this is exactly what is being done by the GA during the initial population. Calculating the mean slice simply takes this view literally and more formally.

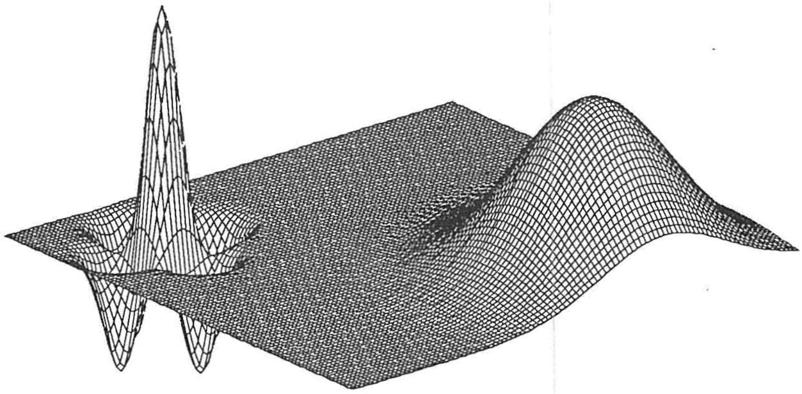

Figure 4: A two-dimensional objective function is used to illustrate a GA's dimension-wise preference for above-average points as a result of selection.

Another useful view of the function may be taken by fixing the second decision variable to its value (x_2^*) at the global optimum:

$$\phi^*(x_1) = \phi(x_1, x_2^*).$$

This *slice at the global optimum* or *global slice* is useful in evaluating convergence behavior when (if) solutions approach the neighborhood of the global optimum. Another view, the *max slice*, may be obtained by varying x_1 and choosing the best ϕ value at each point ranging over all possible x_2 values; and analogously a *min slice* may be defined by choosing the smallest function value. Although max and min slices are not used directly in the theory that follows, they do provide useful information at a glance regarding the relation of the mean at a point to the maximum and the minimum, thus helping to indicate whether the mean is a good predictor of a particular optimum.

Figure 5 shows the max, global, mean, and min slices for the two-dimensional function of figure 4. Initially the points are randomly distributed, but the figure shows the distribution after four generations of tournament selection and simple crossover. Nearly all points are above average and are distributed in only those places that have above-average performance on the mean slice. Note that the narrow feature containing the global optimum has no representatives. This is not surprising because points with an x_1 value in that range are not particularly good on average, and obtaining an initial sample on the fairly narrow two-dimensional peak is a low-probability event.

In the next section, we build on the various slices to define virtual characters and virtual alphabets.

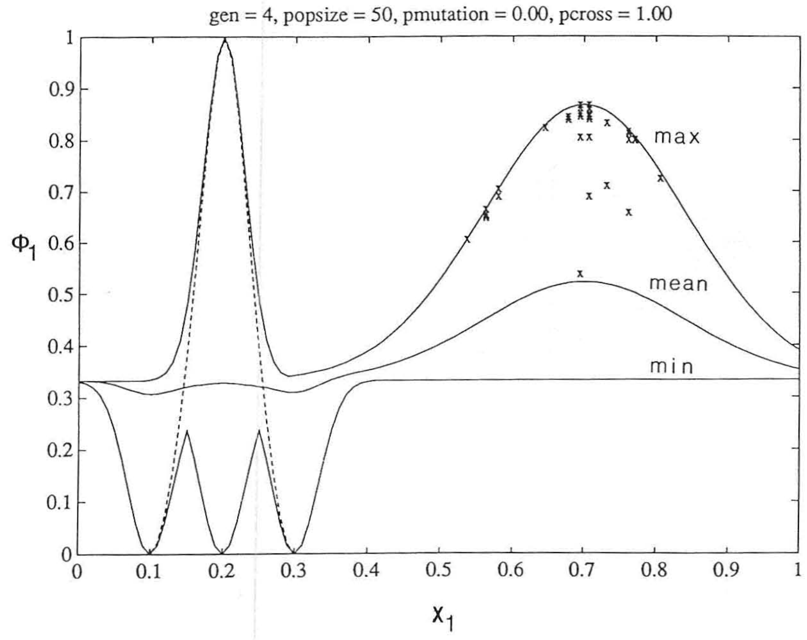

Figure 5: A max slice, global slice, mean slice, and min slice along x_1 for the two-dimensional function of figure 4 shows x's marking the 50 points contained in the population at generation 4. The location of the points coincides roughly with the location of the above-average points of the mean slice. The global slice is shown as a dashed line and coincides with the min slice at a distance from the global optimum.

6. The theory of virtual alphabets

The last section suggested that selection acts quite quickly to emphasize highly fit individuals, thereby reducing drastically the number of alternatives considered. Viewed along a single dimension, the genetic algorithm tends to allocate trials to those intervals associated with above-average function value. These intervals may then be viewed as the basic building blocks for further genetic search via operators such as crossover and mutation, and it is this view that is developed more rigorously in this section. Specifically, this loose talk of "useful intervals" is made more rigorous through the introduction of the notions of *virtual characters* and *virtual alphabets.* These concepts turn out to be the keys to understanding the schema processing that takes place in high-cardinality GAs.

6.1 Some mathematical preliminaries

Assume that the object of search is a real-valued function ϕ with domain Ω in R^n. The function ϕ is said to have a *local maximum* at x^* if a $\delta > 0$

exists such that $\phi(x) \leq \phi(x^*)$ for all $x \in \Omega$ with $|x - x^*| < \delta$. A *strict* local maximum is said to exist when the inequality in ϕ may be replaced by a strict inequality ($<$), and a *global* or *strict global* maximum may be defined by requiring the appropriate inequality on ϕ to apply throughout Ω. Likewise, analogous definitions for local and global *minima*, either strict or not, may be defined by reversing the direction of the inequalities on ϕ. In the remainder, we consider functions ϕ to be maximized and use the terms optimum and maximum interchangeably.

6.2 Slices

It is also useful to generalize the slicing notions developed intuitively somewhat earlier. Consider an appropriate probability density function $f(x)$ with $x \in \Omega$, and recognize that the marginal density function $f_I(x_I)$ for the subset of the n real variables specified by the index set I may be calculated as

$$f_I(x_I) = \int_{\Omega_{I'}} f(x) d\Omega_{I'}, \tag{6.1}$$

where $I' = \{1, \ldots, n\} - I$, the complement of the index set I. With these definitions, the *mean slice* $\bar{\phi}_I$, or the expected value of ϕ with respect to f and the free variables x_i $(i \in I)$, may be calculated as follows:

$$\bar{\phi}_I(x_I) = \int_{\Omega_{I'}} f_{I'}(x_{I'}) \phi(x) d\Omega_{I'}, \tag{6.2}$$

where $d\Omega_{I'} = \prod_{j \in I'} dx_j$ and $\Omega_{I'}$ is the indicated $|I'|$-dimensional subspace of Ω spanned by the averaged variables. When the index set is empty, $\bar{\phi}_I(x_I)$ reduces to $\bar{\phi}$, the expected value of ϕ with respect to f. When the index set I is singleton, the slice is one-dimensional like the one considered in the last section; and since the singleton slices are so important, a shorthand notation of subscripting $\bar{\phi}$ by a single index will be used. For example, $\bar{\phi}_1(x_1)$ is the expected value of ϕ with respect to f for the single free variable x_1.

The notation is straightforward, but we should pause and check its meaning. Essentially we have generated a set of functions that are the averages of ϕ with respect to any number of variables. Of course, in discrete GAs this is the role played by schema fitness averages, and the $\bar{\phi}$ functions play that same role in real-coded GAs. In fact, the schema theorem carries over to the real-coded case as pointed out by Wright [35]; as with most codings, proper interpretation of a schema and appropriate definition of genetic operators with high schema survival probabilities are enough to ensure that the schema theorem applies.

The notion of taking a slice through the best point may also be generalized by first considering a *point slice*. The point slice $\phi_I^a(x_I)$ with respect to the point $x = a$ and index set I is defined as the function obtained by setting $x_j = a_j$ $(j \notin I)$ and allowing the other variables to vary freely. An *optimum slice* $\phi_I^*(x_I)$ is obtained by choosing the reference point as any optimum x^*, and a *global slice* is obtained when the point selected is globally optimal. Of course, when multiple optima or global optima exist, care should be exercised to consider the various slices separately.

The *max slice* of the previous section may be generalized from the singletons to arbitrary index sets. Specifically, the max slice $\phi_I^{\max}(x_I)$ with respect to an index set I is defined as follows:

$$\phi_I^{\max}(x_I) = \max\{\phi(x) : x \in \Omega \text{ and } x_i \text{ fixed, } i \in I\}. \tag{6.3}$$

In other words, for a given fixing of the x_I, the max slice displays the largest value of ϕ ranging over the remaining variables. An analogous definition yields a *min slice* straight away.

We will use these various slices in a moment to help understand real-coded GA processing. However, since genetic algorithms contain elements of both random search (during initialization) and hillclimbing (via selection plus mutation), we need to consider each of these modes of operation.

6.3 Random search

Consider performing a random search for a *target* $\tau \subset \Omega$ according to the random search probability density function $f_r(x)$ with $x \in \Omega$. The probability P_τ of finding a point within the target on a given random trial may be calculated as follows:

$$P_\tau = \int_\tau f_r(x) d\Omega. \tag{6.4}$$

In n_r trials, the chances of having at least one success is $1 - (1 - P_\tau)^{n_r}$. We say that the target τ is a *needle in a haystack* (NIAH) or *probabilistically lost* at the level α if $1 - (1 - P_\tau)^{n_r} < \alpha$. Rearranging, a target is NIAH if $P_\tau < 1-(1-\alpha)^{1/n_r}$, which may be solved approximately as $P_\tau < -(\ln(1 - \alpha))/n_r$, the approximation improving with increasing n_r.

6.4 Hillclimbing

In many search problems, hillclimbing—whether performed genetically through selection and mutation or otherwise—may be sufficient to find good solutions. To separate such relatively easy problems from those that require more sophisticated genetic processing, such as recombination, reordering, niching, and expression, we consider an abstract framework for hillclimbing algorithms, their basins of attraction, and whether repeated hillclimbing is likely to achieve a solution.

There are many forms of hillclimbing algorithm available to a search algorithm designer or user. When mutation is enabled in a GA, its action together with selection promotes a broad hillclimbing mechanism, and in restricted problem domains, specialized local search algorithms are often available. We are less concerned here with the details of how hills are climbed in a problem and more interested in identifying whether we are at all likely to climb the right hill in a particular problem. With this is mind, we imagine a *hillclimbing system* (HCS) as a five-tuple (X, M, Φ, g, h), where $X \subset \Omega$ is the allowable search space, M is the space of allowable memory configurations, Φ is the set of possible function values, and g and h are memory transition functions and output functions defined as follows:

$$\begin{aligned} m_{t+1} &= g(m_t, \phi_t); & (6.5) \\ x_{t+1} &= h(m_t, \phi_t). & (6.6) \end{aligned}$$

In words, the memory $m \in M$ and search point $x \in X$ are updated depending on the ϕ value received from the "environment" of the HCS and the current state of HCS memory. This definition of a hillclimber is broad enough to encompass most algorithms. Probabilistic schemes such as simulated annealing and GAs (with mutation and selection) are covered if the transition functions are viewed as determining a probability distribution over the search and memory configuration spaces. Note that the output, x_{t+1}, is viewed as a single point (as it is in most hillclimbers), although this assumption does not prevent the formalism from encompassing population-oriented schemes simply by varying the amount and type of information retained in memory.

Once a hillclimber is specified, it is useful to talk about its ability to find a particular optimum. We say that an HCS is *attracted to* a point x^* from a point x if application of the HCS starting within a δ-neighborhood of x results in a non-zero probability of being within an $\epsilon > 0$ of x^* or within an ϵ of any ϕ-equal point connected to x^*. The basin of attraction $\mathcal{B}_{x^*}$ is then defined as the maximal connected set of points $x \in \Omega$ attracted to x^*. For a deterministic HCS the probability of randomly choosing a point in the basin leading to x^* is simply

$$P_{x^*} = \int_{\mathcal{B}_{x^*}} f_r(x) d\Omega, \tag{6.7}$$

and this is the probability of being attracted to x^* because, in the deterministic case, choosing a point in the basin guarantees ultimate arrival.

For a probabilistic HCS, if we define the conditional probability density function $f_{x^*}(x)$ of arriving near x^* (or near some points in a ϕ-equal connected region) given a start near x, the probability of ending up at the desired optimum after a single trial of random initialization and hillclimbing is simply

$$P_{x^*} = \int_{\mathcal{B}_{x^*}} f_r(x) f_{x^*}(x) d\Omega. \tag{6.8}$$

Of course, it is no longer necessary to restrict the domain of integration to the basin, as the second density function properly accounts for those points that are unable to access a particular optimum through hillclimbing, but the restriction causes no harm and further emphasizes which points can lead to Rome.

It is now possible to connect these calculations to those of the previous subsection by recognizing that, when using randomly started hillclimbing (as opposed to random search), it is no longer necessary to hit the target on the head; we must only find a point in the appropriate basin of attraction.

In unimodal problems this usually poses no difficulty, and in problems with extensive global basins, waiting times are quite reasonable. On the other hand, problems exist with fairly limited global basins, and it is these problems that we hope to solve with the innovative capability of a recombinative GA. Assuming the repeated application of random start followed by the application of an HCS to convergence, we say that a function is *hillclimbing hard* or simply *HC-hard* if its global basin of attraction is NIAH (needle in a haystack). Otherwise, we say the problem is *HC-easy*. Because hillclimbing algorithms are so widely studied, and because even genetic search has its own built-in hillclimber (mutation plus selection), we assume that HC-easy problems can be solved, and restrict further consideration to those problems that are HC-hard.

6.5 The action of selection

In an earlier section, we saw empirically how selection dominates the early performance of a genetic algorithm. We recognize that, in a relatively short time, only individuals with relatively high function values will be represented in the population. To quantify this, assume binary tournament selection (or linear ranking selection with two copies given to the best and none given to the worst), and further assume a uniformly distributed objective function such that roughly 50% of the individuals in the initial population are above average. It may then be shown by the methods of another paper [21] that the number of generations t required for above-average points to all but take over the population may be calculated as

$$t = \log \log n_p, \tag{6.9}$$

where n_p is the population size and the logarithm is taken base two. Even with a population of 10^9 individuals, the number of generations needed to take over is roughly $t \approx 5$, and for typical population sizes (30–1000), three or four generations is enough to fix the population at a function value that is above average.[2]

With this viewpoint of rapid takeover by above-average points, and recalling that a *level set* L of ϕ with level λ may be defined as the set

$$L(\phi, \lambda) = \{x : x \in \Omega \text{ and } \phi(x) \geq \lambda\}, \tag{6.10}$$

we define the *selection set* or *S-set* as the level set with level $\lambda = \bar{\phi}$. In words, the S-set is the portion of Ω with above-average ϕ value.

We subdivide the S-set into connected subsets, calling each maximal connected subset of the S-set a *feature*, as each tends to stand out above the

[2]It may be argued that I have been insufficiently conservative in my assumptions in this calculation. But even if the population starts from a lesser state, and even if the selection scheme used is less pushy, the takeover time for many selection methods grows no more slowly than as a logarithmic function of n_p [21]. It is possible to be more precise about the level of convergence obtained in a particular number of generations for a given type of selection, although this has not been done here so we may carry on to the main result.

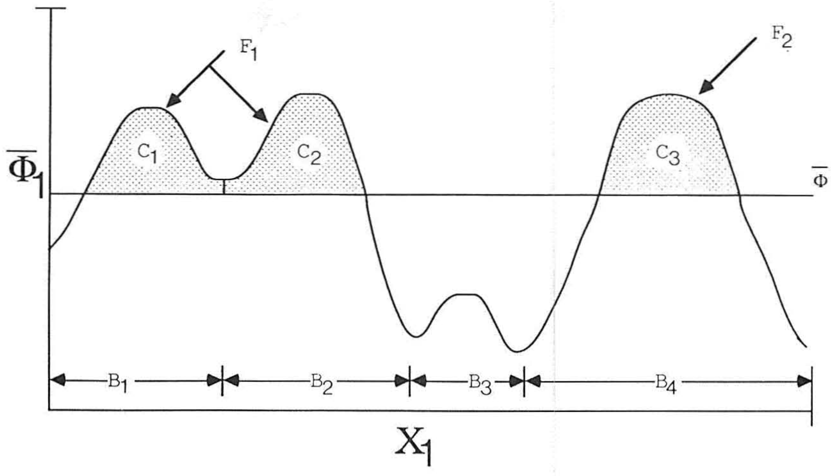

Figure 6: The sketch shows a mean slice, its basins, its features, and its virtual characters. Note that the definition allows for overlapping virtual characters, depending upon the choice of a hillclimbing system, even though the figure shows disjoint sets.

crowd. With these preliminaries, we rigorously develop the notion of a virtual character.

6.6 Virtual characters and alphabets defined

The definition of a feature gets us close to a rigorous statement of the ideas developed loosely regarding above-average intervals. Restricting our attention to the mean slices of ϕ over singleton index sets—the one-dimensional mean slices $\bar{\phi}_i$—a *virtual character* may then be defined as a non-empty intersection of a basin and a feature on ϕ_i, and a *virtual alphabet* is simply the collection of virtual characters along a given dimension. The sketch of figure 6 illustrates the basic ideas. In the figure, the selection set is divided into two features and four basins. The first feature (F_1) contains portions of two different basins (B_1 and B_2) and is divided into two virtual characters (C_1 and C_2). The second feature (F_2) contains a portion of only one basin and as a result has only a single character (C_3). Note that the third basin is unrepresented as a character. Even though it is locally optimal, selection will prefer other regions early on, and B_3 will be ignored.

This view is fairly straightforward, but at least two questions come to mind as we think of the genetic processing that follows the initial selective phase. How are virtual alphabets processed by the combined action of selection and recombination, and why don't we need to consider mean slices of two, three, and more dimensions in defining the virtual alphabet?

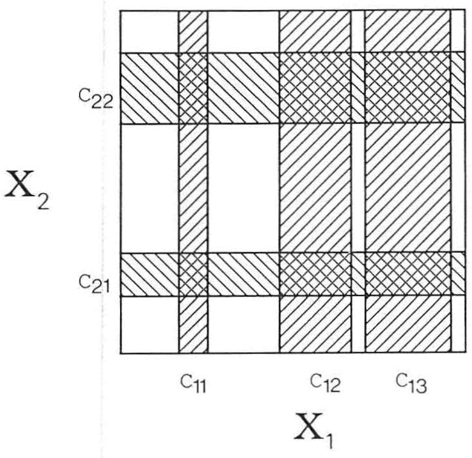

Figure 7: Selection and recombination in a real-coded GA process the virtual alphabets much like a low-cardinality GA processes its fixed alphabet. In two dimensions, this biases the processing to the cross-hatched regions. In higher dimensions, the picture is the same except that the linkage biases of the particular crossover operator must be accounted in determining the varying probabilities of visiting particular regions.

The first of these questions is the easier to answer. Reduction to the virtual alphabet so soon means that selection and recombination will process the virtual alphabets as though they were some underlying set of fixed alphabets. For example, in two dimensions a ternary alphabet along the x_1 axis and a binary alphabet along the x_2 axis might give a search picture something like that shown in figure 7. The regions of likely subsequent search are shown as cross-hatched intersections between virtual characters on different dimensions. Of course, this picture is somewhat modified by the action of mutation because mutation permits the system to climb hills one dimensionally once the solution has been transported to a point in some other cross-hatched region. (Actually mutation is occurring simultaneously, but for mutation steps that are small with respect to the virtual character width, the errors made by assuming no change during the recombinative phase are small).

The more difficult question is why consider only the one-dimensional slices? The answer is that features on slices of higher dimension are unlikely to be sampled properly. Imagine a problem specified on the n-dimensional hypercube, a problem mapping $[0,1]^n \rightarrow R$, and assume that features occupy a width δ of the unit interval in each dimension. Thus, one-dimensional features occupy a proportion δ of the unit interval, features in two dimensions occupy δ^2 of the unit area, features in three dimensions occupy δ^3 of the unit volume, and, more generally, features in k dimensions occupy δ^k of the k-dimensional hypercube. Moreover, the populations are likely to be sized so that the smallest character receives only $O(1)$ samples; put another way, $\delta \approx 1/n_p$. When this is done, the probability of having representatives of all two-dimensional features in a population is itself of $O(1/n_p)$ and, as a result, we may say that such features (and features in higher dimensions) are NIAH for GAs that are run with fewer than $O(n_p)$ restarts. Thus, the simple theory suggested here is a reasonable first cut, and mean slices in higher dimensions may safely be ignored.[3]

7. Blocking

Virtual characters and alphabets provide a useful perspective from which to view the convergence mechanisms of real-coded GAs. Simply restated, one-dimensional basinic features are selected early in the GA dimension-by-dimension, and the collection of virtual alphabets thus selected is used in subsequent recombinative-selective search. On the positive side of the ledger, this mechanism seems to sidestep the precision and aliasing problems that may occur when low-cardinality codes are used by allowing real GAs to adaptively select their own alphabets. The empirical success enjoyed by users of *Evolutionsstrategien* and real-coded genetic algorithms can in large part be explained by this single factor. Moreover, we note that the convergence mechanism described here is consistent with the theory of schemata and essentially says that, if you don't present a selectionist method with a low-cardinality alphabet, it will choose one for you. On the other hand, the convenience of having a selection-selected alphabet has been bought at a price, as it is possible to imagine impediments to subsequent search that are a direct result of allowing the alphabets to be so chosen. In this section, we consider the possibility of blocking and establish some necessary conditions for its occurrence on polynomial fitness functions.

7.1 Simple blocking

To see that real-coded GAs can be thwarted from finding global optima, consider the sketch showing a mean slice and a global (point) slice of a hypothetical multi-dimensional function as shown in figure 8. Considering the mean slice first, we see that there are two virtual characters and, by the reasoning of the last section, these will be preferred early in the search. Later on, assuming that search along the other dimensions has largely been

[3]On the other hand, when larger populations are used or fairly broad k-dimensional features exist but are not well predicted by low-dimensional features, the methods of this paper can be adopted to sort out meaningful algorithm mechanism. Specifically, k-dimensional basinic features become the *building blocks* of subsequent solutions, and special care can be taken to add the salient combinations that are not represented dimension-by-dimension. Suppose a, b, and c represent intervals along dimensions 1, 2, and 3, that are not above average, but suppose the three-dimensional structure abc contains a highly fit feature that is not NIAH. Then the building block abc (call it a *compound virtual character*) should be added to the evaluation of final convergence as long as the crossover operator adopted and the ordering chosen permit the building block to remain linked with high probability.

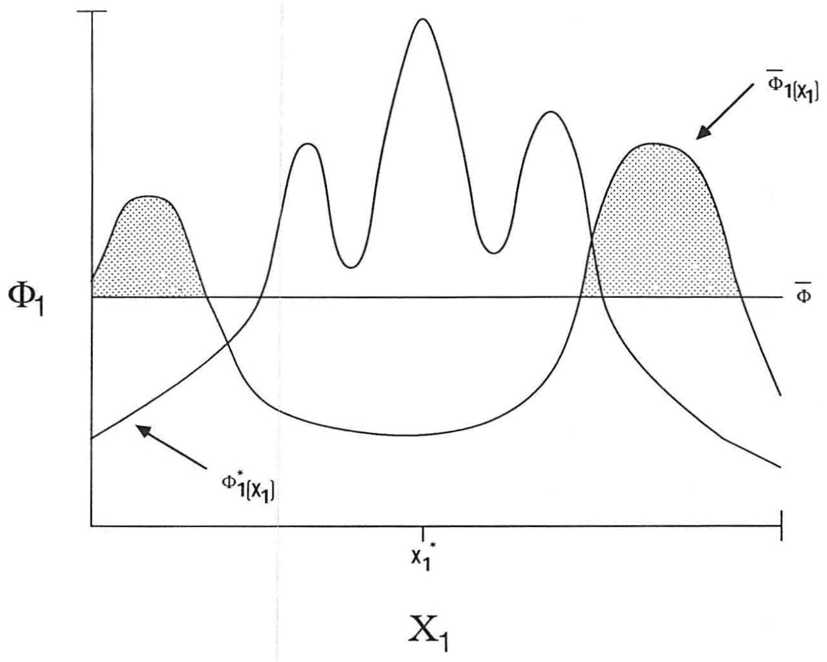

Figure 8: A mean slice and a global (point) slice of a hypothetical function illustrates the notion of blocking. The real GA will prefer the virtual characters (shaded), but when the GA gets in the vicinity of the optimum, it will be prevented from advancing beyond the local optima that block the global optimum.

successful, the picture should switch to that of the slice through the global optimum. Note in this case that the virtual characters will be prevented from finding the global optimum because selection and mutation will only be able to perform hillclimbing and will get stuck on one of the two local optima guarding the global optimum. We say in such cases that the global optimum is *blocked*, and call the one-dimensional form of blocking *simple blocking*. Higher order blocking can be visualized using mean and global slices in higher dimensions, and these should be researched and categorized. The important thing to recognize at this point is that there are limits to the use of real-coded GAs—limits that must be recognized and attacked with careful analysis, algorithm design, and innovation.

It might be suggested that the trouble here is simply the operators being used; perhaps by shifting to other mutation or recombination operators, the problem can be sidestepped. Although there may be operators that effectively avoid blocking, other variants of crossover and mutation in current use are likely to be only of limited relief. At any rate, analysis of these other operators proceeds directly using the theoretical tools developed here.

For example, one might suggest a form of mutation that jumps anywhere within the allowable parameter interval to overcome blocking. In theory, since the GA is no longer restricted to the asymptotic hillclimbing behavior of selection and creeping mutation, it can get unstuck. Unfortunately, such operators are very disruptive and can only be used with low probability. Additionally, for a jump-mutated offspring to survive, it had better jump to a point at or above the current average fitness. Point slices through the likeliest individuals (the most highly fit recombinations of the virtual characters) can be checked to determine whether such jumps are going to do much good, but some simple reasoning suggests that they won't often be of much help. The virtual characters are located where they are because the feature or features associated with that interval are of sufficient breadth and height to stick out above the crowd. Jumping to an above-average, unrepresented point that can hillclimb to the global optimum is an unlikely event. In other words, the line search of jump mutation is likely to fail because good features that are not close to already-represented virtual characters are likely to be NIAH with respect to that search.

Similarly, the use of averaging recombination operators is unlikely to be of much practical help in overcoming blocking. There are many variations on averaging recombination. One can simply average parents dimension-by-dimension; one can choose a single random parameter $\alpha \in [0,1]$ and simply take the convex combination of the parents, $\alpha x_1 + (1-\alpha)x_2$, as an offspring; or one can choose a set of α_i, one for each dimension, and take a convex recombination componentwise. Again, each of these (and their many mutants) theoretically offers some hope against blocking because each can jump somewhere very different from current parents. As with jump mutation, however, the chance of hitting a useful target is quite small. The individual point, line, or regional searches implied by these recombination operators can be investigated using the various slices defined in this paper, thereby determining whether good points are likely to be found in a particular problem. In general, however, finding features that are not represented by virtual characters is likely to be an NIAH subproblem.

7.2 Some necessary conditions for simple blocking of polynomials

The previous discussion of simple blocking can be made somewhat more quantitative if we restrict ourselves to the polynomials. Of course, we are interested in functions other than polynomials, but because many functions can be well approximated by power series, and many such series are convergent and can be truncated without great loss of information, it is useful to explore some necessary conditions for blocking on polynomials.

Restricting ϕ to a polynomial over the decision variables x_i, and referring to figure 8, we note that there must be a total of five extrema (three peaks, two valleys) in ϕ. As a result the derivative of ϕ must have at least five 0s, and the polynomial must be at least of degree six to exhibit simple blocking. Note that this example is a case with *two-sided blocking*, where one or more virtual characters on each side is separated from the global optimum by at least one local optimum per side. Thus we have proved the following:

Lemma 1. *For active, two-sided simple blocking to occur on a polynomial ϕ, the function must be at least of degree six in the blocked decision variable.*

The term *active* is used here to indicate that the blocking is caused by the interposition of a local optimum between a virtual character and the global optimum.

One-sided active blocking can occur when no virtual characters exist on one side of the global optimum. There are two possibilities in this case: the global optimum occurs on the boundary, or the global optimum occurs in the interior of the interval. When the global optimum occurs on the boundary, two extrema are necessary for active blocking (one local minimum, and one local maximum), and the polynomial must be no less than degree three. When the global optimum occurs in the interior of the interval there are three extrema (the global optimum, a local minimum, and a local maximum), and the polynomial required is of degree four or more. Thus we have shown the following:

Lemma 2. *For active, one-sided simple blocking to occur on a polynomial ϕ, the function must be at least of degree three in the blocked variable when its global optimum is located on the boundary, and degree four when the global optimum is interior.*

This reasoning is straightforward, but we might ask whether active blocking (the interposition of a local optimum between a virtual character and the global optimum) is necessary to prevent a virtual alphabet from reaching the global optimum. Interestingly enough, the answer is no.

The picture painted earlier is a little too strict. If a virtual character is not actively blocked, but is located on a *slope* leading away from the global optimum, we say that the problem is *passively blocked* or *slope blocked.* Since this is a weaker form of blocking, it is not surprising that the polynomials required are of lower degree. Two-sided slope blocking requires a single global optimum and two local minima with the virtual characters located outside the local minima protecting the global optimum as shown in figure 9. Thus, with three extrema, passive two-sided blocking requires a quartic in the blocked variable:

Lemma 3. *For passive two-sided simple blocking to occur on a polynomial ϕ, the function must be at least of degree four in the blocked variable.*

One-sided passive blocking can also occur, and if the global optimum exists in the interior there are two extrema, the global optimum and the local minimum; a cubic polynomial is required. If the global optimum exists on the boundary, only a single local minimum is required, thereby dictating a quadratic polynomial:

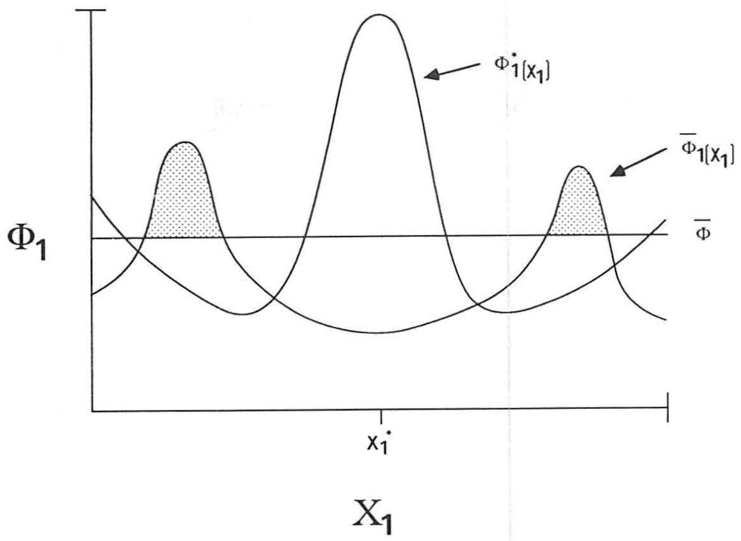

Figure 9: An illustration of two-sided passive blocking shows the virtual characters on the slopes of the global slice leading away from the global optimum.

Lemma 4. *For passive one-sided simple blocking to occur on a polynomial ϕ, the function must be at least of degree two in the blocked variable when its global optimum is located on the boundary, and degree three when the global optimum is interior.*

These results can be gathered together if we define a number of Boolean variables. Requiring the *activity* a to take value 1 when blocking is active and 0 otherwise, requiring the *sidedness* s to take value 1 when blocking is two sided and 0 otherwise, and requiring the *locale* l to be 1 when the global optimum is interior to Ω and 0 otherwise, the previous results may be stated compactly:

Theorem 3. *To exhibit simple blocking of activity a, sidedness s, and locale l, a function ϕ must be of degree $2 + a + s + l + a \cdot s$ in the blocked variable, subject to the requirement that $l = 1$ when $s = 1$.*

That many practical problems are of low degree and often are no worse than quadratic helps explain why *Evolutionsstrategien* and real-coded GAs have been fairly successful. Since active blocking requires a polynomial of degree three, four, or six, problems of lesser degree should not usually be in the blocking ballpark. In these cases, real-coded GAs may be preferable to discrete-coded GAs because the use of the adaptively selected virtual alphabet can overcome the known precision and aliasing problems of fixed discrete codes. On the other hand, it is premature to state this too strongly, as the possibility of higher order blocking must be investigated. At any rate, that problems can be blocked at all should be fairly alarming to current users of real-coded GAs. The existence of a large class of functions that thwart the underlying mechanism of an algorithm is never good news, and researchers and users of real-coded GAs need to investigate further both their functions and their algorithms in an effort to find out whether blocking is a significant practical problem, and if it is, to determine how or whether it may be overcome.

8. Ramifications and extensions

So what's a genetic algorithmist to do? Given a choice between binary-coded, real-coded, or in-between-coded GAs, which should we choose to obtain the best performance most of the time? If this paper has demonstrated one thing, it is that the decision is far from clear cut. Binary-coded GAs do have an abundance of schemata available for processing, which can lead to rapid processing of problems of bounded deception, but they can be thwarted by more fully deceptive problems or when linkage is inappropriate to the degree of deception. Real-coded GAs adaptively choose their own virtual alphabets, which can lead to more rapid discovery of near global points in easy problems, but they can be stymied by blocking and deception, singly or in combination.

The course we chart therefore depends upon our confidence in a GA's ability to defeat its enemies. Lack of such confidence in the ability of binary-coded GAs to defeat problems of bounded deception led to the recent invention of messy genetic algorithms [22, 23]. It has been conjectured that messy GAs defeat problems of bounded deception in polynomial time, and both empirical results and asymptotic theoretical analysis support this claim. This grounding gives us a good bit of confidence in our ability to solve difficult problems quickly and effectively.

Simple real-coded GAs can suffer at the hands of deception and blocking, and while there is some hope that messy techniques can help alleviate deception difficulty within real-coded GAs, there is a good bit of doubt whether averaging recombination or other operator variants can effectively alleviate blocking. Carefully designed empirical studies and more analysis are needed (and are underway). In the meanwhile, the practitioner needs to get on with his work and make a decision.

Perhaps the most rational response at this juncture is simply to decide, and not agonize over one's coding. If one is concerned with having some theoretical assurance that problems of bounded difficulty can be solved to global optimality, then perhaps proven messy techniques and binary codings should get the nod. On the other hand, as we have seen in this paper, genetic algorithms do something with whatever codings and operators we hand them, and oftentimes that something is surprisingly good. To summarize the present case, recall that a little theory told us that GAs would not like big alphabets, some users ignored that warning and got good results, and further inquiry has suggested that the GA sidesteps the problem by turning big alphabets into little alphabets. Anyone who has ever played with genetic algorithms can tell similar anecdotes where selectionist schemes were quite good at (and sometimes quite surprising in) exploiting any opening in their quest for improvement; it shouldn't come as much surprise that GAs regularly take computational lemons and compute lemonade.

Of course, I am not arguing here for resting on our laurels, and this paper has only begun to scratch the surface of real-coded and high-cardinality GAs. Further empirical and analytical study are needed along a number of lines:

1. Creating blocked test functions, and testing simple and advanced real-coded GAs.
2. Using slices to analyze different operators, such as averaging recombination and jump mutation.
3. Extending alphabet theory along variance or probability lines to define virtual alphabets in terms of characters that are achievable in populations of a particular size.
4. Developing and testing hybrid binary-real GAs.
5. Considering floating-point messy GAs and their relation to simple real-coded GAs.

The results of this paper have been largely theoretical. The theory needs to be tested through the creation of a suite of blocked problems, and simple operators and their variants should be tested to see if the bogeyman envisioned here is made of straw or stone. These studies are underway, and results should be available soon. Note that it is important in the design of blocked problems to make sure that global basins are NIAH (needle in a haystack). Otherwise, any hillclimber can solve the problem and little is learned about blocking.

Slice analysis should be applied to other operators more rigorously. This extension was discussed briefly herein, but a more formal treatment should permit the analysis of all known forms of averaging recombination and a variety of mutation operators.

The theory of virtual alphabets takes the mean slice as the dividing line between those intervals that are likely to have highly fit samples and those that are not. This theory can be extended usefully if the mean slice is replaced by a profile of fitness values that have a specified probability of being found at random. Such a profile can be calculated directly, or it can be estimated from mean and variance profiles together with some assumption regarding the distribution of function values.

Hybrid real-binary GAs should be developed and tested. The success enjoyed by Lucasius and Kateman [27] using both binary and creeping mutation might be carried over to the simultaneous use of a number of recombination operators. Some thought should be given to the function suite used in testing the hybrids, and it is likely that functions that are both binary-deceptive and real-blocked or real-deceptive can be generated. Perhaps the more interesting test cases are those where the function thwarts the algorithm in one space and not in the other, or in a manner such that part of the problem can be solved in one space and part of the problem can be solved in the other.

Messy floating-point GAs have been partially investigated [13]. This approach may be viewed as a hybrid, combining aspects of both real-coded and binary-coded GAs. Messy GAs use variable numbers of mantissa bits and exponent bits together with parameter-ending punctuation marks to adaptively place precision and emphasis where it is needed. More work is necessary to determine whether the appropriate combination of messy operators can overcome deception and blocking.

9. Conclusions

This paper has developed a theory of real-coded genetic algorithm operation called the theory of virtual alphabets. The theory suggests that selection reduces the continuum to a virtual alphabet along each dimension, where the virtual characters of that alphabet are selected from basinic features (above-average intervals attracted to a local optimum) of the one-dimensional mean slice of the function along each dimension. The theory reconciles an apparent paradox between simple schema theory and the empirical success enjoyed by users of real-coded GAs. Although simple schema theory is correct in saying that lower cardinality alphabets process higher numbers of schemata, this new theory suggests how selection itself reduces high-cardinality actual alphabets to low-cardinality virtual alphabets quite quickly, the alphabets thereafter undergoing processing through the action of recombination and other genetic operators. Although the theory provides a plausible mechanism of convergence consistent with the theory of schemata, it also predicts that problems exist that effectively block real-coded GAs from finding global optima, and examples of simple blocking have been given. Although the theory was developed for simple real-coded GAs, straightforward suggestions for alleviating blocking through averaging recombination or jump mutation appear to be of limited utility; the basic argument against their usefulness has been outlined and the route to more detailed analysis has been sketched. Early computational results tend to confirm this theory and its predictions, but more work needs to be done. Nonetheless, the results of this paper are on firm enough ground that users of real-coded GAs would be wise to turn to the development and investigation of other operators that can circumvent the impediment of blocking, if real-coded GAs are not to be permanently limited to the relatively small and simple class of unblocked problems.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants CTS-8451610 and ECS-9022007. I thank Dave Davis for provoking the thinking that went into this paper during a late night discussion (read argument) at the 1989 NIPS workshops. Erik Winfree's thoughtful review was instrumental in sharpening the focus and the writing of the final manuscript. Anthony Hester assisted in the preparation of computer codes and graphs at the University of Alabama, and Kelsey Milman prepared a number of graphs at the University of Illinois.

References

[1] A. D. Bethke, "Genetic Algorithms as Function Optimizers" (Doctoral dissertation, University of Michigan) *Dissertation Abstracts International*, **41**(9) (1981) 3503B (University Microfilms No. 8106101).

[2] W. W. Bledsoe, "The Use of Biological Concepts in the Analytical Study of Systems," paper presented at the ORSA-TIMS National Meeting, San Francisco, CA (1961).

[3] J. Bosworth, N. Foo, and B. P. Zeigler, *Comparison of Genetic Algorithms with Conjugate Gradient Methods*, CR-2093 (Washington, DC, National Aeronautics and Space Administration, 1972).

[4] G. E. P. Box, "Evolutionary Operation: A Method for Increasing Industrial Productivity," *Journal of the Royal Statistical Society C*, **6**(2) (1957) 81–101.

[5] H. J. Bremermann, "Optimization through Evolution and Recombination," in *Self-Organizing Systems*, edited by M. C. Yovits, G. T. Jacobi, and G. D. Goldstein (Washington, DC, Spartan, 1962).

[6] H. J. Bremermann, "Numerical Optimization Procedures Derived from Biological Evolution Processes," in *Cybernetic Problems in Bionics*, edited by H. L. Oestreicher and D. R. Moore (New York, Gordon and Breach, 1968).

[7] H. J. Bremermann and M. Rogson, "An Evolution-Type Search Method for Convex Sets," Technical report, Contracts NONR 3656(08), NONR 222(85) (Berkeley, University of California, Department of Mathematics, 1964).

[8] H. J. Bremermann, M. Rogson, and S. Salaff, "Global Properties of Evolution Processes," in *Natural Automata and Useful Simulations*, edited by H. H. Pattee, E. A. Edelsack, L. Fein, and A. B. Callahan (Washington, DC, Spartan, 1966).

[9] R. A. Caruana and J. D. Schaffer, "Representation and Hidden Bias: Gray versus Binary Coding for Genetic Algorithms," *Proceedings of the Fifth International Conference on Machine Learning* (1988) 153–162.

[10] S. Coombs and L. Davis, "Genetic Algorithms and Communication Link Speed Design: Constraints and Operators," *Proceedings of the Second International Conference on Genetic Algorithms* (1987) 257–260.

[11] L. Davis and S. Coombs, "Genetic Algorithms and Communication Link Speed Design: Theoretical Considerations," *Proceedings of the Second International Conference on Genetic Algorithms* (1987) 252–256.

[12] L. Davis, personal communication, November 30, 1989.

[13] K. Deb, "Binary and Floating-Point Function Optimization using Messy Genetic Algorithms" (Doctoral dissertation, University of Alabama, 1991; available from The Clearinghouse for Genetic Algorithms, Department of Engineering Mechanics, University of Alabama, Tuscaloosa, AL 35487).

[14] N. Y. Foo and J. L. Bosworth, *Algebraic, Geometric, and Stochastic Aspects of Genetic Operators*, CR-2099 (Washington, DC, National Aeronautics and Space Administration, 1972).

[15] G. J. Friedman, "Digital Simulation of an Evolutionary Process," *General Systems Yearbook*, **4** (1959) 171–184.

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

[17] D. E. Goldberg, "Genetic Algorithms and Walsh Functions: Part I, A Gentle Introduction," *Complex Systems*, **3** (1989) 129–152.

[18] D. E. Goldberg, "Genetic Algorithms and Walsh Functions: Part II, Deception and Its Analysis," *Complex Systems*, **3** (1989) 153–171.

[19] D. E. Goldberg, "Sizing Populations for Serial and Parallel Genetic Algorithms," *Proceedings of the Third International Conference on Genetic Algorithms* (1989) 70–79.

[20] D. E. Goldberg, "Zen and the Art of Genetic Algorithms," *Proceedings of the Third International Conference on Genetic Algorithms* (1989) 80–85.

[21] D. E. Goldberg, and K. Deb, "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms," *Proceedings of a Workshop on the Foundations of Genetic Algorithms* (forthcoming).

[22] D. E. Goldberg, K. Deb, and B. Korb, "Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale," *Complex Systems*, **4** (1990) 415–444.

[23] D. E. Goldberg, B. Korb, and K. Deb, "Messy Genetic Algorithms: Motivation, Analysis, and First Results," *Complex Systems*, **3** (1989) 493–530.

[24] J. H. Holland, "Outline for a Logical Theory of Adaptive Systems," *Journal of the Association for Computing Machinery*, **3** (1962) 297–314.

[25] J. H. Holland, "Hierarchical Descriptions of Universal Spaces and Adaptive Systems," Technical report ORA Projects 01252 and 08226 (Ann Arbor, University of Michigan, Department of Computer and Communication Sciences, 1968).

[26] J. H. Holland, *Adaptation in Natural and Artificial Systems* (Ann Arbor, The University of Michigan Press, 1975).

[27] C. B. Lucasius and G. Kateman, "Application of Genetic Algorithms in Chemometrics," *Proceedings of the Third International Conference on Genetic Algorithms* (1989) 170–176.

[28] D. J. Montana and L. Davis, "Training Feedforward Neural Networks Using Genetic Algorithms," *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence* (1989) 762–767.

[29] I. Rechenberg, "Cybernetic Solution Path of an Experiment Problem," Royal Aircraft Establishment translation no. 1122, translated by B. F. Toms (Farnborough Hants, Ministry of Aviation, Royal Aircraft Establishment, 1965).

[30] I. Rechenberg, "The Evolution Strategy: A Mathematical Model of Darwinian Evolution," pages 122–132 in *Synergetics: From Microscopic to Macroscopic Order*, edited by E. Frehland (Berlin, Springer Verlag, 1984).

[31] I. Rechenberg, *Literaturnachweis zur Evolutionsstrategie* [Bibliography for the evolution strategy] (Unpublished manuscript, Technische Univesität Berlin, Fachgebiet Bionik und Evolutionstechnik, Berlin, 1986).

[32] H. Schwefel, *Numerical Optimization of Computer Models*, translated by M. Finnis (Chichester, John Wiley, 1981; original work published in German, 1977.)

[33] O. J. Selfridge, "Pandemonium: A Paradigm for Learning," *Proceedings of the Symposium on the Mechanization of Thought Processes* (1959) 511–529.

[34] R. Weinberg, "Computer Simulation of a Living Cell" (Doctoral dissertation, University of Michigan), *Dissertations Abstracts International*, **31**(9) (1970) 5312B (University Microfilms No. 71-4766).

[35] A. H. Wright, "Genetic Algorithms for Real Parameter Optimization, *Proceedings of a Workshop on the Foundations of Genetic Algorithms* (forthcoming).

[36] B. P. Zeigler, J. L. Bosworth, and A. D. Bethke, "Noisy Function Optimization by Genetic Algorithms," Technical report no. 143 (Ann Arbor, University of Michigan, Department of Computer and Communication Sciences, 1973).