Relative Building-Block Fitness and the Building-Block Hypothesis

Stephanie Forrest
Dept. of Computer Science
University of New Mexico
Albuquerque, NM 87131

Melanie Mitchell[*]
AI Laboratory
University of Michigan
Ann Arbor, MI 48109

In D. Whitley (ed.) *Foundations of Genetic Algorithms 2*,
Morgan Kaufmann, San Mateo, CA, 1993.

Abstract

The *building-block* hypothesis states that the GA works well when short, low-order, highly-fit schemas recombine to form even more highly fit higher-order schemas. The ability to produce fitter and fitter partial solutions by combining building blocks is believed to be a primary source of the GA's search power, but the GA research community currently lacks precise and quantitative descriptions of how schema processing actually takes place during the typical evolution of a GA search. Another open problem is to characterize in detail the types of fitness landscapes for which crossover will be an effective operator. In this paper we first describe a class of fitness landscapes (the "Royal Road" functions) that we have designed to investigate these questions. We then present some unexpected experimental results concerning the GA's performance on simple instances of these landscapes, in which we vary the strength of reinforcement from "stepping stones"—fit intermediate-order schemas obtained by recombining fit low-order schemas. Finally, we compare the performance of the GA on these functions with that of three commonly used hill-climbing schemes,

[*]Current address: Santa Fe Institute, 1660 Old Pecos Trail, Suite A, Santa Fe, NM 87501

and find that one of them, “random-mutation hill-climbing”, significantly outperforms the GA on these functions.

1 INTRODUCTION

Research on the foundations of genetic algorithms aspires to answer two general questions: How do GAs work, and what are they good for? A successful theory of GAs would describe the laws governing the behavior of schemas in GAs and characterize the types of fitness landscapes on which the GA is likely to perform well, especially as compared with other search methods such as hill-climbing. This, of course, requires a statement of what it means for a GA to “perform well”. That is, we need a better understanding of what it is the GA is good at doing (e.g., finding a global optimum versus quickly finding a fairly good solution).

Our strategy for answering these questions consists of the following general approach. We begin by identifying *features* of fitness landscapes that are particularly relevant to the GA’s performance. A number of such features have been discussed in the GA literature, including local hills, “deserts”, deception, hierarchically structured building blocks, noise, and high fitness variance within schemas. We then design simplified landscapes containing different configurations of such features, varying, for example, the distribution, frequency, and size of different features in the landscape. We then study in detail the effects of these features on the GA’s behavior. A longer-term goal of this research is to develop statistical methods of classifying any given landscape in terms of our spectrum of hand-designed landscapes, thus being able to predict some aspects of the GA’s performance on the given landscape.

It should be noted that by stating this problem in terms of the GA’s performance on fitness landscapes, we are sidestepping the question of how a particular problem can best be represented to the GA. The success of the GA on a particular function is certainly related to how the function is “encoded” (Goldberg, 1989b; Liepins & Vose, 1990) (e.g., using Gray codes for numerical parameters can greatly enhance the performance of the GA on some problems), but since we are interested in biases that pertain directly to the GA, we will simply consider the landscape that the GA “sees.”

In this paper we describe some initial results from this long-term research program. We began by focusing on the *building-block hypothesis* (Holland, 1975; Goldberg, 1989b), which states that the GA works well when short, low-order, highly-fit schemas (“building blocks”) recombine to form even more highly fit higher-order schemas. In Goldberg’s words, “. . . we construct better and better strings from the best partial solutions of past samplings”(Goldberg, 1989b, p. 41). The ability to produce fitter and fitter partial solutions by combining building blocks is believed to be the primary source of the GA’s search power. However, in spite of the presumed central role of building blocks and recombination, the GA research community lacks precise and quantitative descriptions of how schemas interact and combine during the typical evolution of a GA search. Thus, we are interested in isolating landscape features implied by the building-block hypothesis, and studying in detail the GA's behavior—the way in which schemas are processed and building blocks are combined—on simple landscapes containing those features.

Other GA researchers have studied these same questions using different techniques. The most prominent approach has been to study the effects of *GA deception* on the GA's performance (e.g., Goldberg, 1987, 1989a; Liepins & Vose, 1990; Whitley, 1991). However, deception is only one among many features of a problem that affect GA performance (e.g., see Liepins & Vose, 1990, and Forrest & Mitchell, 1991). Rather than studying hard problems on which the GA fails, our initial approach has been to examine the GA's behavior on landscapes for which it is likely to perform well. By understanding what features of those landscapes lead to good performance, we hope to better characterize the class of such landscapes.

One major component of this endeavor is to define the simplest class of landscapes on which the GA performs "as expected", thus confirming the broad claims of the building-block hypothesis. However, the task of designing such landscapes has turned out to be substantially more difficult and complex than we originally anticipated. Our initial choices of simple landscapes have revealed some surprising and unanticipated phenomena. The story of how small variations of a basic landscape can make GA search much less effective reveals a great deal about the complexity of GAs and points out the need for a deeper theory of how low-order building blocks are discovered and combined into higher-order solutions.

In the following sections we introduce the *Royal Road* functions, a class of non-deceptive functions in which the building blocks are explicitly defined. We then show how simple variants of these functions can have quite different effects on the performance of the GA, and discuss the reasons for these differences.

2 STEPPING STONES IN THE CROSSOVER LANDSCAPE

The building-block hypothesis suggests two landscape features that are particularly relevant for the GA: (1) the presence of short, low-order, highly fit schemas; and (2) the presence of intermediate "stepping stones"—intermediate-order higher-fitness schemas that result from combinations of the lower-order schemas, and that in turn can combine to create even higher-fitness schemas. Two basic questions about stepping stones are: How much higher in fitness do the intermediate stepping stones have to be for the GA to work well? And how must these stepping stones be configured? To investigate these questions, we first define the Royal Road functions, which contain these features explicitly.

To construct a Royal Road function, we select an optimum string and break it up into a number of small building blocks, as illustrated in Figure 1. We then assign values to each low-order schema and each possible intermediate combination of low-order schemas, and use those values to compute the fitness of a bit string x in terms

s_1 = 11111111**; $c_1 = 8$
s_2 = ********11111111**; $c_2 = 8$
s_3 = ****************11111111**; $c_3 = 8$
s_4 = ************************11111111********************************; $c_4 = 8$
s_5 = ********************************11111111************************; $c_5 = 8$
s_6 = **11111111****************; $c_6 = 8$
s_7 = **11111111********; $c_7 = 8$
s_8 = **11111111; $c_8 = 8$
s_{opt}=1111111111111111111111111111111111111111111111111111111111111111

Figure 1: An optimal string broken up into eight building blocks. The function $R1(x)$ (where x is a bit string) is computed by summing the coefficients c_s corresponding to each of the given schemas of which x is an instance. For example, $R1(1111111100\ldots0) = 8$, and $R1(1111111100\ldots011111111) = 16$. Here $c_s = \mathrm{order}(s)$.

of the schemas of which it is an instance.

The function $R1$, illustrated in Figure 1, is computed very simply: a bit string x gets 8 points added to its fitness for each of the given order-8 schemas of which it is an instance. For example, if x contains exactly two of the order-8 building blocks, $R1(x) = 16$. Likewise, $R1(111\ldots1) = 64$. Stated more generally, the value $R1(x)$ is the sum of the coefficients c_s corresponding to each given schema of which x is an instance. Here c_s is equal to $order(s)$. The fitness contribution from an intermediate stepping stone (such as the combination of s_1 and s_3 in Figure 1) is thus a linear combination of the fitness contribution of the lower-level components. $R1$ is similar to the "plateau" problem described by Schaffer and Eshelman (1991).

According to the building-block hypothesis, $R1$'s building-block and stepping-stone structure should lay out a "royal road" for the GA to follow to the global optimum. In contrast, an algorithm such as simple steepest-ascent hill-climbing, which systematically tries out single-bit mutations and only moves in an uphill direction, cannot easily find high values in such a function, since a large number of single bit-positions must be optimized simultaneously in order to move from an instance of a lower-order schema (e.g., 11111111**...*) to an instance of a higher-order intermediate schema (e.g., 11111111********11111111**...*). While some random search may be involved in finding the lowest-level building blocks (depending on the size of the initial population and the size of the lowest-level blocks), the interesting aspect of $R1$ is studying how lower-level blocks are combined into higher-level ones, and this is the aspect with which we are most concerned. Part of our purpose in designing the Royal Road functions is to construct a class of fitness landscapes that distinguishes the GA from other search methods such as hill-climbing. This actually turned out to be more difficult than we anticipated, as will be discussed in Section 5.

This class of functions provides an ideal laboratory for studying the GA's behavior:

- The landscape can be varied in a number of ways. For example, the "height" of various intermediate stepping stones can be increased or decreased (e.g., the fitness contribution can be a nonlinear combination of the fitness contributions from the components). Also, the size of the lowest-order building blocks can be varied, as can the degree to which they cover the optimum. Finally, different degrees of deception can be introduced by allowing the lower-order schemas to differ in some bits from the higher-order stepping stones, effectively creating low-order schemas that lead the GA away from the good higher-order schemas. The effects of these variations on the GA's behavior can then be studied in detail.

- Since the global optimum, and, in fact, all possible fitness values, are known in advance, it is easy to compare the GA's performance on different variations of Royal Road functions.

- All of the desired schemas are known in advance, since they are explicitly built into the function. Therefore, the dynamics of the search process can be studied in detail by tracing the ontogenies of individual schemas.

We are using the Royal Road functions to study a number of questions about the effects of crossover on various landscapes, including the following: For a given landscape, to what extent does crossover help the GA find highly fit schemas? What is the effect of crossover on the waiting times for desirable schemas to be discovered? What are the bottlenecks in the discovery process? How does the configuration of stepping stones and size of steps defined by stepping stones affect the GA's performance? Answering these questions in the context of the idealized Royal Road functions is a first step towards answering them for more general cases.

We first investigated the effect of the step size of the intermediate stepping stones on the GA's performance. To do this, we compared the performance of the GA on $R1$ with its performance on a second function $R2$, where the fitness contributions of certain intermediate stepping stones are much higher. $R2$ is illustrated in Figure 2. $R2$ is calculated in the same way as $R1$: the fitness of a bit string x is the sum of the coefficients corresponding to each schema (s_1–s_{14}) of which it is an instance.

For example, $R2(1111111100\ldots011111111) = 16$, since the string is an instance of both s_1 and s_8, but $R2(1111111111111111100\ldots0) = 32$, since the string is an instance of s_1, s_2, and s_9. Thus, a string's fitness depends not only on the number of 8-bit schemas to which the string belongs, but also on their positions in the string. The optimum string 11111111...1 has fitness 192, since the string is an instance of each schema in the list.

3 ROYAL ROAD EXPERIMENTS

In an earlier paper (Mitchell, Forrest, & Holland, 1992) we reported some initial results on Royal Road functions. Our main performance measure was the number

s_1 = 11111111**; $c_1 = 8$
s_2 = ********11111111**; $c_2 = 8$
s_3 = ****************11111111**; $c_3 = 8$
s_4 = ************************11111111********************************; $c_4 = 8$
s_5 = ********************************11111111************************; $c_5 = 8$
s_6 = **11111111****************; $c_6 = 8$
s_7 = **11111111********; $c_7 = 8$
s_8 = **11111111; $c_8 = 8$
s_9 = 1111111111111111**; $c_9 = 16$
s_{10} =****************1111111111111111********************************; $c_{10} = 16$
s_{11} =********************************1111111111111111****************; $c_{11} = 16$
s_{12} =**1111111111111111; $c_{12} = 16$
s_{13} =11111111111111111111111111111111********************************; $c_{13} = 32$
s_{14} =********************************11111111111111111111111111111111; $c_{14} = 32$
s_{opt}=1111111111111111111111111111111111111111111111111111111111111111

Figure 2: Royal Road Function $R2$. $R2(x)$ is computed in the same way as $R1$: by summing the coefficients c_s corresponding to each of the given schemas of which x is an instance. For example, $R2(1111111100\ldots011111111) = 16$, but $R2(111111111111111100\ldots0) = 32$. $R2(11111111\ldots1) = 192$.

of generations it took the GA to find the function optimum, although for some experiments we also looked at the discovery time for schemas of different orders. We first confirmed that the GA performs significantly better on $R1$ and $R2$ when crossover is used than when crossover is turned off and only mutation is used, and we showed that both versions of the GA perform significantly better than a simple steepest-ascent hill-climbing algorithm. These results were expected. We then described some unexpected experimental results comparing the GA's performance on $R2$ with its performance on $R1$. Here we extend these experimental results and analyze them in more detail, and compare the GA's performance with that of a more sophisticated hill-climber.

For our initial experiments, we used functions defined over strings of length 64. The GA population size was 128, with the initial population generated at random. In each run the GA was allowed to continue until the optimum string was discovered, and the total number of function evaluations performed was recorded. We used a generational GA with single-point crossover and sigma scaling (Tanese, 1989; Forrest & Mitchell, 1991): an individual i's expected number of offspring is $1+\frac{F_i-\overline{F}}{2\sigma}$, where F_i is i's fitness, $\overline{F}$ is the mean fitness of the population, and σ is the standard deviation. The maximum expected offspring of any string was 1.5—if the above formula gave a higher value, the value was reset to 1.5. This is a strict cutoff, since it implies that most individuals will reproduce only 0, 1, or 2 times. The effect of this selection scheme is to slow down convergence by restricting the effect that a single individual can have on the population, regardless of how much more fit it is than the rest of the population. Even with this precaution, we observe some

ORIGINAL EXPERIMENT		
	Function Evaluations to Optimum	
500 runs	$R1$	$R2$
Mean	62099 (std err: 1390)	73563 (std err: 1794)
Median	56576	66304

Table 1: Summary of results of running the GA on $R1$ and $R2$. The table gives the mean and median function evaluations taken to find the optimum over 500 runs on each function. The numbers in parentheses are the standard errors.

interesting premature convergence effects (described in the following section). The crossover probability was 0.7 per pair of parents and the mutation probability was 0.005 per bit.

The probability that a randomly generated string contains one of the bottom-level order-8 schemas is $8 * \frac{1}{2^8} = \frac{1}{32}$. Since the initial population has 128 randomly generated individuals, there were on average $\frac{128}{32} = 4$ total instances of bottom-level schemas in the initial population. That is, there is a 0.5 probability that there will be an instance of any particular block; thus, since there are 8 different lowest-level blocks, there will on average be 4 total instances of lowest-level blocks in the population.

3.1 EXPERIMENTS ON $R1$ AND $R2$

We expected the GA to perform better—that is, find the optimum more quickly—on $R2$ than on $R1$. In $R2$ there is a very clear path via crossover from pairs of the eight initial order-8 schemas (s_1–s_8) to the four order-16 schemas (s_9–s_{12}), and from there to the two order-32 schemas (s_{13} and s_{14}), and finally to the optimum (s_{opt}). We believed that the presence of this stronger path would speed up the GA's discovery of the optimum, but our experiments showed the opposite: the GA performed significantly better on $R1$ than on $R2$. Statistics summarizing the results of 500 runs on each function are given in Table 1. This table gives the mean and median number of function evaluations taken to find the optimum over 500 runs each on $R1$ and $R2$.

If we hope to understand the GA's performance in general, we need to understand in detail what are the potential bottlenecks for discovering desirable schemas. This has been studied extensively in the deception literature, but $R2$ is a non-deceptive function that nonetheless contains some features that keep the GA from discovering desirable schemas as quickly as in $R1$.

What slows down the GA in the case of $R2$? To investigate this, we took a typical run of the GA on $R2$ and graphically traced the evolution of each schema in the tree. Figure 3 gives this trace for three sets of schemas: s_1, s_2,and s_9; s_3, s_4, and s_{10}; and s_5, s_6, and s_{11} (see Figure 2). In each plot, the density (% of population) of each schema is plotted against time (generations). The density is sampled every

10 generations.

These plots show a striking phenomenon. In the top plot in Figure 3, s_1 and s_2 appear early and instances of them quickly combine to form s_9. Once each schema is discovered, its density in the population rises quite quickly to over 90% of the population by generation 60 or so. Around generation 220 there is a distinct dip in the densities of these three schemas.

The middle plot shows a very different evolution for s_3, s_4, and s_{10}. The schemas s_3 and s_4 are both present in the initial (randomly generated) population (though s_3's presence at generation 0 is not visible on the plot), but while s_4 rises quickly, s_3 dies out by generation 10, is fleetingly rediscovered (along with s_{10}) at generation 120 (see blip on the x-axis), and does not return until the very end of the run, at which point a mutation brings it (along with s_{10}) back (see blip on the x-axis). This same mutation is responsible for creating s_{opt} at generation 535, when the run ends. The schema s_4, after a quick initial rise, enters a pronounced dip at the same time the milder dip can be seen in the top plot of Figure 3, around generation 220.

What is the cause of these dips, and what prevents s_3 from persisting in the population? A likely answer can be inferred from the bottom plot. Schema s_6 appears around generation 30, rises fairly quickly, taking a sharp upturn around generation 220 and rising to about 95% of the population. Schema s_5 appears briefly around generation 20 (blip on the x-axis) and dies out, but appears again at generation 220. The instance of it in the population is also an instance of s_{11}, and instances of s_{11} rise very quickly. This rise exactly coincides with the minor dip in s_1, s_2, and s_9, and the major dip in s_4. What appears to be happening is the following: in the first few instances of s_{11}, along with the 16 1's in the fifth and sixth blocks are several 0's in the first through fourth blocks. An instance of s_{11} has fitness $8 + 8 + 16 = 32$, whereas an instance of an order-8 schema such as s_4 has fitness 8. This difference causes s_{11} to rise very quickly compared to s_4, and instances of s_{11} with some 0's in the fourth block tend to push out many of the previously existing instances of s_4 in the population. This phenomenon has been called "hitchhiking", where 0's in other positions in the string hitchhike along with the highly fit s_{11}. The most likely positions for hitchhikers are those close to the highly fit schema's defined positions, since they are less likely to be separated from the schema's defined positions under crossover. Such effects are seen in real population genetics, and have been discussed in the context of GAs by Schraudolph and Belew (1990), and Das and Whitley (1989), among others. Note that this effect is pronounced even with the relatively weak form of selection used in our GA. (We also compared the GA's performance on $R1$ and $R2$ using a linear rank-scaling method (Baker, 1985) instead of the sigma-scaling method described above, and obtained results similar to those given in Table 1.)

The plots given in Figure 3 come from a single run, but this run was typical; the same type of phenomenon was observed on many of the other runs on $R2$ as well. Our hypothesis is that this hitchhiking effect is what causes the relatively slower times (on average) for the GA to find the optimum on $R2$. The power of crossover

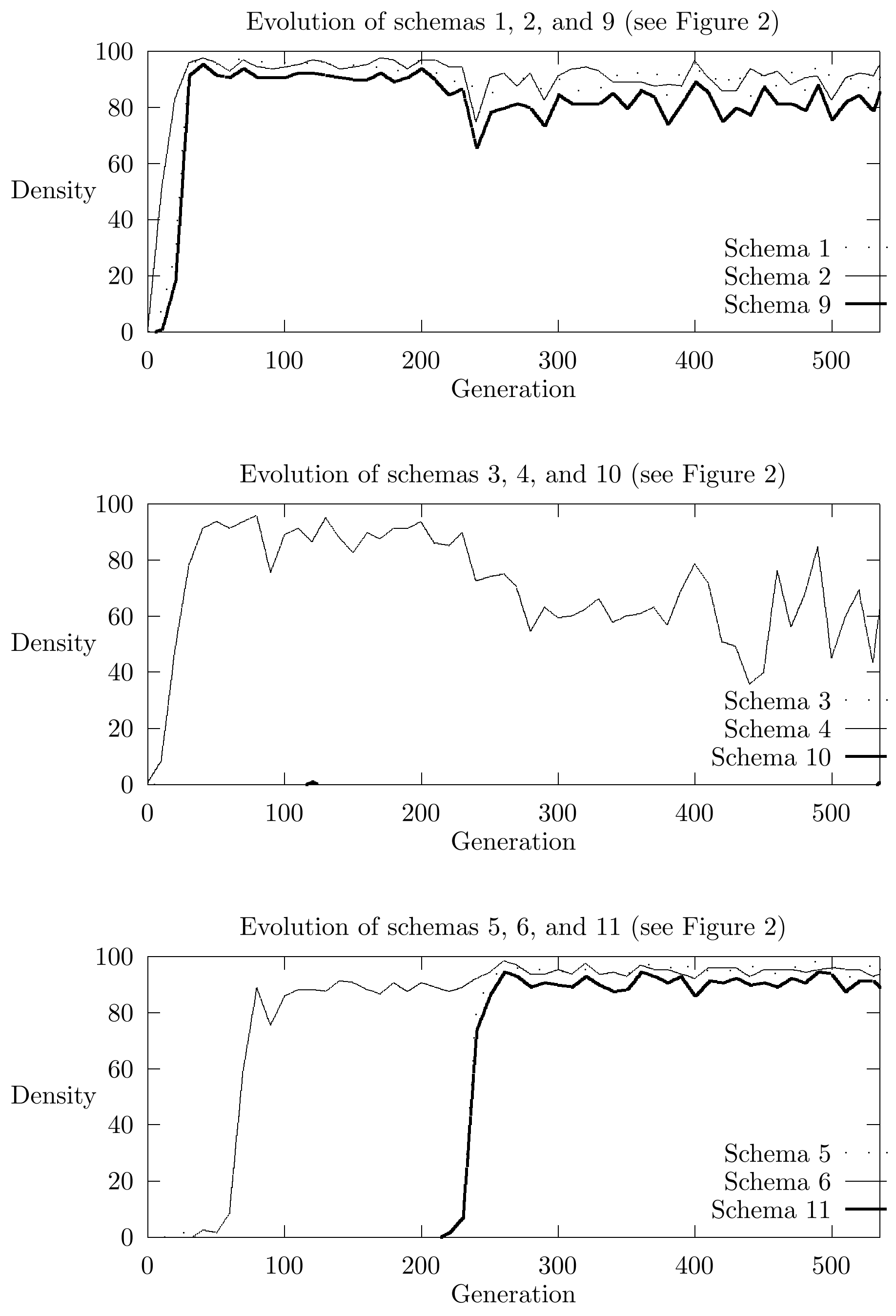

Figure 3: Evolution of three sets of schemas in a typical run of the GA on $R2$. In each plot, the density of each schema (% of population) is plotted against generation. Note that in the middle plot, schemas 3 and 10 are visible only as tiny bumps on the x-axis at generations 120 and 535.

POPULATION SIZE 1024		
	Function Evaluations to Optimum	
200 runs	$R1$	$R2$
Mean	37453 (std err: 868)	43213 (std err: 1275)
Median	34816	36864

Table 2: Summary of results of 200 runs of the GA with population size 1024 on $R1$ and $R2$.

to combine lower-level building blocks was hampered, since some of the necessary building blocks were either partially or totally suppressed by the quick rise of disjoint building blocks. This suggests that there is more to characterizing a GA landscape than the absolute direction of the search gradients. In these functions, it is the actual differences in relative fitnesses for the different schemas that are relevant.

In $R1$, which lacks the extra fitness given to some intermediate-level schemas, the hitchhiking problem does not occur to such a devastating degree. The fitness of an instance of, say, s_{11} in $R1$ is only 16, so its discovery does not have such a dramatic effect on the discovery and persistence of other order-8 schemas in the function. Contrary to our initial intuitions, it appears that the extra reinforcement from some intermediate-level stepping stones actually harms the GA in these functions.

It might be thought that these results are due in part to sampling error: since the lowest order-building blocks are of length 8, a GA with a population of 128 has no samples of many of the lowest-order building blocks in the initial population (on average, 1/2 of the lowest-order blocks will not be represented), and thus has to wait until the order-8 building blocks are created by random variation. [1] To test the effect of this on our results, we performed two additional experiments: (1) we ran the GA on $R1$ and $R2$ with a population size of 1024, which gives 4 expected instances of each order-8 schema in the initial population; and (2) we ran the GA with population size 128 on modified versions of $R1$ and $R2$ in which the lowest-order building blocks were of length 4 rather than length 8. In this latter case, there are on average 8 instances of each order-4 building block in the initial population. The results from these two experiments are given in Tables 2 and 3. They show that these modifications, although improving the GA's absolute performance (especially in the case of order-4 building blocks, which makes the function much easier to optimize), do not change the qualitative difference between the time to optimum for $R1$ and $R2$. This indicates that the difference is not primarily due to sampling error.

These results point to a pervasive and important issue in the performance of GAs: the problem of premature convergence. The fact that we observe a form of premature convergence even in this very simple setting suggests that it can be a factor in

[1] Note: this fact caused our experiments (time to find the optimum) to have a higher-than-normal variance, which is why we performed at least 200 runs for most of our experiments.

LOWEST-ORDER SCHEMAS LENGTH 4		
	Function Evaluations to Optimum	
200 runs	$R1$	$R2$
Mean	6568 (std err: 198)	11202 (std err: 394)
Median	5760	9600

Table 3: Summary of results of 200 runs of the GA on modified versions of $R1$ and $R2$, in which the lowest-order building blocks are of length 4.

VARIANTS OF R2		
	Function Evaluations to Optimum	
200 runs	$R2_{introns}$	$R2_{flat}$
Mean	75599 (std err: 2697)	62692 (std err: 2391)
Median	70400	56448

Table 4: Summary of results of 200 runs of the GA on two variants of $R2$.

any GA search in which the population is simultaneously searching for two or more non-overlapping high fitness schemas (e.g., s_4 and s_{11}), which is often the case. The fact that the population loses useful schemas once one of the disjoint good schemas is found suggests one reason that the rate of effective implicit parallelism of the GA (Holland, 1975; Goldberg, 1989b) may need to be reconsidered. (For another discussion of implicit parallelism in GAs, see Grefenstette & Baker 1989.)

3.2 DO INTRONS SUPPRESS HITCHHIKERS?

In order to understand the hitchhiking behavior more precisely, we performed an experiment that we believed would eliminate it to some degree. Our hypothesis was that hitchhiking occurred in the loci that were spatially adjacent to the high-fitness schemas (e.g., s_{11} above). In order to reduce this effect, we constructed a new function, $R2_{introns}$, by introducing blocks of 8 "introns"—8 additional *'s—in between each of the 8-bit blocks of 1's. Thus in $R2_{introns}$, strings were of length 128 instead of 64. For example, in $R2_{introns}$, s_1 is 11111111**********. . . *, s_2 is ****************11111111**. . . *, and their combination, s_9, is 11111111********11111111**. . . *. The optimum is the string containing each block of 8 1's, where the blocks are each separated by eight loci that can contain either 0's or 1's. The idea here was that a potentially damaging hitchhiker would be at least 8-bits away from the schema on which it was hitchhiking, and would thus be likely to be lost under crossover. (Levenick, 1991, found that inserting introns into individuals improved the performance of the GA in one particular set of environments.)

As shown in column 1 of Table 4, running the GA on $R2_{introns}$ yielded results not significantly different from those for $R2$. This was contrary to our expectations, and the reasons for this result are not clear, but one hypothesis is that once an instance of a higher-order schema (e.g., s_{11}) is discovered, convergence is so fast that hitchhikers are possible even in loci that are relatively distant from the schema's defined positions.

3.3 VARYING THE COEFFICIENTS IN $R2$

It is clear that some intermediate-level reinforcement is necessary for the GA to work. Consider $R1'$, a variant of $R1$, where $R1'(x) = 8$ if x is an instance of at least one of the 8-bit schemas, and $R1'(x) = 64$ if x is an instance of *all* the 8-bit schemas. Here the GA would have no reason to prefer a string with block of 16 1's over a string with a block of 8 1's, and thus there would be no pressure to increase the number of 1's. Intermediate schemas in $R1$ provide *linear* reinforcement, since the fitness of an instance of an intermediate-order schema is always the sum of the fitnesses of instances of the component order-8 schemas. Some schemas in $R2$ provide strong *nonlinear* reinforcement, since the fitness of an instance of, say, s_9 is much higher than the sum of the fitnesses of instances of the component order-8 schemas s_1 and s_2. Our results indicate that the nonlinear reinforcement given by some schemas is too high—it hurts rather than helps the GA's performance.

Does nonlinear reinforcement ever help the GA rather than hinder it? To study this we constructed a new function, $R2_{flat}$, with a much weaker nonlinear reinforcement scheme: for this function, c_1–c_{14} are each set to the flat value 1. Here the reinforcement is still nonlinear (an instance of s_9 will have fitness $1 + 1 + 1$, which is greater than the sum of the two components), but the amount of reinforcement is reduced considerably.

The results of running the GA on $R2_{flat}$ is given in the second column of Table 4. The average time to optimum for this function is approximately the same as for $R1$. Thus the smaller fitness advantage in $R2_{flat}$ does not seem to hurt performance, although it does not result in *improved* performance over that on $R1$.

These phenomena may be related to results by Feldman and his colleagues on the effects of super- and sub-multiplicative fitness functions on the evolutionary viability of crossover (Liberman & Feldman, 1986; Bergman & Feldman, 1990). However, there are several problems with applying Feldman's theorems directly. One problem is that Feldman studies the evolutionary viability of crossover rather than the degree to which crossover helps discover high-fitness individuals. Our work concentrates on the latter. We are currently investigating how these two concerns are related. (This was also studied by Schaffer and Eshelman, 1991.)

4 DISCUSSION

The results described in the previous two sections show that the GA's ability to process building blocks effectively depends not only on their presence, but also on their relative fitness. If some intermediate stepping stones are too much fitter than the primitive components, then premature convergence slows down the discovery of some necessary schemas. Simple introns and a very mild selection scheme do not seem to alleviate the premature convergence and hitchhiking problems.

Our results point out the importance of making the building-block hypothesis a more precise and useful description of building-block processing. While the disruptive effects that we observed (hitchhiking, premature convergence, etc.) are already known in the GA literature, there is as yet no theorem associating them with the building-block structure of a given problem.

In our experiments we have observed that the role of crossover varies considerably throughout the course of the GA search. In particular, three stages of the search can be identified: (1) the time it takes for the GA to discover the lowest-order schemas, (2) the time it takes for crossover to combine lower-order schemas into a higher-order schema, and (3) the time it takes for the higher-order schema to take over the population. In multi-level functions, such as the Royal Road functions, these phases of the search overlap considerably, and it is essential to understand the role of crossover and the details of schema processing at each stage (this issue has also been investigated by Davis, 1989, and by Schaffer & Eshelman, 1991, among others). In previous work, we have discussed the complexities of measuring the relative times for these different phases (Mitchell, Forrest, & Holland, 1992).

5 EXPERIMENTS WITH HILL-CLIMBING

As was mentioned earlier, part of our purpose in designing the Royal Road functions is to construct the simplest class of fitness landscapes on which the GA will not only perform well, but on which it will outperform other search methods such as hill-climbing. In addition to our experiments comparing the GA's performance on $R1$ and $R2$, we compared the GA's performance with that of three commonly used iterated hill-climbing schemes: steepest-ascent hill-climbing, next-ascent hill-climbing (Mühlenbein, 1991), and a scheme we call "random-mutation hill-climbing", that was suggested by Richard Palmer (personal communication). Our implementation of these various hill-climbing schemes is as follows:

- **Steepest-ascent hill-climbing (SAHC):**
  1. Choose a string at random. Call this string *current-hilltop*.
  2. Systematically mutate each bit in the string from left to right, recording the fitnesses of the resulting strings.
  3. If any of the resulting strings give a fitness increase, then set *current-hilltop* to the resulting string giving the highest fitness increase.
  4. If there is no fitness increase, then save *current-hilltop* and go to step 1. Otherwise, go to step 2 with the new *current-hilltop*.
  5. When a set number of function evaluations has been performed, return the highest hilltop that was found.
- **Next-ascent hill-climbing (NAHC):**

HILL-CLIMBING ON R2			
	Function Evaluations to Optimum		
200 runs	SAHC	NAHC	RMHC
Mean	$> 256{,}000$ (std err: 0)	$> 256{,}000$ (std err: 0)	6551 (std err: 212)
Median	$> 256{,}000$	$> 256{,}000$	5925

Table 5: Summary of results of 200 runs of various hill-climbing algorithms on $R2$.

1. Choose a string at random. Call this string *current-hilltop*.
2. Mutate single bits in the string from left to right, recording the fitnesses of the resulting strings. If any increase in fitness is found, then set *current-hilltop* to that increased-fitness string, without evaluating any more single-bit mutations of the original string. Go to step 2 with the new *current-hilltop*, but continue mutating the new string starting after the bit position at which the previous fitness increase was found.
3. If no increases in fitness were found, save *current-hilltop* and go to step 1.
4. When a set number of function evaluations has been performed, return the highest hilltop that was found.

Notice that this method is similar to Davis's "bit-climbing" scheme (Davis, 1991). In his scheme, the bits are mutated in a random order, and *current-hilltop* is reset to any string having *equal* or better fitness than the previous best evaluation.

- **Random-mutation hill-climbing (RMHC):**

1. Choose a string at random. Call this string *best-evaluated*.
2. Choose a locus at random to mutate. If the mutation leads to an equal or higher fitness, then set *best-evaluated* to the resulting string.
3. Go to step 2.
4. When a set number of function evaluations has been performed, return the current value of *best-evaluated*.

Table 5 gives results from running these three hill-climbing schemes on $R2$. In each run the hill-climbing algorithm was allowed to continue either until the optimum string was discovered, or until 256,000 function evaluations had taken place, and the total number of function evaluations performed was recorded. As can be seen, steepest-ascent and next-ascent hill-climbing never found the optimum during the allotted time, but random-mutation hill-climbing found the optimum on average more than ten times faster than the GA with population size 128, and more than six times faster than the GA with population size 1024. Note that random-mutation hill-climbing as we have described it differs from the bit-climbing method used by Davis (1991) in that it does not systematically mutate bits, and it never gives up and starts from a new random string, but rather continues to wander around on plateaus indefinitely. Larry Eshelman (personal communication) has pointed out that the random-mutation hill-climber is ideal for the Royal Road functions—in fact much better than Davis's bit-climber—but will have trouble with any function with local minima. (Eshelman found that Davis's bit-climber does very poorly on $R1$, never finding the optimum in 50 runs of 50,000 function evaluations each.)

In addition to basing our comparison on the number of function evaluations to the optimum, we also compared the average on-line performance (De Jong, 1975) of the GA (population sizes 128 and 1024) with random-mutation hill-climbing, both running on $R2$. Figure 4 plots the results of that comparison. For a given run of the GA or of RMHC, the on-line performance at a given number of function evaluations is defined as the average value of *all* function evaluations made up to that point. We recorded the on-line performance values at intervals of 128 function evaluations, and repeated this procedure for 100 runs. We then averaged the on-line performance values, at each interval of 128 function evaluations, over all the runs. Thus each point on a plot in Figure 4 represents an average of on-line performance values over a number of runs. (The number of values averaged at each point varies: since the GA and RMHC stop when the optimum is found, different runs performed different numbers of function evaluations. We give only averages for which there were a significant number of values to average.)

It can be seen from the plots that RMHC significantly outperforms both versions of the GA under this measure as well.[2]

These results are a striking demonstration that, when comparing the GA with hill-climbing on a particular problem or test-suite, it matters *which* type of hill-climbing algorithm is used. Davis (1991) has also made this point.

The Royal Road functions were originally designed to serve two quite different purposes: (1) as an idealized setting in which to study building-block processing and the role of crossover, and (2) as an example of a simple function that distinguishes GAs from hill-climbing. While we have discovered that certain forms of hill-climbing outperform the GA on these functions (thus, they are inappropriate in exactly this form for the second purpose), they do fulfill the first purpose.

6 CONCLUSIONS AND FUTURE DIRECTIONS

The research described in this paper is an initial step in understanding more precisely how schemas are processed under crossover. By studying the GA's behavior on simple landscapes in which the desirable building blocks are explicitly defined, we have discovered some unanticipated phenomena related to the GA's ability to process schemas efficiently, even in nondeceptive functions. The Royal Road functions capture, in an idealized and clear way, some landscape features that are particularly relevant for the GA, and we believe that a thorough understanding of the GA's be-

[2]It is interesting to note that under this measure, the 128 population GA outperforms the 1024 population GA in the initial stages of the run.

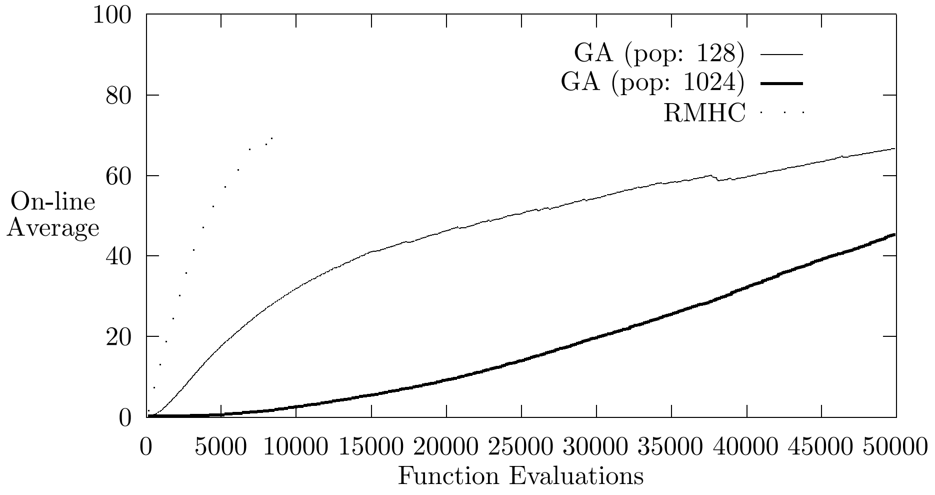

Figure 4: Plots of the average on-line performance of the GA (population sizes 128 and 1024) and of random-mutation hill-climbing (RMHC), over 100 runs. The plot for RMHC stops at around 6000 function evaluations because RMHC had almost always found the function optimum by that time.

havior on these simple landscapes will be very useful in developing more detailed and useful theorems about GA behavior.

The research reported here is work in progress, and there are several directions for future investigation. Here we sketch some of our short and longer range research plans.

In the short term, we plan to study more carefully the bottlenecks in the discovery of desirable schemas, and to quantify more precisely the relationship between the fitness values of the various building blocks and the degree to which these bottlenecks will occur. Hitchhiking is evidently one bottleneck, and we need to understand better in what way it is occurring and under what circumstances. Once we have described the phenomena in more detail, we can begin developing a mathematical model of the schema competitions we observe (illustrated in Figure 3) and how they are affected by different building-block fitness schemes. This model may be related to models proposed by Vose and Liepins (1991).

Our hitchhiking results need to be further analyzed and explained, and we plan a more detailed analysis of the different effects of various nonlinear reinforcement schemes. In particular, more details are needed in the comparison of GA performance on $R1$ with performance on $R2_{flat}$, and on other coefficient schemes.

We believe that there are versions of "royal-road" landscapes that will fulfill our goal of finding simple functions that distinguish GAs from hill-climbing. For example, we plan to try the following variants: adding noise, including all combinations of lower-order schemes in the explicit list of schemas, and allowing schemas to overlap.

The Royal Road functions explore only one type of landscape feature that is of relevance to GAs: the presence and relative fitnesses of intermediate-order building blocks. Our longer-term plans include extending the class of fitness landscapes under investigation to include other types of relevant features; some such features were described by Mitchell, Forrest, and Holland (1992). We are also interested in developing statistical measures that could determine the presence or absence of the features of interest. These might be related to work on determining the correlation structure of fitness landscapes (see Kauffman, 1989; Lipsitch, 1991; and Manderick, de Weger, and Spiessens, 1991). If such measures could be developed, they could be used to help predict the likelihood of successful GA performance on a given landscape.

Acknowledgments

The research reported here was supported by the Center for Nonlinear Studies at Los Alamos National Laboratory, Associated Western Universities, and NSF grant IRI-9157644 (support to S. Forrest); the Michigan Society of Fellows and the University of Michigan EECS Department (support to M. Mitchell); and the Santa Fe Institute (support to both authors). John Holland has collaborated with us in much of this work, and is a constant source of invaluable assistance and advice. We thank Richard Palmer for suggesting the "random-mutation hill-climbing" algorithm, and

Larry Eshelman for comments and reports of experiments on various hill-climbing schemes. We also thank Robert Axelrod, Aviv Bergman, Arthur Burks, Michael Cohen, Marcus Feldman, Rick Riolo, and Carl Simon for many helpful discussions on these issues.

References

J. E. Baker (1985). Adaptive selection methods for genetic algorithms. In J. J. Grefenstette (Ed.), *Proceedings of the First International Conference on Genetic Algorithms and Their Applications.* Hillsdale, NJ: Lawrence Erlbaum Associates.

A. Bergman and M. W. Feldman (1990). More on selection for and against recombination. *Theoretical Population Biology, 38(1)*, 68–92.

R. Das and D. Whitley (1991). The only challenging problems are deceptive: Global search by solving order-1 hyperplanes. In R. K. Belew and L. B Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

L. D. Davis (1989). Adapting operator probabilities in genetic algorithms. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

L. D. Davis (1991). Bit-climbing, representational bias, and test suite design. In R. K. Belew and L. B Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

K. A. De Jong (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor, MI.

S. Forrest and M. Mitchell (1991). The performance of genetic algorithms on Walsh polynomials: Some anomalous results and their explanation. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

D. E. Goldberg (1987). Simple genetic algorithms and the minimal deceptive problem. In L. D. Davis (Ed.), *Genetic Algorithms and Simulated Annealing* (Research Notes in Artificial Intelligence). Los Altos, CA: Morgan Kaufmann.

D. E. Goldberg (1989a). Genetic algorithms and Walsh functions: Part II, Deception and its analysis. *Complex Systems*, 3:153–171.

D. E. Goldberg (1989b). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison Wesley.

J. J. Grefenstette and J. E. Baker (1989). How genetic algorithms work: A critical look at implicit parallelism. In J. D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

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

S. A. Kauffman (1989). Adaptation on rugged fitness landscapes. In D. Stein (Ed.), *Lectures in the Sciences of Complexity*, 527–618. Reading, MA: Addison-Wesley.

J. R. Levenick (1991). Inserting introns improves genetic algorithm success rate: Taking a cue from biology. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms.* 123–127. San Mateo, CA: Morgan Kaufmann.

U. Liberman and M. W. Feldman (1986). A general reduction principle for genetic modifiers of recombination. *Theoretical Population Biology, 30(3)*, 341–371.

G. E. Liepins and M. D. Vose (1990). Representational issues in genetic optimization. *Journal of Experimental and Theoretical Artificial Intelligence, 2*, 101-115.

M. Lipsitch (1991). Adaptation on rugged landscapes generated by local interactions of neighboring genes. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

B. Manderick, M. de Weger, and P. Spiessens (1991). The genetic algorithm and the structure of the fitness landscape. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms.* San Mateo, CA: Morgan Kaufmann.

M. Mitchell, S. Forrest, and J. H. Holland (1992). The royal road for genetic algorithms: Fitness landscapes and GA performance. In *Proceedings of the First European Conference on Artificial Life.* Cambridge, MA: MIT Press/Bradford Books.

H. Mühlenbein (1991). Evolution in time and space—The parallel genetic algorithm. In G. J. E. Rawlins (Ed.), *Foundations of Genetic Algorithms*, 316–337. San Mateo, CA: Morgan Kaufmann.

J. D. Schaffer and L. J. Eshelman (1991). On crossover as an evolutionarily viable strategy. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, 61–68. San Mateo, CA: Morgan Kaufmann.

N. N. Schraudolph and R. K. Belew (1990). Dynamic parameter encoding for genetic algorithms. CSE Technical Report CS 90-175. Computer Science and Engineering Department, University of California, San Diego.

R. Tanese (1989). *Distributed Genetic Algorithms for Function Optimization.* Unpublished doctoral dissertation, University of Michigan, Ann Arbor, MI.

M. Vose and G. Liepins. (1991). Punctuated equilibria in genetic search. *Complex Systems 5*, 31–44.

L. D. Whitley (1991). Fundamental principles of deception in genetic search. In G. Rawlins (Ed.), *Foundations of Genetic Algorithms.* San Mateo, CA: Morgan

Kaufmann.