Building Blocks, Cohort Genetic Algorithms, and Hyperplane-Defined Functions

John H. Holland
Professor of Psychology
Professor of Computer Science and Engineering
The University of Michigan
Ann Arbor, MI 48109, USA
and
External Professor
Santa Fe Institute
Santa Fe, NM 87501, USA

Abstract

Building blocks are a ubiquitous feature at all levels of human understanding, from perception through science and innovation. Genetic algorithms are designed to exploit this prevalence. A new, more robust class of genetic algorithms, cohort genetic algorithms (cGA's), provides substantial advantages in exploring search spaces for building blocks while exploiting building blocks already found. To test these capabilities, a new, general class of test functions, the hyperplane-defined functions (hdf's), has been designed. Hdf's offer the means of tracing the origin of each advance in performance; at the same time hdf's are resistant to reverse engineering, so that algorithms cannot be designed to take advantage of the characteristics of particular examples.

Keywords

Building blocks, chromosome-like strings, crossover, fitness, genetic algorithms, schema, search spaces, robustness, selection, test functions.

1 Introduction

The "building block thesis" holds that most of what we know about the world pivots on descriptions and mechanisms constructed from elementary building blocks. My first comments on this thesis, though it did not yet have that name, were published at the 1960 Western Joint Computer Conference[1] (WJCC) as part of a special session on "The design, programming, and sociological implications of microelectronics" (Holland, 1960). The paper ended by tying together topics that play a key role in the thesis: implicit definition of structure, connected generators, hierarchical definition, adaptation, autocatalytic systems, Art Samuel's checkersplayer, and implicit definition of the problem space (as contrasted to a state-by-state definition). In both the session and the published proceedings, my paper was followed by Al Newell's wonderful riff on the framework I'd presented, a paper still exciting to read after all these years. The upshot was an avid desire, on my part, to look further into algorithmic approaches to adaptation.

[1] As an aside, the WJCC was a remnant of an east-coast/west-coast division in the computing community, a rift that had largely healed by 1960, but similar at its height to the current division in the EC community. None of the exploration reported there would likely be considered "computer science" under today's hardened definitions, but far-horizon exploration was encouraged in those days.

Level (science)	Typical Mechanisms
Nucleus (physics)	quarks, gluons
Atom (physics)	protons, neutrons, electrons
Gas & Fluid (physics)	
confined (e.g., a boiler)	PVT laws, flows
free (e.g., weather)	circulation (e.g. fronts), turbulence
Molecule (chemistry)	bonds, active sites, mass action
Organelle (microbiology)	enzymes, membranes, transport
Cell (biology)	mitosis, meiosis, genetic operators
Organism (biology)	growth factors, apoptosis, other morphogenetic operators
Ecosystem (ecology)	predation, symbiosis, mimicry

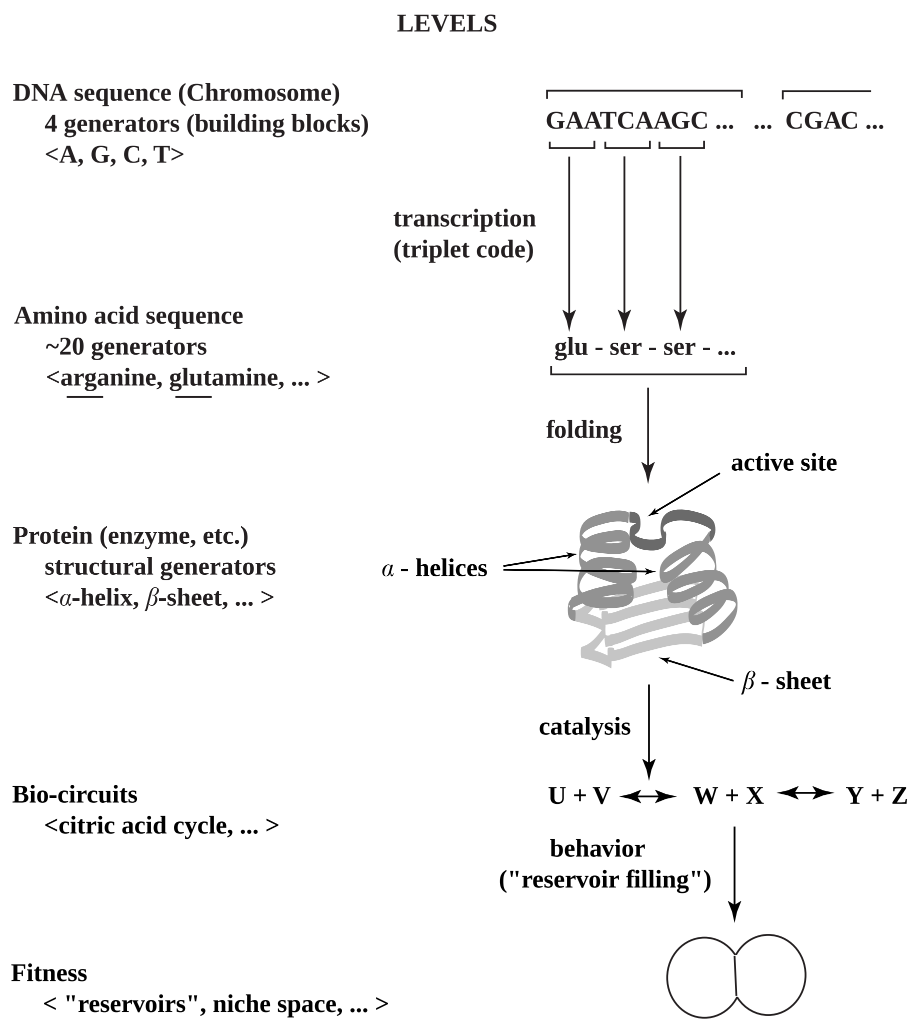
Building blocks from lower levels provide constraints and suggest what to look for at higher levels. Proposed additions at any level must be consistent with observations at all levels.

Figure 1: A typical set of interlocked levels of investigation in science.

The building block thesis has been validated repeatedly in the scientific advances of the 40 years since that WJCC. Each major advance over this period, as was also the case in earlier years, depended on the discovery and recombination of well-chosen building blocks. Section 2 of this paper presents enough old and new examples to make this claim plausible. Once a computer scientist starts thinking about building blocks as a source of innovation, the next obvious step is to look for algorithms that can discover and exploit building blocks. It is my claim that genetic algorithms (GA's) are particularly qualified for this task. Sections 3 and 4 present a part of that argument, while Section 5 outlines some of the obstacles that stand in the way of unfettered use of GA's for this purpose. Section 6 discusses a new class of GA's, called cohort GA's (cGA's), designed to overcome these obstacles, while Section 7 outlines a broad class of test functions, the hyperplane defined functions (hdf's), that allow one to trace in detail just how the obstacles are overcome. Section 8 discusses cGA/hdf interaction, and Section 9 makes some observations about this interaction. The paper closes with a look to the future.

2 Building Blocks

The successive levels of building blocks used in physics are familiar to anyone interested in science — nucleons constructed from quarks, nuclei constructed from nucleons, atoms constructed from nuclei, molecules constructed from atoms, and so on (see Figure 1). Nowadays a similar succession presents itself in daily newspaper articles discussing progress in biology: chromosomal DNA constructed from 4 nucleotide building blocks; the basic structural components of enzymes: alpha helices, beta sheets, and the like, constructed from 20 amino acids; standard "signaling" proteins for turning genes "on" and "off," and "autocatalytic bio-circuits," such as the citric acid cycle, that perform similar functions over extraordinarily wide ranges of species; organelles constructed from situated bio-circuits, and so on (see Figure 2). And, of course, there are the long-standing taxonomic categories: species, genus, family, etc., specified in terms of morphological and chromosomal building blocks held in common. However, the pervasiveness of building blocks only becomes



A metazoan typically has dozens of alleles that distinguish it from other organisms of its species. These differences are *not* fitness-neutral.

Most recombinations yield a change in fitness.

[Emergence conjecture: Neutrality decreases with each successive level.]

Figure 2: Levels and building blocks in biological systems.

apparent when we start looking at other areas of human endeavor. In some cases we take building blocks so much for granted that we're not even aware of them. Human perception is a case in point. The objects we recognize in the world are always defined in terms of elementary, reusable building blocks, be they trees (leaves, branches, trunks,...), horses (legs, body, neck, head, blunt teeth,...), speech (a limited set of basic sounds called phonemes), or written language (the 26 letters of English, for example).

In other cases we just don't make the building blocks explicit. Consider two major inventions of the 20th century, the internal combustion engine and the electronic computer. The building blocks of the internal combustion engine: gears, Venturi's aspirator, Galvani's sparking device, and so on, were well-known prior to the invention. The invention consisted in combining them in a new way. Similarly, the components of early electronic programmable computers: wires, Geiger's counting device, cathode ray tubes, and the like, were well-known. Even earlier, Babbage had spelled out an overall architecture using long-standard, mechanical building blocks (gears, ratchets, levers, etc.). The later invention consisted in combining the electronic building blocks in a way that implemented Babbage's mechanical layout. And, of course, building blocks underpin the critical step for universal computation: arbitrary algorithms are constructed by combining copies of a small set of basic instructions. For both the internal combustion engine and the programmable computer, the building blocks were a necessary precursor, but the innovation required a new combination of the blocks. That is a theme to which we'll return.

There are two main characteristics of building blocks: (i) they must be easy to identify (once they've been discovered or picked out), and (ii) they must be readily recombined to form a wide variety of structures (much as can be done with children's building blocks). Descriptions so-constructed will only be useful if there are many, frequently encountered objects that can be so described. For actual constructions, as contrasted to descriptions, building blocks will only be useful if there are many objects (useful and not so useful) that can be constructed from the blocks.

3 Toward Genetic Algorithms

Building blocks play a central role both in Darwin's original formulation of natural selection and in the neo-Darwinian synthesis. This role becomes quite clear when we consider Darwin's strong emphasis on the relation between artificial selection (breeding) and natural selection:

Artificial selection depends upon mating animals or plants that have desirable characteristics. That is, the breeder mates individuals that have different, recognizable building blocks (be it long legs or an unusual color) that contribute to capabilities the breeder desires. After mating, the breeder artificially selects the best of the offspring, those combining more of the desired characters in a single individual, for further breeding. The common term for this kind of breeding is "crossing." We now know that, at the chromosomal level, "crossover" (crossing over of the relevant chromosomes) implements "crossing," although artificial selection was practiced for millennia prior to this understanding. Natural selection operates in a similar fashion, but now the favored characteristics are those leading to the acquisition of resources that can be turned into offspring.

[It used to be a common assumption that, at the level of the genome, building blocks are specified by individual genes having cumulative additive effects. Much of the early seminal work, including Fisher's (1930) fundamental mathematical development, takes this assumption as a starting point. However, it is now amply clear from work in genomics and proteonomics, that (i) linked groups of genes can serve as building blocks under crossover and (ii) the interactions they encode can be highly nonlinear. See for example, Schwab and Pienta (1997), Hutter et al. (2000), Walter et al. (2000), Lander and Weinberg (2000), and Rubin et al. (2000).

Typically, in both artificial and natural selection, there are large sets of possibilities to be explored. These sets are often called *search spaces*. In the simplest form, each element (possibility) in a search space has an assigned value called its *fitness*. In artificial selection, this fitness measures the extent to which an element (individual) has the characteristics desired by the breeder. In natural selection, the fitness measures the ability of the corresponding individual to find and convert resources into offspring.[2]

In exploring this search space, the usual objective is not so much one of finding the best individual, as it is to locate a succession of improvements. The search is for individuals of ever-higher fitness. Such a search could be implemented as a uniform random sampling, and there are search spaces for which it can be shown that there is no faster search technique. However, this is not a good way to explore a search space that has relevant regularities. If the regularities can be exploited to guide the search, search time can be reduced by orders of magnitude. Building blocks supply regularities that can be exploited.

It is clear that both artificial and natural selection can locate individuals of increasing fitness in quite complex search spaces. Current work in molecular biology makes it increasingly clear that these "solutions" depend on new combinations of extant building blocks, along with the occasional discovery of a new building block. We only need to look again to Darwin to see that these ideas played a prominent role in his treatises. For example, Darwin's discussion of the origin of complex adaptations, such as the eye, pivots on the accumulation and exploitation of appropriately simple components that were already valuable in other contexts.

4 Designing Genetic Algorithms

It is not much of a leap, then, to think of adopting a Darwinian approach to exploring search spaces that abound in building blocks. Indeed, given the pervasiveness of building blocks in the actual and figurative devices we use to understand the world, this would seem a productive way to approach many of the world's complexities. The question then becomes:

How do we turn Darwin's ideas into algorithms?

With our knowledge of chromosomes as the carriers of heredity, it seems reasonable to start by encoding individuals in the search space as chromosome-like strings. Binary encodings work, though more complex encodings have been used. Once we have a string representation it is then natural to want to relate building blocks to pieces of strings. This is now familiar ground, though it took some thought at the outset.

Given a set S containing all objects of interest, it is common in mathematics to identify a property of an object with the subset of all objects that have the property. There is a set of easily defined subsets of S that makes a good starting point. Treat each string position (locus) as a dimension of the space S and then consider the hyperplanes in S. For the set

[2] I will not discuss here the shortcomings of such one-dimensional measures, nor will I discuss the advantages of an *implicit* definition of fitness over this explicit definition.

S of binary strings of length n, $\{1,0\}^n$, the set of all hyperplanes H can be identified with the set $\{1,0,\#\}^n$. The string $s \in S$ belongs to hyperplane $h \in H$ if and only if each of the non-# bits in the string h matches the corresponding bit in the string s. That is, the strings h and s match at all positions where there is *not* a #. Accordingly, the string $h = 1\#0\#\cdots\#$ specifies the hyperplane consisting of all strings in S that start with a 1 and have a 0 at the third position. These hyperplanes are called *schemata* in papers dealing with genetic algorithms.

From the point of view of genetics, all the individuals belonging to a given hyperplane hold certain alleles in common. If this set of alleles is correlated with some increment in fitness, the set constitutes a useful building block called a *coadapted set of alleles*. If we were talking of cars instead chromosomes, we might be considering the advantages of a coadapted set of components called a fuel injection system.

How can an algorithm discover and exploit hyperplanes corresponding to coadapted sets of alleles? This is not the place to review the standard implementations of genetic algorithms. Most reading this paper will be familiar with GA's; for those not so informed, Melanie Mitchell's (1996) book is an excellent introduction. Suffice it to say that a typical GA operates on a population (set) B of n strings by iterating a three-step procedure:

1. Evaluate the fitness $f(s)$ in the search space of each $s \in B$.
2. Select $n/2$ pairs of strings from B, biased so that strings with higher fitness are more likely to be chosen.
3. Apply genetic operators, typically crossover and mutation, to the pairs to generate $n/2$ new pairs, replacing the strings in B with the n new strings so created; return to step (1).

It takes a theorem, a generalization of Fisher's famous Fundamental Theorem, to show that this simple procedure does interesting things to building blocks. Let $\hat{f}_h$ be the *average observed fitness* of the strings in B (if any) that belong to schema h, and let $\hat{f}$ be the average fitness of all the strings in B. Further, let the bias for selecting a given string in step (2) above be $\hat{f}_h/\hat{f}$. Then it can be shown that the expected proportion of instances of h in the population formed in step (3) is bounded below by

$$(\hat{f}_h/\hat{f})(1 - e_h)P_h$$

where P_h is the proportion of strings in population B belonging to h, and the "error function" e_h measures the destruction of instances of h by the genetic operators. e_h is an invariant property of h — for example, e_h increases with the distance between the endpoints of the defining loci for the schema h. This theorem is called the *schema theorem* (Holland, 1992).

In intuitive terms, this theorem says that the number of instances of a building block increases (decreases) when the *observed* average fitness in the current generation B is above (below) the *observed* population average. The theorem holds for all schemata represented in the population. In typical cases, e_h is small for "short" schemata (say, schemata for which the distance between endpoints is less than a tenth the length of the string). For such schemata the error term can be ignored so that the number of instances of the schema in the next generation increases (decreases) when $\hat{f}_h/\hat{f} > 1$ ($\hat{f}_h/\hat{f} < 1$). For realistic values of population size n and strings of any considerable length, say length > 100, it is easy to show that the number of schemata so processed vastly exceeds n. That is, the number of schemata manipulated as if the statistic $\hat{f}_h/\hat{f}$ had been calculated is much larger than the number of strings explicitly processed. This is called *implicit parallelism* (Holland, 1992).

[In recent years, there have been glib, sometimes blatantly incorrect, statements about the schema theorem. One widely distributed quote claimed that the theorem was "either a tautology or incorrect." It is neither. It has been carefully reproved in the context of mathematical genetics (Christiansen and Feldman, 1998), and it is no more a tautology than Fisher's theorem. Quite recently a book purporting to provide the mathematical foundations for the "simple genetic algorithm (SGA)" says "... the 'schema theorem' explains virtually nothing about SGA behavior" (Vose, 1999, xi). Even explanations as brief as the one above make it clear that the theorem provides insight into the generation-to-generation dynamics of coadapted sets of alleles, as does Fisher's theorem with respect to individual alleles. That same book offers, as its own explanation of dynamics, stochastic vectors wherein, for small populations and short strings, the *largest* entry — the most probable population — can be smaller than the reciprocal of the number of atoms in the universe! Standard mathematical approaches, such as Markov processes and statistical mechanics, typically offer little insight into the dynamics of processes that involve the nonlinearities inherent in coadaptations.]

It is clear from the schema theorem that a GA does exploit building blocks on a generation-by-generation basis. Crossover shows up in the schema theorem only as an impediment (usually slight) that increases e_h. What purpose does crossover serve then? As is well-known to most practitioners, it serves two purposes: First of all, it recombines building blocks, bringing blocks residing on separate strings into combination on a single string. The second effect is more subtle. Crossover uncovers new building blocks by placing alleles in new contexts. Moreover, as we've seen in Section 2, building blocks are often combinations of more elementary building blocks, so recombination works at this level, too.

It's worth a moment to put these observations in the broader context of biology. It is well-known that the crossover rate in mammals is about 6 orders of magnitude greater than the point mutation rate. Four main explanations have been given for the prevalence of crossover:

i. Crossover provides long (random) jumps in the space of possibilities, thus providing a way off of local maxima.

ii. Crossover repairs mutational damage by sequestering deleterious mutations in some offspring while leaving other offspring free of them.

iii. Crossover provides persistent variation that enables organisms to escape adaptive targeting by viruses, bacteria, and parasites.

iv. Crossover recombines building blocks.

I've listed these explanations in what I consider to be their order of (increasing) importance. All four explanations have some basis in experiment and observation but, given the overwhelming use of building blocks at all biological levels, it would seem that (iv) is sufficient in itself to account for the prevalence of crossover.

5 Robustness

Are GA's, then, a robust approach to all problems in which building blocks play a key role? By no means! After years of investigation we still have only limited information about the GA's capabilities for exploiting building blocks. In an attempt to learn more of the GA's capabilities along these lines, several of us defined a class of building-block-based functions some years ago. We called them Royal Road (RR) functions (Mitchell et al., 1992) because improvement in the RR domain depended entirely on the discovery and exploitation of building blocks. We even thought that a GA would easily "outrun" competitors on this royal road.

We were wrong on several counts. First of all, in trying to make the functions simple, we made them too simple. The RR functions are convex, thus constituting a single "hill" to be climbed. There are gradient algorithms that, once positioned at the base of a hill, can double the accuracy of the "hilltop" coordinates with every iteration. No GA can move that fast. Worse yet, there are mutation-only algorithms that perform quite competitively in convex domains. *Lesson 1: If you want test functions that exhibit the GA's strength, don't use convex functions.*

There was a second flaw in the particular RR functions we used in our tests. They were too shallow. In the RR functions, building blocks at each level are formed from two adjacent building blocks at the previous level. As a result, building blocks at the next-to-highest level occupy half the chromosome, and they have a 50-50 probability of being "destroyed" each time crossover occurs. Because we used four-level functions, the GA had substantial difficulty getting beyond the first two levels. We call this "the saturation effect." Because the GA's greatest strengths lie in finding improvements, not in optimization, shallow functions are not a good test of performance. *Lesson 2: If you want test functions that exhibit the GA's strengths, use "deep" functions.*

There was a third, more subtle difficulty, only partially attributable to the RR functions, though they highlighted the problem. It is a problem called "hitchhiking," a phenomenon well-known in population genetics, but previously little studied in the context of GA's. Hitchhiking occurs when some newly discovered allele (or coadapted set of alleles) offers great fitness advantages (say, DDT resistance for an insect species immersed in a DDT-rich environment). Then, as that allele spreads rapidly through the population, so do nearby, closely linked alleles (though they may make no contribution to the fitness). The net result is a greatly reduced exploration of alternatives at the loci involved in hitchhiking.

The hitchhiking problem presents typical GA's with a dilemma. One side of the dilemma occurs when the scaled maximum reproduction rate is set high to provide rapid increase in above-average schemata. Then the defining bits of the schemata in the best string(s) in the population, *along with the nearby bits*, quickly come to occupy most of the population. That is, the nearby bits "hitchhike" to a prominence in the population, leaving few variants at those loci. This loss of diversity in the vicinity of the better schemata greatly impedes the search for further improvements. The effect is particularly damaging in a smaller population. Consider a population of 1000 individuals and a newly discovered individual with a scaled fitness of 2 (i.e., the individual produces 2 offspring for the next generation). If successive progeny of that individual retain a similarly high scaled fitness, the schemata in that individual, and nearby bits, will appear in almost every individual in the population in 10 generations. In such a short time, genetic operators like mutation cannot introduce enough differences into the population to counteract this rapid loss of diversity.

The other side of the dilemma arises if the reproduction rate for the best string is scaled downward to 1.2 or less, to give the GA time to counteract hitchhiking. Then we encounter difficulties because of the way in which typical GA's handle "fractional offspring." Almost all GA's use some variant of a stochastic approach to deal with the fractional part of the fitness: For fitness 1.2, for example, a second offspring is produced with probability 0.2. Under this approach, the *expected* number of offspring for this individual, after several generations, is correct. We would *expect* to find 2 copies of an individual of fitness 1.2 after 4 generations ($1.2^4 = 2$). However, the variance associated with this expectation is large; the probability of only *one* copy after 4 generations is 0.4. Valuable schemata are likely to be lost under such circumstances.

This dilemma motivates a new class of genetic algorithms which I will discuss in the next section. The two earlier problems, convexity and shallowness, motivate a new class of test functions which I will discuss in Section 7.

6 Cohort Genetic Algorithms

Cohort genetic algorithms (cGA's) are designed to allow low maximal reproduction rates without suffering the high variance induced by a stochastic approach to fractional offspring. The central idea for the cGA is that a string's fitness determines how long it will have to wait before it produces offspring. A string of high fitness produces offspring quickly, while a string of low fitness may have to wait a long time before reproducing. With this arrangement, all strings can have the same number of offspring, say two, at the time they reproduce. The difference lies in the interval that must elapse before the two offspring are produced. Clearly, a highly fit string with building blocks that yield highly fit offspring will, over multi-generational time T, have many more progeny than a string of lower fitness. A little calculation shows that the variance of this method of reproduction is *much* less than the variance under the probabilistic approach to fractional offspring.

To implement this delayed-reproduction idea, the population of the cGA is divided into an ordered set of non-overlapping subpopulations called *cohorts*. Reproduction is carried out by cycling through the cohorts in the given order, determining the offspring for individuals in cohort 1, then cohort 2, and so on. When the last cohort is reached, the procedure returns to cohort 1 and is repeated.

Reproduction in cohort i, in a simple cGA with N cohorts, proceeds at follows:

1. Each string in cohort i produces 2 offspring.

2. The fitness u of each offspring is determined and is scaled using a function $f(u)$, where $f(u) = 1.2$, say, for the maximum fitness so far observed, and $f(u) = 1$ for the current average fitness.

3. The doubling time $d(u)$ corresponding to fitness $f(u)$ is used to place each offspring in cohort $(i + d(u)) \bmod N$. [The doubling time for $f(u) < 1.2$ is roughly $d(u) = .8/(f(u) - 1)$. When $(f(u) - 1) < .25$, say, it is pragmatically useful to use a linear function of $f(u) - 1$, taking arguments $.25 > x > -1$, to distribute the corresponding offspring over the cohorts that lie between 16 and N steps "ahead" of the current cohort. Note that this provision assigns cohorts to offspring that are below average, whereas strict use of doubling time would put them in an infinitely distant cohort.]

There are several details that must be handled carefully in implementing the cGA, but overall the procedure is simple in concept and execution. A few observations are in order:

- Even if all of the cohorts are assigned the same initial size, they will soon be of different sizes because of the different numbers of offspring assigned to them.
- When reproduction involves two parents, as it does in a simple cGA, the paired parents must produce 4 offspring (two for each).
- If the overall population is to be bounded, say by keeping its size constant, then strings must be deleted from the population to make room for offspring. Even if the two parents are deleted when they reproduce, two additional individuals must be deleted to make room for 4 offspring.
- It takes some care to keep the deletions from impoverishing "distant" cohorts; otherwise there is a "piling up" in nearby cohorts, negating the diversity provided by having individuals in all N cohorts.

Part of the reason for the pragmatic linear function of step (3) of the reproduction routine is to alleviate "piling up." Under this arrangement, strings of lowest fitness are assigned to a cohort at maximal distance, and strings of below average fitness are distributed in cohorts that are beyond $N/2$ cohorts ahead of the current cohort. It is also helpful to delete (some) strings of below-average fitness at the time of formation.

As a final point, the very essence of good GA design is retention of diversity, furthering exploration, while exploiting building blocks already discovered. In the case of the cGA, it is particularly effective to lower the fitness of strings containing *alleles that are common* in the population: the more common alleles a string contains, the more its fitness is lowered from the fitness it would otherwise have. For example, the scaled fitness $f(u)$ is modified to $bf(u)$, where $b = 1-$(number of common alleles in string/string length). Occasional sampling will give reasonable estimates as to which alleles are common, so this procedure is not computationally intensive. (For those looking for a biological metaphor, the common alleles are the counterpart to characteristics that exploit the resources of an overcrowded niche). In a similar fashion, but with opposite intent, strings carrying rare alleles have their fitness incremented. Empirically, this technique works well. In conjunction with the lowered maximum fitness permitted by the cGA, it prevents the cGA from prematurely converging to a population consisting mostly of strings containing a few, early-discovered building blocks.

7 Hyperplane-Defined Functions

What we require now is a set of test functions that will let us see exactly what a cGA does vis-a-vis building blocks. From Section 5 it is clear the RR functions are inadequate for this purpose. For present purposes we need a set of test functions that are:

i. generated from elementary building blocks,

ii. nonlinear, nonseparable, and nonsymmetric (and, so, resistant to hillclimbing),

iii. scalable in difficulty, and

iv. in a canonical form.

These criteria are closely related to the test function guidelines set by Whitley et al. (1996). It would also be useful if the test functions:

v. can be generated at random and are difficult to reverse engineer (so that the algorithms being tested do not inadvertently or deliberately exploit incidental features, as frequently happens with the "bit-counting" functions, for example);

vi. exhibit an array of landscape-like features[3] ("hills," "badlands," "ridges," etc.) in controllable proportions (so that one can examine what parts of the "landscape" are exploited by different genetic operators);

vii. include all finite functions in the limit. [This last criterion recalls the presentation of analytic functions via Fourier coefficients. There are functions that have a simple representation (e.g., functions with simple periodicities) while other functions require a representation with large numbers of coefficients. All analytic functions are included, but there is an "ordering" in the simplicity of their representation.]

There is a metaphor that is useful in thinking about such functions. Think of a large, porous meteorite plunged into a high pressure bath of some easily traced fluid. In this metaphor, the meteorite's chambers correspond roughly to building-block-based inventions, and the differential porosity corresponds to the difficulty of getting from one set of inventions to another. Several aspects of a building-block-based search are easily understood in terms of the metaphor:

- Because any given chamber can be reached in a variety of ways, the search can proceed simultaneously along several paths. The expected time at which a chamber will first be occupied (the time to discovery) depends upon the cumulative probabilities of transiting these paths.
- Once a chamber is occupied, other chambers easily reached from that chamber become likely candidates for subsequent occupation. The corresponding dynamic process, then, consists of a succession of occupations, proceeding from chambers near the surface to chambers connected to those chambers by high porosity interfaces, and so on. A chamber with high porosity connections to a large number of other chambers will serve as a seed for many subsequent occupations (discoveries).
- The equilibrium (fixed point), at which all fluid particles are distributed uniformly throughout the meteorite, tells us little about the dynamic process (the sequence of inventions).

While the metaphor is suggestive for the discovery process, it is limited in several ways when the search is conducted by a GA. First among these limitations are:

- The metaphor only partially captures the combinatorial aspect of the search. While a combination of occupied chambers will enhance the probability of the occupation of a chamber jointly connected to them, this is not quite the same as combination of building blocks serving as a building block for subsequent larger combinations.

[3]A word of caution about "fitness landscapes": In a high-dimensional space, the features are themselves high-dimensional counterparts of the familiar features of two-dimensional landscapes. These features only become "visible" under a suitable coarse-graining of the overall space. Under such circumstances, intuition based on two-dimensional landscapes can be quite misleading.

- There is no direct counterpart of the *value* of an improvement, yet value plays a key role in both the fitness of a phenotype and the impact of an invention.
- There is no counterpart of the "increasing returns" aspect of GA's. To imitate this there would have to be a rapid, autocatalytic increase in the pressure in a chamber once it is occupied. This changes the dynamics of the search, strongly favoring chambers connected to chambers already occupied (discovered).

The *hyperplane-defined functions* (hdf) I'm about to describe follow the intuitions supplied by this metaphor, make up for its shortcomings, and meet the criteria set forth at the beginning of this section.

The search space X over which an hdf is defined is the set of all strings of length n. For simplicity, I'll consider only binary strings. Other alphabets could be used and the strings could be of variable length (up to some maximum length n). However, these additions only change the generality of the definition in trivial ways. The building blocks that underpin the definition of an hdf are represented as schemata (see Section 4). The object is to provide a class of fitness functions generated entirely by values assigned to a wide range of schemata having different lengths and defining bits. The value of each string (argument) in the function's domain, then, is determined by the schemata present in the string. Thus, an hdf function f is a real-valued function over the space X, where the value $f(x)$ represents the fitness of $x \in X$.

With each specific hyperplane-defined function f there is an associated set of schemata B used to determine $f(x)$. Each schema $b \in B$ is assigned a fundamental value $u(b)$ that can be either positive or negative. In the simplest case, the value of an x under f is simply the sum of the values of all the building blocks in x. More carefully,

$$f(x) = \max\{0, \sum_{x \in b | b \in B} u(b)\}.$$

It is easy to combine the values of the schemata in more complicated, nonlinear ways, but this simple version serves present purposes. By using an apropos random number generator, the set B can be picked with a given distribution over the set H of all schemata, and the values $u(b)$ can be similarly assigned with a predefined distribution. Under this arrangement, the hdf functions can be used as an infinite set of challenge functions: easy to generate, hard to reverse-engineer, and easy to analyze after the fact.

The hdf's of most interest for testing cGA's (and other GA's) satisfy the following additional constraints:

1. A set E of *elementary* schemata is chosen. These schemata are short relative to string length n, say a length (distance between the outside defining loci) $< n/10$, but long enough to be rare in a randomly generated population, say 8 or more defining loci. The schemata in E may overlap and they may be incompatible (i.e., they may require different defining bits at some locus they hold in common).
   [I have typically used hdf's with $n = 500$ and 15 randomly generated elementary schemata.]
2. Pairs of elementary schemata that are close to each other are selected at random and are combined in pairs to yield higher order schemata in B. These pairs are in turn combined in pairs to yield still higher order schemata, and so on until there are

schemata of a length close to n.
[In the hdf's I've been using, there are typically 50 or more of these higher order schemata.]

3. There are schemata in B defined by adding one or more additional defining bits to some elementary schema. These additional bits are chosen to match the locus and value of some nearby elementary schema. The values assigned to these new schemata are chosen to be less than the value of the elementary schemata involved in their definition. They constitute "valleys" that must be crossed to get from elementary schemata to higher order schemata.
[In the hdf's I've been using, every pair of adjacent elementary schemata has at least two associated "valley" schemata.]

4. Certain elementary schemata are chosen to be the generators of sets of schemata that are the high-dimensional analogues of "hills," "badlands" (highly irregular terrain), "ridges," and the like.
[These additions allow observation of the effects of genetic operators in relatively familiar contexts. I often use the hdf without them.]

To give one, overly simple example, consider an hdf where:

i. Each elementary schema e has value $u(e) = 2$;

ii. Each selected combination d of two elementary schemata, called a two-combinant, has value $u(d) = 3$, and all higher order combinants c have value $u(c) = 2$;

iii. Each elementary schema has two refinements, a forward refinement r and a backward refinement r'. The refinement r is defined by selecting one defining bit from the nearest non-overlapping elementary schema on the right (e.g., given e_i, add a bit from e_{i+1} to get r), while r' is similarly defined by selecting a defining bit from a schema on the left. The refinements r and r' have values $u(r) = -1$ and $u(r') = -1$.

Consider, then, a string x that is an instance of two adjacent, non-overlapping elementary schemata that form a two-combinant. Of necessity, for this particular hdf, each of the elementary schemata will also have a refinement present. So the total contribution to the value of x will be $2u(e) + u(r') + u(r) + u(d) = 5$. Note that, if only one of the elementary schemata were present, along with the refining bit belonging to the other schema, the value would be $(2) + (-1) = 1$, a kind of "pothole" on the road to the second schema.[4]

With the constraints in the previous paragraph we get functions that are nonconvex, with many overlapping sequences of improvements having common origins and enough levels to prevent early saturation effects. The resulting hdf is still a sum of weighted basis functions, as in Fourier or Walsh decompositions, so the algorithmic definition is simple, about 80 lines of Mathematica code. It is easy to use a random number generator to select candidate schemata and parameters, quickly providing an example hdf that is difficult to reverse engineer, while having the properties desired.

A typical hdf, then, is a search space in which elementary building blocks can be combined in a wide variety of ways. Most combinations offer no improvement, and many

[4] In more realistic hdf's we have two refinements in each direction, so that a "pothole" cannot be bypassed by adding it as the last correct bit in the sequence of bits defining the second schema.

are deleterious, giving a performance less than any of the constituent elementary building blocks. There are also some improvement sequences that wind up at dead ends, where no further additions of building blocks produce improvements. Some combinations serve as higher order building blocks, making possible a range of further improvements. In short, even though the elementary building blocks are easy to locate and large numbers of combinations are possible, the fruitful combinations are few. The search, then, is far from trivial and improvements are hard won. Because the building blocks that enter into an hdf definition are explicitly specified, it is easy to trace the dynamics of the discovery process. It is worth emphasizing again that there are representatives of every finite function in the class hdf functions, though the hdf's with *short* definitions have quite special characteristics.

8 The cGA in an hdf Search Space

Though all this work is still in an exploratory phase, there are already some interesting results to report. As the cGA and hdf were being developed, it was important to sample the cGA's rate of improvement in the hdf search space, comparing it to that of a good hillclimber.

For this purpose, I adapted Ted Belding's (1999) Lax Random Mutation Hillclimber (LRMHC) to the hdf. The LRMHC is a knowledge-based hillclimber that uses prior knowledge of the particular search space it faces. In the case of hdf's, the LRMHC is supplied with the maximum depth d of the "valleys" in the particular hdf it is searching. As with the usual hillclimber, it generates new sample points in the vicinity of the "current best observation" x^*. However, the LRMHC will replace x^* with a point y if the value $u(y) > u(x^*) - d$. This allows the LRMHC to wander through the valleys in the course of its search, instead of being required to go only to points of greater value, as would be the case with ordinary hillclimbers. The LRMHC provides a formidable opponent for the cGA, which uses no prior knowledge in its search.

As anecdotal evidence, I'll report the outcome of a some runs of a year ago (May 25, 1999). In these runs I used an hdf search space where a string of value in the 50-60 range is an instance of a schema of at least level[5] 6 or 7. The cGA had strings of length 400 and a population of size 400. In a typical run, the cGA attained in succession:

a string of value 27 at 12,575 evaluations,

a string of value 50 at 24,368 evaluations, and

a string of value 67 at 27,067 evaluations.

A second run with a different randomly generated initial population and a different random number seed yielded a string of value 59 after 25, 884 evaluations. The LRMHC, for comparison, found a best string of value 23 at 14,493 evaluations, with no further improvement through 40,000 evaluations. On a second run, with a different random number seed, the LRMHC located a string of value 36 at 33,019 evaluations, with no further improvement through 40,000 evaluations.

Of course, these are only sample runs and so do not constitute anything close to a statistical validation. Nevertheless, we see the cGA can exhibit sustained improvement and,

[5]A schema at level j is a compound schema that increments payoff by an amount more than the sum of the payoffs of the j elementary schema of which it is composed.

in these cases, the hdf does constitute a search space that discriminates between the cGA and a sophisticated hillclimber. It is particularly pleasing that the cGA performs so well when, unlike the LRMHC, it has no prior information about the hdf.

There was a later run (May 26, 1999) that revealed an interesting anomaly: There were some minor changes in the cGA program, but it was essentially the same as the program of the previous runs. However, after finding a string of value 36 at 18,825 evaluations, there was no further improvement through 64,000 evaluations. Investigation of the hdf used in that run revealed a pair of overlapping incompatible schemata, each of which was a starting point for a distinct sequence of improvements leading through higher levels. One of the sequences "dead-ended" at the string with value 36.

This is a clear example of the "founder" effect, an effect well known in population genetics. In biology, the founder effect arises when some species finds itself without competitors in a range of related niches. The finches of the Galapagos Islands and the fruit flies of the Island of Hawaii are classic examples. Variants of the originating species fill the unoccupied niches. In the present case, the first discovered of the two incompatible schemata comes to occupy a large part of the population. Because of the incompatibility, the other schema cannot appear on any string that contains the founder schema. As a result, the founder schema effectively blocks the testing of the other incompatible schema. Further improvements stem from the founder, making it progressively less likely that the other schema will influence further development. That is, the first-discovered schema founds a dynasty of improvements that effectively precludes the other dynasty. Subsequent runs have shown that, for the cGA, the founder effect is a much more important constraint on exploration than "hitchhiking." I'll discuss possible remedies in Section 10.

9 Observations

Certain interactions and observations seem obvious (though they would have to be verified in detail):

i. Long schemata that are not combinations of shorter schemata will be difficult to discover; they amount to 'spikes' in the hdf landscape.

ii. Schemata that are refinements (i.e., schemata having all the defining bits of some shorter schema along with a few additional defining bits) make gradual improvements possible. Any schema that has several distinct refinements has a multi-functional character, allowing it to initiate several lines of improvement.

iii. The waiting time until a given favorable schema h occupies a portion P_h of the population turns on (a) time to discovery of the schema, and (b) the rate of increase of schema once discovered. The time to discovery of h depends upon the prevalence of schemata (if any) that are building blocks for h, as well as the number of defining bits in h. The generation-to-generation rate of increase of h, once discovered, is given by the schema theorem.

iv. At any given time, the likely (short waiting time) discoveries of compound schemata will be based on schemata h with large P_h — the analog of the filled chambers in the meteorite.

v. Lines of improvement based on easily achieved combinants will be incorporated rapidly, becoming a common characteristic of the population. Difficult improvements (e.g., improvements involving intermediate refinements with lower values) will provide sudden changes in direction when discovered — the analog of difficult to reach chambers in the meteorite that are connected to many other chambers.

vi. The average observed fitness of a schema will increase as its instances come to include other positively valued schemata.

vii. Linkage plays a role in both the probability of discovery and the loss rate once a schema is discovered: Roughly, the probability of discovery of compound schema hh' at any given mating is $cP_hP_{h'}$, where c is the probability that a cross will fall between h and h'. The probability that an instance of h will be destroyed by crossover depends on the length of h. Note that the loss rate from crossover approaches 0 as P_h approaches 1.

viii. There will often be a 'race' between improvements provided by refinements vs. new combinations of schemata already present in the population.

ix. Incompatible schemata belonging to mutually exclusive lines of improvement amount to distinct 'niches' in the hdf environment. They amount to opportunities for speciation under selective mating (see next section).

10 Future Explorations

Let me start with the founder effect. It can be overcome in several ways, but one of the more interesting ways is via "speciation." To see the relevance of speciation, consider a cGA/hdf function combination (think of an ecosystem) subject to "carrying capacity" limitations for various schemata. Under such limitations, the value of a schema decreases as the number of strings carrying that schema increases. In effect, the schema defines a niche and, as the niche gets overcrowded, the per capita resources available for reproduction decrease. Speciation allows other schemata to avoid this "local" overcrowding by occupying other niches.

In more detail: Let the initial cGA population be large enough to contain several incompatible schemata. Add selective mating to the cGA by, for example, only allowing crossover between strings with similar "tag" regions. Under these conditions, a schema that acquires founder status will have the same initial effect as before, blocking the occurrence of other schemata that are incompatible with it. However, as this schema comes to occupy an increasing proportion of the population, it will have less and less advantage because of the carrying capacity limitation. In effect, it shares a limited resource over an increasing number of instances. That "leaves room" in the population for disjoint subpopulations carrying the other incompatible schemata. The net result, though this is yet to be tested carefully, should be an increased diversity (wider exploration) based on incompatible schemata. Note that, by using frames to designate tag regions on the string (as in the Echo models of *Hidden Order* (Holland, 1995)), it is possible to select for tags that relate to adaptive features of the string.

Even the limited observations so far suggest several interesting data-gathering experiments. Here are a few suggestive questions [along with my conjectures about the results]:

- What form does the "rate of improvement" curve take under various combinations of mutation and crossover probabilities? How well are discovered schemata retained? [Under high single-point crossover/low point-mutation probabilities, useful schemata should rarely be lost once discovered, and improvement should be rapid until the best schemata come to have lengths that are a substantial fraction of the string length.]

- Can the discovery process be described as a diffusion with increasing returns (an extension of the meteorite metaphor)? [Likely.]
- Does the cGA almost always operate near linkage equilibrium? [Unlikely.]
- Will 'species' form to fit incompatible 'niches' in the hdf under adaptive, frame-mediated selective mating (where the frame specifying part of the string is subject to the same genetic operations as the rest of the string) ? [Likely.]
- Geneticists have advanced four reasons to explain the fact that crossing over is several orders of magnitude more frequent than point mutation: (1) large steps in search space (as measured by a Hamming metric); (2) "repair" of mutational damage; (3) escape from predator targeting; (4) recombination of building blocks. Setting aside predator targeting (which requires a more dynamic environment), what are the relative proportions of these effects in cGA/hdf interactions? [Recombination effects are likely to be overwhelmingly important.]
- There is still the central conjecture to be verified in detail: Do most easily generated hdf's prove difficult for any variant of mutation-only search, or indeed for any method not making substantial use of the discovery and recombination of building blocks? [Likely.]

A final comment: In the early phases of research, selective probes guided by taste and intuition are more likely than data-gathering suites to yield advances. Collection of statistically valid results early on is likely to trap the researcher on a local peak, while probes can suggest more interesting alternatives involving substantial changes in the concept. Journals carry few exploratory papers, sometimes by design, but such papers help other researchers develop a sense of the field. Shades of the genetic algorithm at the meta-level!

Appendix: A Simple Example of a Hyperplane-Defined Function

Here is an example of an hdf that uses strings of length 50 and has only 5 elementary schemata and 4 levels. This example is highly simplified. A minimally interesting hdf for tests would (i) use strings with several hundred loci, (ii) have at least 15-20 elementary schemata with an average of 8 or more defining loci and some incompatible overlaps, (iii) use 2 or more "backward" and "forward" refinements ("potholes") for each elementary schema, and (iv) provide 8-10 levels of combinants with multiple ways of forming combinants at each level.

Elementary Schema (each has value $u(e) = 2$)

$e1$	`##################001#############################`
$e2$	`###################011############################`
$e3$	`#########################10#######################`
$e4$	`################################111100############`
$e5$	`#################################00###############`

Forward Refinements (each has value $u(r) = -1$)

$r1_{e1}$	`##################0011############################`
$r1_{e2}$	`###################011###1########################`
$r1_{e3}$	`#########################10#####1#################`
$r2_{e3}$	`#########################10#######1###############`
$r3_{e3}$	`#########################10#####1##1##############`

Backward Refinements (each has value $u(r') = -1$)

$r1'_{e2}$	`##################0011############################`
$r1'_{e3}$	`###################0#####10#######################`
$r2'_{e3}$	`####################1####10#######################`
$r3'_{e3}$	`###################01####10#######################`
$r1'_{e4}$	`#########################1######111100############`
$r1'_{e5}$	`################################100###############`
$r2'_{e5}$	`#################################00#0#############`
$r3'_{e5}$	`################################100#0#############`

Two-Combinants (each has value $u(d) = 3$)

$e1e3$	`##################001####10#######################`
$e2e3$	`###################011###10#######################`
$e3e4$	`#########################10#####111100############`
$e1e5$	`##################001############00###############`

Higher Order Combinants (each has value $u(c) = 3$)

$e1e3e4$	`##################001####10#####111100############`
$e2e3e4$	`###################011###10#####111100############`
$e1e2e3e5$	`##################0011###10######00###############`

Given the string

$$0010111011000110100100011\overset{e3}{\underline{10}}11011\overset{e4}{\underline{111100}}100110101100$$

which is an instance of the 2 elementary schemata $e3$ and $e4$, this hdf computes the value $2u(e) + 3u(r) + u(r') + u(d) = 3$

References

Belding, T. C. (1999). Potholes on the royal road. Unpublished manuscript, University of Michigan, Ann Arbor, Michigan.

Christiansen, F. B. and Feldman, M. W. (1998). Algorithms, genetics, and populations: The schemata theorem revisited. *Complexity*, 3(3):57–64.

Fisher, R. A. (1930). *The Genetical Theory of Natural Selection*. Clarendon Press, Oxford, England.

Holland, J. H. (1960). On Iterative Circuit Computers Constructed of Microelectronic Components and Systems. In *Proceedings of the 1960 Western Joint Computer Conference (WJCC) – Session on the Design, Programming, and Sociological Implications of Microelectronics*, National Joint Computer Committee, IEEE.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, Massachusetts.

Holland, J. H. (1995). *Hidden Order: How Adaptation Builds Complexity*. Addison-Wesley, Reading, Massachusetts.

Hutter, H., Vogel, B. E., Plenefisch, J. D., Norris, C. R. et. al. (2000). Conservation and novelty in the evolution of cell adhesion and extracellular matrix genes. *Science*, 287(5455):989–994.

Lander, E. S. and Weinberg, R. A. (2000). Journey to the center of biology. *Science*, 287(5459):1777–1782.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, Massachusetts.

Mitchell, M., Forrest, S., and Holland, J. H. (1992). The royal road for genetic algorithms: Fitness landscapes and GA performance. In Varela, F. J. and Bourgine, P., editors, *Proceedings of the First European Conference on Artificial Life*, pages 245–254, MIT Press, Cambridge, Massachusetts.

Rubin, G. M., Yandell, M. D., Wortman, J. R., Miklos, G. L. G., et al. (2000). Comparitive genomics of the eukaryotes. *Science*, 287:2204–2215.

Schwab, E. D. and Pienta, K. J. (1997). Modeling signal transduction in normal and cancer cells using complex adaptive systems. *Medical Hypotheses*, 48:111–123.

Vose, M. D. (1999). *The Simple Genetic Algorithm*. MIT Press, Cambridge, Massachusetts.

Walter, P., Keenan, R., and Schmitz, U. (2000). SRP - where the RNA and membrane worlds meet. *Science*, 287(5456):1212–1213.

Whitley, D., Rana, S., Dzubera, J., and Mathias, K. E. (1996). Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(1-2):245–276.