B.2.7.5: Fitness Landscapes: Royal Road Functions

Melanie Mitchell
Santa Fe Institute
1399 Hyde Park Road
Santa Fe, NM 87501
mm@santafe.edu

Stephanie Forrest
Dept. of Computer Science
University of New Mexico
Albuquerque, NM 87131
forrest@cs.unm.edu

To appear in Bäck, T., Fogel, D., and Michalewicz, Z. (Eds.), *Handbook of Evolutionary Computation.* Oxford: Oxford University Press.

An important goal of research on genetic algorithms (GAs) is to understand the class of problems for which GAs are most suited, and in particular, the class of problems on which they will outperform other search algorithms such as gradient methods. We have developed a class of fitness landscapes—the "Royal Road" functions (Mitchell, Forrest, and Holland 1992; Forrest and Mitchell 1993)—that isolate some of the features of fitness landscapes thought to be most relevant to the performance of GAs. Our goal in constructing these landscapes is to understand in detail how such features affect the search behavior of GAs and to carry out systematic comparisons between GAs and other search methods.

It has been hypothesized that GA's work by discovering, emphasizing, and recombining high-quality *building blocks* of solutions in a highly parallel manner (Holland 1975; Goldberg 1989). These ideas are formalized by the "Schema Theorem" and "Building-Block Hypothesis" (see section B2.5, this volume). The GA evaluates populations of strings explicitly, and at the same time, it is argued, it implicitly estimates, reinforces, and recombines short, high-fitness *schemas*—building blocks encoded as templates, such as 11****** (a template representing all 8-bit strings beginning with two 1s).

A simple Royal Road function, R_1, is shown in Figure 1. R_1 consists of a list of partially specified bit strings (*schemas*) s_i in which '$*$' denotes a wild card (i.e., allowed to be either 0 or 1). A bit string x is said to be an *instance* of a schema s, $x \in s$, if x matches s in the defined (i.e., non-'$*$') positions. The fitness $R_1(x)$ of a bit string x is defined as follows:

$$R_1(x) = \sum_{i=1}^{8} \delta_i(x) o(s_i), \text{where } \delta_i(x) = \begin{cases} 1 & \text{if } x \in s_i \\ 0 & \text{otherwise,} \end{cases}$$

and where $o(s_i)$, the *order* of s_i, is the number of defined bits in s_i. For example, if x is an instance of exactly two of the order-8 schemas, $R_1(x) = 16$. Likewise, $R_1(111\ldots1) = 64$. R_1 is meant to capture one landscape feature of particular relevance to GAs: the presence of fit low-order building blocks that recombine to produce fitter, higher-order building blocks. (A different class of functions, also called "Royal Road functions", was developed by Holland and is described in Jones, 1995.)

The Building-Block Hypothesis implies that such a landscape should lay out a "royal road" for the GA to reach strings of increasingly higher fitnesses. One might also expect that simple hill-climbing schemes would perform poorly because a large number of bit positions must be optimized simultaneously in order to move from an instance of a low-order schema (e.g., 11111111**...*) to an instance of a higher-order intermediate schema (e.g., 11111111********11111111**...*). However, the results of our experiments ran counter to both these expectations (Forrest and Mitchell 1993). In these experiments, a simple GA (using fitness-proportionate selection with sigma scaling, single-point crossover, and point mutation—see sections C2 and C3, this volume) optimized R_1 quite slowly, at least in part because of "hitchhiking": once an instance of a higher-order schema was discovered, its high fitness allowed the schema to spread quickly in the population, with 0s in other positions in the string hitchhiking along with the 1s in the schema's defined positions. This slowed down the discovery of schemas in the other positions, especially those that are close to the highly-fit schema's defined positions. Hitchhiking can in general be a serious bottleneck for the GA, and we observed similar effects in several variations of our original GA.

The other hypothesis—that the GA would outperform simple hill-climbing on these functions—was also proved wrong. We compared the GA's performance on R_1 (and variants of it) with three different hill-climbing methods: steepest ascent hill-climbing (SAHC), next-ascent hill-climbing (NAHC), and "random mutation hill-climbing" (RMHC) (Forrest and Mitchell 1993). These work as follows (assuming that *max_evaluations* is the maximum number of fitness-function evaluations allowed):

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

4. If no increases in fitness were found, save *current-hilltop* and go to step 1.

Notice that this method is similar to Davis's (1991) "bit-climbing" scheme, in which 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. If the optimum has been found, stop and return it. If *max_evaluations* has been e-qualed or exceeded, stop and return the current value of *best-evaluated*. Otherwise go to step 3.
3. 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, and go to step 2.

Note that in SAHC and NAHC, the current string is replaced only if an *improvement* in fitness is found, whereas in RMHC, the current string is replaced whenever a string of *equal* or greater fitness is found. This difference allows RMHC to explore "plateaus", which, as will be seen, produces a large difference in performance.

The results of SAHC and NAHC on R_1 were as expected—while the GA found the optimum on R_1 in an average of $\sim 60,000$ function evaluations, neither SAHC nor NAHC ever found the optimum within the maximum of 256,000 function evaluations. However, RMHC found the optimum in an average of $\sim 6,000$ function evaluations—approximately a factor of 10 faster than the GA. This striking difference on landscapes originally designed to be "royal roads" for the GA underscores the need for a rigorous answer to the question posed earlier: "Under what conditions will a GA outperform other search algorithms, such as hill climbing?"

To answer this, we first performed a mathematical analysis of RMHC, which showed that the expected number of function evaluations to reach the optimum on an R_1-like function with N blocks of K 1s is $\sim 2^K N(logN + \gamma)$ (where γ is Euler's constant) (Mitchell, Holland, and Forrest 1994; our analysis is similar to that given for a similar problem in Feller, 1960, p. 210.) We then described and analyzed an "idealized GA" (IGA), a very simple procedure that significantly outperforms RMHC on R_1. The IGA works as follows. On each iteration, a new string is chosen at random, with each bit independently being set to 0 or 1 with equal probability. If a string is found that contains one or more of the desired schemas, that string is saved. When a string containing one or more not-yet-discovered schemas is found, it is crossed over with the saved string in such a way so that the saved string contains all the desired schemas that have been found so far. (Note that the probability of finding a given 8-bit schema in a randomly chosen string is 1/256.)

This procedure is unusable in practice, because it requires knowing precisely what the desired schemas are. However, the idea behind the IGA is that it does explicitly what the GA is thought to do implicitly, namely identify and sequester desired schemas via reproduction and crossover ("exploitation") and sample the search space via the initial random population, random mutation, and crossover ("exploration"). We showed that the expected number of function evaluations for the IGA to reach the optimum on an R_1-like function with N blocks of K 1s is $\sim 2^K(logN + \gamma)$, a factor of N faster than RMHC (Mitchell, Holland, and Forrest 1994).

What makes the IGA so much faster than the simple GA and RMHC on R_1? A primary reason is that the IGA perfectly implements the notion of "implicit parallelism" (Holland, 1975): each new string is completely independent of the previous one, so new samples are given independently to each schema region. In contrast, RMHC moves in the space of strings by single-bit mutations from an original string, so each new sample has all but one of the same bits as the previous sample. Thus each new string gives a new sample to only one schema region. We ignore the construction time to construct new samples and compare only the number of function evaluations to find particular fitness values. This is because in most interesting GA applications, the time to perform a function evaluation dominates the time required to execute the other parts of the algorithm. For this reason, we assume that the remaining parts of the algorithm take constant time per function evaluation.

The IGA gives a lower bound on the expected number of function evaluations that the GA will need to solve this problem. It is a lower bound because the IGA is given perfect information about the desired schemas, which is not available to the simple GA. (If it were, there would be no need to run the GA as the problem solution would already be known.)

Independent sampling allows for a speed-up in the IGA in two ways: it allows for the possibility of more than one desired schema appearing simultaneously on a given sample, and it also means that there are no wasted samples, as there are in RMHC. Although the comparison we have made is with RMHC, the IGA will also be significantly faster on R_1 (and similar landscapes) than any hill-climbing method that works by mutating single bits (or a small number of bits) to obtain new samples.

The hitchhiking effects described earlier also result in a loss of independent samples for the real GA. The goal is to have the real GA, as much as possible, approximate the IGA. Of course, the IGA works because it explicitly knows what the desired schemas are; the real GA does not have this information and can only estimate what the desired schemas are by an implicit sampling procedure. However, it is possible for the real GA to approximate a number of the features of the IGA:

- *Independent samples:* The population size has to be sufficiently large, the selection process has to be sufficiently slow, and the mutation rate has to be sufficiently great to ensure that no single locus is fixed at a single value in every string (or even a large majority of strings) in the population.

- *Sequestering desired schemas:* Selection has to be strong enough to preserve desired schemas that have been discovered, but it also has to be slow enough (or, equivalently, the relative fitness of the non-overlapping desirable schemas has to be small enough) to prevent significant hitchhiking on some highly-fit schemas, which can crowd out desired schemas in other parts of the string.

s_1 = 11111111**;

s_2 = ********11111111**;

s_3 = ****************11111111**;

s_4 = ************************11111111********************************;

s_5 = ********************************11111111************************;

s_6 = **11111111****************;

s_7 = **11111111********;

s_8 = **11111111;

Figure 1: Royal Road function R_1.

- *Instantaneous crossover:* The crossover rate has to be such that the time until a crossover combines two desired schemas is small with respect to the discovery time for the desired schemas.

- *Speed-up over RMHC*: The string length (a function of N) has to be large enough to make the N speed-up factor significant.

These mechanisms are not all mutually compatible (e.g., high mutation works against sequestering schemas), and thus must be carefully balanced against one another. A discussion of how such a balance might be achieved is given by Holland (1993); some experimental results are given in Mitchell, Holland, and Forrest (1994).

In conclusion, our investigations of a simple GA, RMHC, and the IGA on R_1 and related landscapes are one step towards our original goals—to design the simplest class of fitness landscapes that will distinguish the GA from other search methods, and to characterize rigorously the general features of a fitness landscape that make it suitable for a GA. Our results have shown that it is not enough to invoke the Building-Block Hypothesis for this purpose. Royal Road landscapes such as R_1 are not meant to be realistic examples of problems to which one might apply a GA. Rather, they are meant to be idealized problems in which certain features most relevant to GAs are explicit, so that the GA's performance can be studied in detail. Our claim is that, in order to understand how the GA works in general and where it will be most useful, we must first understand how it works and where it will be most useful on simple yet carefully designed landscapes such as these.

Acknowledgments

This work has been supported by the Santa Fe Institute's Adaptive Computation Program, the Alfred P. Sloan Foundation (grant B1992-46), and the National Science Foundation (grants IRI-9157644 and IRI-9224912).

References

Davis, L. D. 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*, 18–23. San Francisco, CA: Morgan Kaufmann.

Feller, W. 1960. *An Introduction to Probability Theory and its Applications.* New York: John Wiley & Sons. (Second Edition.)

Forrest, S., and Mitchell, M. 1993. Relative building block fitness and the building block hypothesis. In L. D. Whitley, ed., *Foundations of Genetic Algorithms 2*, 109–126. San Francisco, CA: Morgan Kaufmann.

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

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press. (Second edition: MIT Press, 1992.)

Holland, J. H. 1993. Innovation in Complex Adaptive Systems: Some Mathematical Sketches. Working Paper 93-10-062, Santa Fe Institute.

Jones, T. 1995. A Description of Holland's Royal Road Function. *Evolutionary Computation* 2(4): 409–415.

Mitchell, M., Forrest, S., and Holland, J. H. 1992. The royal road for genetic algorithms: Fitness landscapes and GA performance. In F. J. Varela and P. Bourgine, eds., *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, 245–254. Cambridge, MA: MIT Press.

Mitchell, M., Holland, J. H., and Forrest, S. 1994. When will a genetic algorithm outperform hill climbing? In J. D. Cowan, G. Tesauro, and J. Alspector, eds., *Advances in Neural Information Processing Systems 6*, 51–58. San Francisco, CA: Morgan Kaufmann.