When Will a Genetic Algorithm Outperform Hill-Climbing?

Melanie Mitchell
Santa Fe Institute
1660 Old Pecos Trail, Suite A
Santa Fe, New Mexico 87501
Email: mm@santafe.edu

John H. Holland
Departments of Psychology and EECS
University of Michigan
Ann Arbor, Michigan 48109
Email: John.Holland@um.cc.umich.edu

Abstract

In this paper we review some previously published experimental results in which a simple hill-climbing algorithm—Random Mutation Hill-Climbing (RMHC)—significantly outperforms a genetic algorithm on a simple "Royal Road" function. We present an analysis of RMHC followed by an analysis of an "idealized" genetic algorithm (IGA) that is in turn significantly faster than RMHC. We isolate the features of the IGA that allow for this speedup, and discuss how these features can be incorporated into a real GA and a fitness landscape, making the GA better approximate the IGA. We use these features to design a modified version of the previously published experiments, and give new experimental results comparing the GA and RMHC.

1. Introduction

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. There have been a number of empirical comparisons of GAs with other search and optimization methods such as simple hill-climbing (e.g., [19, 4, 20, 7, 8, 9]) simulated annealing (e.g., [11, 18]), other methods based on ideas from biological evolution (e.g., [1, 6]), linear, nonlinear, and integer programming techniques [2], and a number of other traditional function optimization techniques (e.g., [12]). However, such comparisons are typically made by comparing the GA and a second algorithm on a single problem or set of problems, often using performance criteria (e.g., the fraction of runs on which the algorithm finds the optimum of a particular function) which may or may not be appropriate to the GA. These comparisons also do not generally investigate what features led to better performance by one or the other algorithm. Thus it is hard to distill general principles from these isolated results. In this paper we look in depth at a particular simple hill-climbing method and an idealized form of the GA in order to distill some general principles about when and why a GA will outperform hill-climbing. We then present some preliminary experimental results to back up our analysis.

2. Royal Road Landscapes

In previous work we have developed a class of fitness landscapes (the "Royal Road" functions) [14, 9] that resulted from our attempt to identify the features of fitness landscapes that are

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: Royal Road function $R1$.

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 $R2$.

most relevant to the performance of the GA, and to construct the simplest class of landscapes that contain these features. One of our purposes in developing these landscapes is to carry out comparisons with other search methods in a systematic and enlightening way.

Two Royal Road functions, $R1$ and $R2$, are shown in Figures 1 and 2. Under $R1$, 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, $R1(111\ldots1) = 64$. ($R1$ is similar to the "plateau" problem described by Schaffer and Eshelman [16].) $R2$ is calculated in the same way as $R1$: the fitness of a bit string x is the sum of the coefficients corresponding to each named 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 $R2(111111111111111100\ldots0) = 32$, 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. Previous Experimental Results on Royal Road Landscapes

In our previous work [14, 9], we used these two landscapes to study building block processing under the GA. We initially expected that the building-block structure of both $R1$ and $R2$ would lay out a "royal road" for the GA to follow to the global optimum, and that the GA would be faster on $R2$ than on $R1$ because of the added reinforcement from certain higher-level schemas. We also expected that simple hill-climbing schemes would perform poorly on these landscapes 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**... *). However both these expectations were overturned [14, 9]. In our experiments, the GA found the optimum more quickly on $R1$ than $R2$, at least in part because of "hitchhiking": once a higher-order schema (such as s_9 in $R2$) is discovered, it quickly attains high levels in the population, with 0's in other positions in the string hitchhiking along with the 1's in the schema positions. This slows down the discovery of schemas in the other positions, especially those close to the highly fit schema's defined positions. The hitchhiking effect is stronger in $R2$ than in $R1$ because certain intermediate-order schemas confer higher fitness in $R2$ than they do in $R1$ and thus are more likely to give rise to destructive hitchhiking. However, hitchhiking occurs in $R1$ as well, and can in general be a serious bottleneck for the GA. (Such effects are seen in real population genetics, and have been discussed in the context of GAs by Schraudolph and Belew [17], and Das and Whitley [3], among others.)

Our other expectation—that the GA would outperform simple hillclimbing on these landscapes—was also proved wrong. Forrest and Mitchell [9] compared the GA's performance on $R2$ with three different hill-climbing methods: steepest ascent hill-climbing (SAHC), next-ascent hill-climbing (NAHC), and a third method, suggested by Richard Palmer [15], that Forrest and Mitchell termed "random mutation hill-climbing" (RMHC). RMHC works as follows:

Repeat until the optimum is found (or a maximum number of function evaluations has been performed):

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.

(See [9] for details of the other two hill-climbing methods.) The results on SAHC and NAHC were as expected—while the GA found the optimum on $R2$ in an average of 73,563 function evaluations, neither SAHC nor NAHC ever found the optimum even after 256,000 function evaluations (the maximum allowed for all the algorithms).[1] In addition, Eshelman [5] found that Davis's bit-climber [4] never found the optimum of $R1$ in 50 runs of 50,000 function evaluations each.

[1] Forrest and Mitchell's hill-climbing experiments were performed on $R2$, but the results are expected to be roughly the same for $R1$, since each hill-climbing method looks only at the direction of the fitness gradient, not its magnitude.

However, RMHC found the optimum on $R2$ in an average of 6551 function evaluations—more than a factor of 10 faster than the GA. This striking difference on landscapes that were originally designed to be "royal roads" for the GA underscores the need for a rigorous answer to the question posed earlier: "On what types of problems will GAs outperform other search algorithms, such as hill-climbing?"

4. Analysis of Random Mutation Hill-Climbing

To begin to answer this question, we analyzed the RMHC algorithm with respect to the Royal Road functions. A rough analysis of RMHC goes as follows (a similar, more detailed analysis was performed by Richard Palmer [15]). Suppose the Royal Road function consists of N adjacent blocks of K 1's each (in $R1$, $N = 8$ and $K = 8$). What is the expected time (number of function evaluations) to find the optimum string of all 1's? We can first ask a simpler question: what is the expected time to find a single block of K 1's? If all the bits in the block were simultaneously randomly mutated, then the answer would be 2^K, but since we are randomly mutating only one bit at a time, 2^K is only an approximation—actually, a lower bound for most K. For large K this is a fairly good approximation [15]. As will be seen, for our purposes this approximation suffices.

Now suppose we want RMHC to discover a string with N blocks of K 1's. The time to discover a first block of K 1's is $\sim 2^K$, but the time to discover a second block is longer, since many of the function evaluations are "wasted" on testing mutations inside the first block. These mutations will never lead to a higher or equal fitness, since once a first block is already set to all 1's, any mutation to those bits will decrease the fitness. Once a first block of all 1's has been discovered, the proportion of *non-wasted* mutations is $(KN - K)/KN$, that is, the proportion of mutations that occur in the $KN - K$ positions outside the first block. The the expected time to find a second block of K 1's is $\sim 2^K$ times the inverse of this expression: $KN/(KN - K) = N/(N - 1)$. (That is, if the algorithm only spends $1/m$ of its time in useful mutations, then it will require m times as long to accomplish what it could if no mutations were wasted.) Similarly, the expected time to find the third block of K 1's is $\sim 2^K(N/(N-2))$, and so on. The total (approximate) expected time to find all N blocks of K 1's is the sum of these:

$$2^K + 2^K\frac{N}{N-1} + 2^K\frac{N}{N-2} + \ldots + 2^K\frac{N}{N-(N-1)} = 2^K N\left[1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{N}\right].$$

This expression is approximately equal to $2^K N(\log N + \gamma)$, where γ is Euler's constant. For $K = 8, N = 8$, the value of this expression is 5556. When we ran RMHC on this fitness function 200 times, the average number of function evaluations to the optimum was 6551, which shows that the approximation used in the analysis gives a fairly reasonable lower bound.[2]

[2]Palmer gives a figure of 301.2 for the expected time to find a single block of 8 1's [15]; using this figure instead of $2^8 = 256$ in the formula above, the expected time is calculated to be 6549, which is very close to the value obtained in the simulations.

5. An Idealized Genetic Algorithm

In what situations will the GA do better than this? To answer this question it helps to analyze an "idealized GA" (IGA)—a very simple procedure that significantly outperforms RMHC on the Royal Road functions. We will then identify the features of this procedure that lead to the significant speed-up, and give the results of experiments in which we incorporate these features into a real GA and into the Royal Road landscapes.

The IGA procedure is the following:

- On each time step, choose a new string at random, with uniform probability for each bit.
- The first time a string is found that contains one or more of the desired schemas, save that string.
- When a string containing one or more not-yet-discovered schemas is found, instantaneously cross over the new string with the saved string so that the saved string contains all the desired schemas that have been discovered so far.

This procedure is unusable in practice, since it requires knowing precisely what the desired schemas are, whereas in general (as in the GA and in RMHC) an algorithm can only measure the fitness of a string, and does not know ahead of time what schemas make for good fitness. However, the idea behind the GA is to do implicitly what the IGA is able to do explicitly. This idea will be elaborated below.

Suppose again that our desired schemas consist of N blocks of K 1's each. What is the expected time until the saved string contains all the desired schemas? (Here one time step corresponds to the choice of one string.) The following analysis is due to Greg Huber [10]. First consider a single desired schema s (i.e., $N = 1$). Let p be the probability of finding s on a random string (here $p = 1/2^K$), and let q be the probability of not finding s: that is, $q = 1 - p$. Then the probability $\mathcal{P}_1(t)$ that s will be found by time t (that is, at any time step between 0 and t) is:

$$\begin{aligned} \mathcal{P}_1(t) &= 1 - \text{probability that } s \text{ will not be found by time } t \\ &= 1 - q^t. \end{aligned}$$

Now consider the case with N desired schemas. Let $\mathcal{P}_N(t)$ be the probability that all N schemas have been found by time t. Then,

$$\mathcal{P}_N(t) = (1 - q^t)^N.$$

$\mathcal{P}_N(t)$ gives the probability that all N schemas will be found *sometime* in the interval $[0,t]$. But we want the expected time to find all N schemas, which means we need the probability $P_N(t)$ that the last of the N desired schemas will be found at exactly time t. This probability is computed by subtracting the probability that all N schemas will be found by $t-1$ from the probability that all N schemas will be found by t. This gives the probability that the last schema will not be found by $t-1$ but will be found by t:

$$\begin{aligned} P_N(t) &= \mathcal{P}_N(t) - \mathcal{P}_N(t-1) \\ &= [1-q^t]^N - [1-q^{t-1}]^N. \end{aligned}$$

To get the expected time E_N from this probability, we sum over t times the probability:

$$\begin{aligned} E_N &= \sum_1^\infty t\ P_N(t) \\ &= \sum_1^\infty t\ ([1-q^t]^N - [1-q^{t-1}]^N). \end{aligned}$$

The expression $[1-q^t]^N - [1-q^{t-1}]^N$ can be expanded via the binomial theorem and simplifies to:

$$\left[\binom{N}{1}(\frac{1}{q}-1)q^t\right] - \left[\binom{N}{2}(\frac{1}{q^2}-1)q^{2t}\right] + \left[\binom{N}{3}(\frac{1}{q^3}-1)q^{3t}\right] \ldots - \left[\binom{N}{N}(\frac{1}{q^N}-1)q^{Nt}\right].$$

(N is arbitrarily assumed to be even; this assumption gives the minus sign before the last term.)

Now this expression must be multiplied by t and summed from 1 to ∞. We can split the sum into a sum of N infinite sums, over each of the N terms in the expression above. The infinite sum over the first term is:

$$\begin{aligned} & \binom{N}{1}(\frac{1}{q}-1)\sum_1^\infty t\ q^t \\ =\ & \binom{N}{1}(\frac{1}{q}-1)\ [q + 2q^2 + 3q^3 + \ldots] \\ =\ & \binom{N}{1}(\frac{1}{q}-1)\ q\ [1 + 2q + 3q^2 + \ldots] \\ =\ & \binom{N}{1}(\frac{1}{q}-1)\ q\ \frac{d}{dq}[q + q^2 + q^3 + \ldots] \\ =\ & \binom{N}{1}(\frac{1}{q}-1)\ q\ \frac{d}{dq}[\frac{q}{1-q}] (\text{since } 0 \leq q \leq 1) \\ =\ & \binom{N}{1}(\frac{1}{q}-1)\ q\ (\frac{1}{1-q})^2 \\ =\ & \binom{N}{1}\frac{1}{1-q}. \end{aligned}$$

Similary, the infinite sum over the ith term of the sum can be shown to be

$$\binom{N}{i}\frac{1}{1-q^i}.$$

Recall that $q = 1 - p$. If we substitute $1 - p$ for q and assume that p is small so that $(1-p)^n \approx 1 - np$, we obtain the following approximation:

$$E_N \approx \frac{1}{p}\left[\frac{\binom{N}{1}}{1} - \frac{\binom{N}{2}}{2} + \frac{\binom{N}{3}}{3} - \ldots \frac{\binom{N}{N}}{N}\right].$$

For $N = 8$, $K = 8$ the approximation gives an expected time of 696, which is the exact result we obtained as the mean over 200 runs of a simulation of the IGA (the standard error was 19.7).

This expression can (with some work) be shown to be equal to $1/p$ times the harmonic sum $\sum_{n=1}^{N} \frac{1}{n}$, which is approximately $\log N + \gamma$. Since $p = 1/2^K$, we have

$$E_N \approx (1/p)[\log N + \gamma] = 2^K[\log N + \gamma]$$

Setting aside the details of this analysis, the major point is that the IGA gives an expected time that is on the order of $2^K \log N$, where RMHC gives an expected time that is on the order of $2^K N \log N$, a factor of N slower. This kind of analysis can help us understand how and when the GA will outperform hill-climbing.

What makes the IGA faster than RMHC? A primary reason is that the IGA's samples are uniformly distributed while RMHC's samples are neighborhood biased—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. For example, suppose we are running RMHC on $R1$ and at a certain time step during the run the current best string (*best-evaluated*) is an instance of the two leftmost desired order-8 schemas, followed by all 0's:

$$\textit{best-evaluated} = 1111111111111111100\ldots$$

On the next time step, RMHC's next sample will be a single bit mutation of this string, that is, a string in its immediate Hamming neighborhood. This next sample cannot possibly be an instance of any of the other six desired order-8 schemas; in fact, it might be a wasted mutation in the first 16 bit positions. In contrast, if this string were the IGA's current best string, the IGA's next sample would be completely independent of it, with the potential for new bits in every bit position. In fact, the next sample could very well contain one or more of the other six desired schemas. This possibility amounts to an enormous speedup, and the more desired schemas to be found (N), the larger the speedup.

In short, with every function evaluation the IGA is taking independent samples in each of the N schema regions whereas RMHC is not. Independent sampling allows for a speed-up in the IGA in two ways: it allows for the possibility of more than one desirable schema appearing simultaneously on a given sample, and it also means that there are no wasted samples as there are in RMHC.

The hitchhiking effects described earlier also result in a loss of independence of samples. For example, suppose that the GA (the version Forrest and Mitchell used in their previous experiments) is running on $R2$ and has discovered s_1–s_8 (see Figure 2) on different strings, and then puts s_1 and s_2 together on the same string to form s_9. Suppose this string S contains several 0's in the positions corresponding to s_3:

$$S = 1111111111111111101001011 \ldots$$

Because of the presence of s_9, S is much fitter than any other string in the population, and copies of it quickly fill up the population, with the 0's in the s_3 positions hitchhiking along. (Since s_3 is adjacent to s_9, hitchhiking in these positions is more likely than in other positions, since crossover is less likely to separate these bits from s_9.) This means that the previous instances of s_3 will be greatly reduced in the population and that most strings in the population will be nearly identical to S in the s_3 positions. Since most strings are identical in those positions, mutation is the only force that can create variation there and, as in RMHC, each mutation results in a new sample that differs from S by only one bit. As in RMHC, there is no possibilty that a single mutation can produce an instance of s_3 here. As in RMHC, the GA is no longer taking independent samples in that schema position. This was a large factor in the slow-down of the GA on $R2$ as compared to $R1$.

The IGA sequesters the desired schemas as soon as they are discovered, instantly combining them with the desired schemas that have already been found. These desired schemas are never lost. This is also true for RMHC, which will never accept a new string with a lower fitness, and thus preserves the desired schemas it has found. But this is in contrast with the original results on Royal Road functions where hitchhiking often caused desired schemas to be lost after they had been discovered.

Although the comparison we have made is with RMHC, the IGA will also be significantly faster on $R1$ (and similar landscapes) than any hill-climbing method that works by mutating single bits (or a small number of bits) to obtain new samples.

6. The IGA and the Real GA

The goal is to have the GA, as much as possible, approximate the IGA. Of course the IGA works because it explicitly knows what the desired schemas are; the GA does not have this information and can only estimate what the desired schemas are by an implicit sampling procedure. But it is possible for the GA to approximate a number of the features of the IGA, as follows:

- *Independent samples:* The population size has to be large enough, the selection process has to be slow enough, and the mutation rate has to be sufficient to make sure that no single locus is fixed at a single value in every (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.

Level 1: s_1 intron s_2 intron s_3 intron s_4 intron s_5 intron s_6 intron s_7 intron s_8 intron

Level 2: (s_1 intron s_2 intron) (s_3 intron s_4 intron) (s_5 intron s_6 intron) (s_7 intron s_8 intron)

Level 3: (s_1 intron s_2 intron s_3 intron s_4 intron) (s_5 intron s_6 intron s_7 intron s_8 intron)

Level 4: (s_1 intron s_2 intron s_3 intron s_4 intron s_5 intron s_6 intron s_7 intron s_8 intron)

Figure 3: Royal Road Function $R3$.

- *Instantaneous crossover:* The crossover rate has to be such that the time for a crossover that combines two desired schemas to occur 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.

7. Results of Experiments

As a first step in exploring these balances, we designed a new, rather contrived version of $R2$, called $R3$, based on some of the features described above. $R3$ is illustrated in Figure 3. In $R3$, the desired schemas are s_1–s_8 and combinations of them, as in $R2$. However, in $R3$ the lowest-level order-8 schemas are each separated by "introns" (bit positions that do not contribute to fitness—see [9, 13]) of length 24. Thus the combination schemas such as s_9 now consist of combinations of lower-level schemas separated by introns. For example, s_9 consists of s_1 followed by a 24-bit intron followed by s_2 followed by a 24-bit intron. This is displayed as "(s_1 intron s_2 intron)". Under $R3$, a string that is not an instance of any desired schema receives fitness 1. Every time a new level is reached—i.e., a string is found that is an instance of one or more schemas at that level but of no higher-level schemas—a small increment u is added to the fitness. Thus strings at level 1 have fitness $1 + u$, strings at level 2 have fitness $1 + 2u$, etc. For our experiments we set $u = 0.2$. Our population size was 2000, our mutation rate was 0.005 (mutation always flips a bit), and we used multipoint crossover, where the number of crossover points for each pair of parents was selected from a Poisson distribution with mean 2. The number of expected offspring for each individual was its fitness divided by the mean fitness of the population.

This function and set of parameters was a first, contrived attempt to have a real GA approximate an IGA on these functions. The purpose of the introns was to help maintain independent samples in each schema position by preventing linkage between schema positions. The independence of samples is also helped by the large population and the much slower selection scheme given by the function.

We performed 10 different runs of the GA and of RMHC on this function, each with a different random-number seed. The GA and RMHC were each run for a maximum of 10^6 function evaluations. Table 1 gives the mean number of evaluations to reach levels 1, 2, and

10 runs	Level 1	Level 2	Level 3
GA	1,000 (0)	14,800 (3,258)	93,400 (6,854)
RMHC	2,200 (359)	15,500 (3,426)	643,000 (182,655)

Table 1: Mean function evaluations (over 10 runs) to attain each level for the GA and for RMHC. The number of function evaluations is sampled only every 1000 evaluations, so each value is actually an upper bound for an interval of length 1000. The standard errors are in parentheses.

3 (neither algorithm ever reached level 4). As can be seen, the time to reach levels 1 and 2 are roughly the same for the two algorithms, but the GA is much faster at reaching level 3. (In Forrest and Mitchell's experiments, RMHC was significantly faster than the GA at reaching this level in $R2$.) More detailed analyses of these results showed that hitchhiking in the GA is reduced significantly, and that the population is able to maintain instances of all the lowest-level schemas throughout each run. This is what the analysis of IGA would predict. However, on a number of runs the GA loses the schemas at level 3 after it has found them because of our weak selection scheme, whereas RMHC always holds on to a desired schema once it has been discovered. As was said above, it is necessary to balance the maintenance of independent samples with the sequestering of desired schemas. Without an additional mechanism such as elitism here, that balance is not achieved. Working out the details of the various balances necessary for the Royal Road functions is a topic of future work.

8. Conclusion

We have presented analyses of two algorithms, RMHC and the IGA, and have used the analyses to distill some general principles of when and how a genetic algorithm will outperform hill-climbing. We then presented some preliminary experimental results comparing the GA and RMHC on a modified Royal Road landscape. These analyses and results are a further step in achieving our original goals—to design the simplest class of fitness landscapes that will distinguish the GA from other search methods, and to rigorously characterize the general features of a fitness landscape that make it suitable for a GA.

Our modified Royal Road landscape $R3$, like $R1$ and $R2$, is not meant to be a realistic example of a problem to which one might apply a GA. Rather, it is meant to be an idealized problem 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. The work reported here is a further step in this direction.

Acknowledgments

Thanks to Stephanie Forrest for many significant contributions to this project and to this paper. Thanks also to Lashon Booker and Rick Riolo for helpful discussions in formulating the ideas in this paper, to Richard Palmer for suggesting the RMHC algorithm and for sharing his careful analysis with us, and to Greg Huber for his assistance on the analysis of the IGA. This project was supported by the Santa Fe Institute's Adaptive Computation Program and by grant B1992-46 from the Alfred P. Sloan Foundation.

References

[1] T. Bäck, F. Hoffseister, and H. Schwefel. A survey of evolution strategies. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA, 1991. Morgan Kaufmann.

[2] J. C. Bean. Genetics and random keys for sequencing and optimization. Technical Report 92-43, Dept. of Industrial and Operations Engineering, University of Michigan, Ann Arbor, MI 48109, 1992.

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

[4] L. Davis. Bit-climbing, representational bias, and test suite design. In R. K. Belew and L. B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Mateo, CA, 1991. Morgan Kaufmann.

[5] L. J. Eshelman. Personal communication.

[6] D. B. Fogel and J. W. Atmar. Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear search. *Biological Cybernetics*, 63:111–114, 1990.

[7] S. Forrest and G. Mayer-Kress. Genetic algorithms, nonlinear dynamical systems, and models of international security. In L. D. Davis, editor, *Handbook of Genetic Algorithms*, pages 166–185. Van Nostrand Reinhold, 1991.

[8] S. Forrest and M. Mitchell. What makes a problem hard for a genetic algorithm? Some anomalous results and their explanation. To appear in *Machine Learning*.

[9] S. Forrest and M. Mitchell. Relative building-block fitness and the building-block hypothesis. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*, San Mateo, CA, 1993. Morgan Kauffman.

[10] G. Huber. Personal communication.

[11] L. Ingber and B. Rosen. Genetic algorithms and very fast simulated reannealing: A comparison. *Mathl. Comput. Modelling*, 16(11):87–100, 1992.

[12] K. A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, The University of Michigan, Ann Arbor, MI, 1975.

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

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

[15] R. G. Palmer. Personal communication.

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

[17] N. N. Schraudolph and R. K. Belew. Dynamic parameter encoding for genetic algorithms. Technical Report CS 90-175, Computer Science and Engineering Deparment, University of California, San Diego, 1990.

[18] K. Shahookar and P. Mazumder. A genetic approach to standard cell placement using meta-genetic parameter optimization. *IEEE Transactions on Computer-Aided Design*, 9(5):500–511, 1990.

[19] R. Tanese. *Distributed Genetic Algorithms for Function Optimization.* PhD thesis, The University of Michigan, Ann Arbor, MI, 1989.

[20] S. W. Wilson. GA-easy does not imply steepest-ascent optimizable. In R. K. Belew and L. B. Booker, editors, *Proceedings of The Fourth International Conference on Genetic Algorithms*, San Mateo, CA, 1991. Morgan Kaufmann.