The results of Section 5 are of this sort. One sometimes hears the question: Is a GA more powerful than an equal population of hill-climbers, and if so, why? The answer seems to be yes, as noted. But why?

Perhaps the basic reason is that the hill-climber can get tripped up by an individual case, whereas on a problem that is GA-easy but not SAO, the GA is carried over such cases by the statistical power of the schema theorem. Note also that each of a population of hill-climbers is likely to get stuck somewhere, but in different places than its neighbors. In a problem like the one constructed in this paper, each hill-climber will at the same time be *correct* in a number of places (in the string), again at different places than its neighbors. Crossover would permit the good parts to be communicated and accumulated. But only the GA has crossover.

References

[1] D. E. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 6, pages 74-88. Morgan Kaufmann, Los Altos, CA, 1987.

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

5 IS THIS FAIR TO HILL-CLIMBERS?

One might suggest that the genetic algorithm has a built-in unfair advantage over a steepest-ascender since the GA works from a population of N initial strings whereas the steepest-ascender uses just one. What if there were N steepest-ascenders each starting with a random string and searching independently? The probability of finding the optimum somewhere among N such steepest-ascenders certainly increases with N. How does this compare with a population of size N under the GA?

Here we are on somewhat more complicated ground. Let us start by assuming we are trying to optimize the function F using N steepest-ascenders. Since a steepest-ascender starting at four of the eight possible starting strings for f_2 will reach the optimum of f_2, the probability of reaching the optimum of F, starting with a random $3m$-bit string, is $(4/8)^m$. The probability of *not* finding the optimum in such a string is then $[1-(.5)^m]$. If we have N random strings, the probability that the steepest-ascender will not find the optimum in *any* of the strings is $[1-(.5)^m]^N$. Consequently, the probability of finding at least one optimum in a population of N strings, written in terms of string length $n = 3m$, is $1 - [1 - (.5)^{n/3}]^N$.

Let us go one step further and suppose this probability equals some criterion value, say one-half. We could then set the above expression equal to one-half and solve for N. The result $N(n)$ would be the minimum size population for which the steepest-ascender would have probability .5 of finding the optimum. The manner in which $N(n)$ increased with n would suggest how the steepest-ascender's ability scaled with increasing problem size.

Our main purpose is to compare the N-string steepest-ascender with a GA having population N. As noted in Section 2, there is at present no formal analysis that would tell us, say, how large N has to be for the GA, or even the GA on a GA-easy problem such as this, to have probability .5 of finding the optimum. Current formal work on optimal population size, while important, does not yet directly address the question of convergence to the optimum. It is therefore necessary to proceed experimentally.

Our approach is to take a range of values for n, calculate the corresponding values of $N(n)$, then see if a GA using these values for N finds F's optimum substantially more often than half the time. Specifically, we chose values for n of 15, 18, 21, 24, 27, and 30 bits. The resulting values of $N(n)$ are shown in the table below; in each case, the GA's population size was the next lower even integer. The GA employed tournament selection (2-string tournaments), single-point crossover with probability 0.6, and point mutation with probability equal to the reciprocal of population size. For each value of N, the GA was run 10 times for 30 generations. The measure of the GA's success was the proportion of the 10 runs for which the optimum was present in the final generation. Also tabulated are measures of computational effort for the steepest-ascender and the GA. In both cases we simply use the number of string evaluations. For the GA, this is just N times the number of generations (30). For the steepest-ascender, note that to determine the best bit change for a single string requires n evaluations and the total "climb" for that string should average on the order of $n/2$ steps. Since there are N strings, the total number of evaluations is therefore about $(Nn^2)/2$.

n	$N(n)$	GA-success	GA-effort	SA-effort
15	21.83	0.8	600	2250
18	44.01	0.9	1320	7128
21	88.38	1.0	2640	19404
24	177.10	1.0	5280	50688
27	354.54	1.0	10620	129033
30	709.44	1.0	21240	318600

These results suggest that the GA is more effective, with much less effort, than the N-string steepest-ascender on function F. With exponentially rising population sizes, the steepest-ascender has only an even chance of finding the optimum, whereas the GA found the optimum with little difficulty in the small populations, and with near certainty in the larger ones. The GA in fact found the optimum in earlier generations as n increased.

The tentative implication of the results of this and the previous section is that within the class of functions that are GA-easy, and therefore presumably *actually* rather easy for a GA, there exists a quite large subclass of problems that are not easily solvable by "steepest-ascent", a basic hill-climbing technique. Furthermore, on these problems the GA would appear to be more powerful than an equal population of steepest-ascenders. Further research along this particular direction might compare the GA, again on a GA-easy but not SAO problem, with a population of more sophisticated hill-climbers that employ stochastic methods to escape local optima.

6 CONCLUSION

In this paper we compared the GA with simple but well-defined optimization methods on relatively simple problems. We found, perhaps somewhat surprisingly, that "GA-easy" problems can be difficult for these other methods, but not for the GA. The results add to our confidence that "GA-easy" is a non-trivial characterization. Formal analysis of genetic algorithm action on GA-easy problems should be relatively straightforward. If such problems are actually moderately interesting, the analysis is doubly motivated.

Our approach proposed that careful comparisons with methods that the GA beats could yield insight into the GA.

and determine the value of k and the bit value at k for which the highest fitness is found (breaking ties by some definite procedure). Call these k' and b', respectively. Next form a new string S' that is like S except that the bit at position k' is set to b'. Now replace S by S' and repeat this process until the fitness of S no longer increases. If for any initial string the fitness of the string finally arrived at equals the highest possible value of f we shall say that f is *steepest-ascent optimizable* (SAO).

Inspection of Figure 2 shows that f_2 is not steepest-ascent optimizable because 000, which is sub-optimal, will be reached starting from any of the four strings in the lower part of the figure. Let us next see if f_2 is GA-easy:

f(0**)	3/4	f(*0*)	3/4	f(**0)	3/4
f(1**)	5/4 <—	f(*1*)	5/4 <—	f(**1)	5/4 <—
f(00*)	1	f(0*0)	1	f(*00)	1
f(01*)	1/2	f(0*1)	1/2	f(*01)	1/2
f(10*)	1/2	f(1*0)	1/2	f(*10)	1/2
f(11*)	2 <—	f(1*1)	2 <—	f(*11)	2 <—

The function f_2 is indeed GA-easy. We have shown by this example that the proposition "GA-easy does not imply steepest-ascent optimizable" is true. This is stronger than our previous result, since apart from our intuition, it can be proved that functions that are bit-setting optimizable form a proper subset of functions that are steepest-ascent optimizable. Briefly, assume there exists a function f that is BSO but *not* SAO. If f is not SAO, it must have at least one local optimum (a maximum), where the steepest-ascender could get stuck. Now suppose the bit-setting optimizer happened to be started with initial string S precisely on one these local maxima. By definition of local maximum, any bit change in S would produce a string with non-increasing fitness, so that the bit-setting optimizer would also get stuck, contradicting the hypothesis that f is BSO. Thus BSO implies SAO. The converse is not true, since for example f_1 is SAO but not BSO.

4 LONG STRINGS

The previous main result, that GA-easy does not imply SAO, is certainly interesting but we need to check that it doesn't merely apply to short string problems like those used to prove it. Is the proposition also true for arbitrarily long strings? The answer is yes, as will now be shown.

Consider a function F of bit strings of length $n = 3m$ (m an integer >> 1). Further, let the value (fitness) of a string under F be the sum of the values obtained by applying f_2 to successive triples of the bits of the string. That is,

$$F(x_1, x_2, x_3, x_4,...,x_{3m}) = f_2(x_1, x_2, x_3) + f_2(x_4, x_5, x_6) + \ldots + f_2(x_{3m-2}, x_{3m-1}, x_{3m}).$$

Clearly, because f_2 is not steepest-ascent optimizable, F is also not SAO. To see this it is only necessary to assume a starting string for optimizing F in which at least one of the three-bit arguments to at least one of the f_2 terms above is a "bad" (unsuccessful) starting point for optimization of f_2. Then no matter how effective the optimizer on the rest of the string, those three bits will never arrive at 111, so the global optimum of F will not be reached.

Showing that F is GA-easy is more difficult. We shall again proceed by suggestive example, and not attempt a formal proof. First, shorten F, without loss of generality, to a function of six bits. We next need to be convinced that schema values are simply sums of the schema values for f_2, as in this example:

$$F(1***10) = f_2(1**) + f_2(*10).$$

But this equation is equivalent to the sum of the eight equations:

$$F(100010) = f_2(100) + f_2(010)$$
$$\ldots$$
$$F(111110) = f_2(111) + f_2(110),$$

all of which hold by definition of F. Therefore, we can say that F's schema fitnesses just sum f_2's schema fitnesses in the manner above.

Now, to demonstrate GA-easy, we must show that in every complete set of competing schemata, the schema with maximum fitness contains the optimum. In other words, e.g.,

$$F(1***11) > F(0***00),\ F(0***01),\ \ldots\ , F(0***11),$$

meaning that F(1***11) is greater than any schema value on the right. Consider the first inequality. Since we have just found that F's schema values are sums of f_2's, we can write

$$F(1***11) = f_2(1**) + f_2(*11)$$
and
$$F(0***00) = f_2(0**) + f_2(*00).$$

We are therefore asking whether

$$f_2(1**) + f_2(*11) > f_2(0**) + f_2(*00).$$

But because f_2 is GA-easy, this must be true. Since there was nothing unique about our choice of schemata, we conclude that F is GA-easy. Further, If F is GA-easy but not SAO, then we have demonstrated that the proposition "GA-easy does not imply steepest-ascent optimizable" is true for strings of arbitrary length.

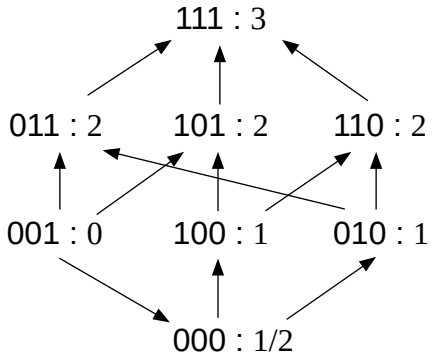

Figure 1: Function f_1

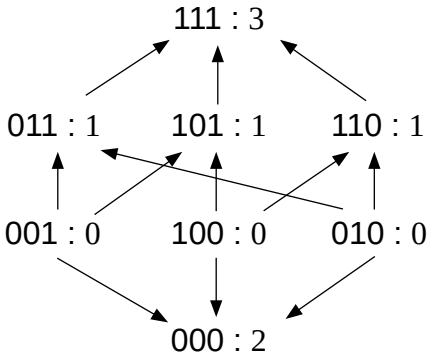

Figure 2: Function f_2

Let us now ask how hard f_1 is for a GA. Unfortunately, while we could frame a definition of "GA-optimizable" as being, say, optimizable by a GA starting from a random initial population of such-and-such a size, etc., it could not be usefully applied to a given problem because the precise action of a GA is not sufficiently well understood to allow prediction of the results. In effect, the only way currently to determine "GA-optimizable" is experimentally. However, optimization problems can be characterized in terms of their *deceptiveness* and there tends to be a correlation between deceptiveness and the difficulty in practice of optimization by a GA.

According to theory centered on the *schema theorem*, GAs "work well when *building blocks*—short, low-order schemata with above-average fitness values—combine to form optima or near-optima" [2]. Goldberg particularly has studied function-coding combinations that are deceptive to the GA in the sense that some or all short, low-order schemata whose fitness values are optimal within complete sets of *competing schemata* ([2], p. 39) do not in fact contain, or sample, the global optimum. For such functions, the schema theorem implies that the GA will be misled and may fail to find or retain the global optimum before converging to a suboptimal solution. Experimentally, this is the case (with some qualifications that are outside our present scope), and it can safely be said that deceptive problems will be difficult to optimize using a GA.

In this paper we focus on the converse proposition and suggest as a working hypothesis that the more *non*-deceptive a problem, the easier it will be to GA-optimize. In particular, let us define a problem to be *GA-easy* if the highest fitness schema in *every* complete set of competing schemata contains the optimum. Noting that our connection between GA-easy and GA-optimizable is reasonable but entirely heuristic, let us now ask whether of not f_1 is GA-easy. To do so we enumerate the problem's complete sets of competing schemata and see if the highest fitness schema in each set contains the optimum. The results are as follows (schema fitness values assume a uniform schema distribution, true on average at the start of search from a random initial population):

f(0**)	7/8	f(*0*)	7/8	f(**0)	9/8
f(1**)	2 <—	f(*1*)	2 <—	f(**1)	7/4 <—
f(00*)	1/4	f(0*0)	3/4	f(*00)	3/4
f(01*)	3/2	f(0*1)	1	f(*01)	1
f(10*)	3/2	f(1*0)	3/2	f(*10)	3/2
f(11*)	5/2 <—	f(1*1)	5/2 <—	f(*11)	5/2 <—

The maximum fitness schema (indicated by an arrow) in each competing set contains the optimum, 111, so that, according to our definition, f_1 is GA-easy. Earlier we found that f_1 is not bit-setting optimizable so that we have shown by example that the proposition "GA-easy does not imply bit-setting optimizable" is true.

This result is somewhat interesting, but not terribly surprising, since bit-setting optimization is a very weak method and should not often succeed. However, we now know that GA-easy, while making perhaps the strongest possible assumption about schema non-deceptiveness, nevertheless includes functions that cannot be bit-setting optimized. In the next section we develop a more interesting result.

3 STEEPEST-ASCENT OPTIMIZATION AND GA-EASY

Consider function f_2 (Figure 2) which is similar to f_1 except that it has a local maximum at 000. Like f_1, f_2 is not BSO. However, unlike f_1, f_2 cannot be optimized by a binary form of *steepest-ascent*. To define (binary) steepest-ascent, suppose we again start with an arbitary string S of length n and see which of the two possible settings of the kth bit gives the string a higher fitness. We perform this experiment, always starting with S, for each possible k,

From *Proceedings of The Fourth International Conference on Genetic Algorithms*,
R. K. Belew and L. B. Booker (eds.), San Mateo, CA: Morgan Kaufmann Publishers (1991)

GA-Easy Does Not Imply Steepest-Ascent Optimizable

Stewart W. Wilson
The Rowland Institute for Science
100 Cambridge Parkway
Cambridge, MA 02142
(wilson@think.com)

Abstract

It is shown that there are many functions which are GA-easy but not readily optimizable by a basic hill-climbing technique. The results, including a comparison of the genetic algorithm with a population of hill-climbers, provide insight into the operation of the GA and suggest further study of GA-easiness.

1 INTRODUCTION

Recent work beginning with Goldberg's [1] seminal paper on deceptiveness has aimed at characterizing functions (or more exactly, function-coding combinations) that are difficult to optimize using a genetic algorithm. The purpose of those investigations has been to explore the limits of GA power and, if possible, to exceed them via innovations in the algorithm. It would also be of interest to have good characterizations of functions that are *easy* for GAs, particularly if the functions cause difficulties for standard optimization methods. Identification of such functions and methods can give us a clearer picture of where and to what the GA is superior, and so aid in the practical selection of optimization techniques. But characterizing GA capability in terms of the methods it beats should also give insight into the still somewhat mysterious operation of the GA by suggesting—when we understand how the other methods work—which aspects of the GA could account for its superiority.

The present paper takes a few steps along this path by showing, through example problems, that there is a large class of functions which are *GA-easy* (according to a quite natural definition), but whose global optima cannot reliably be found by a basic hill-climbing technique. Further, an experiment is performed indicating that the GA is superior, on this class of functions, to a set of hill-climbers equal in number to the GA population size. These demonstrations contribute to the evidence that (1) the GA possesses local-optimum-avoiding capability exceeding that of basic hill-climbers, and (2) crossover gives the GA a distinct advantage over multiple independent local searches.

The rest of the paper is organized as follows. In the next section we warm up by defining what is perhaps the most elementary binary search technique, *bit-setting optimization*, and show that GA-easy does not imply bit-setting optimizable—in other words that there is a class of functions that are GA-easy but cannot be optimized by this technique. Then in Section 3 we strengthen this result by defining *steepest-ascent optimization* and showing that GA-easy does not imply steepest-ascent optimizable. In Section 4 we make sure this result applies to strings of arbitrary length. Then in Section 5 we demonstrate that on functions that are GA-easy but not steepest-ascent-optimizable, the GA should do better than an equal population of steepest-ascenders. Section 6 summarizes and comments on these results.

2 BIT-SETTING OPTIMIZATION AND GA-EASY

Let us consider an optimization problem in which the function f to be optimized is defined over a Hamming space so that points in the space are represented by binary strings. Suppose we begin with an arbitrary string S of length n and see which of the two possible settings of the kth bit gives the string a higher value for f (i.e., a higher fitness). We perform the same experiment, always starting with S, for each possible k, and then form a new string S' consisting of the better settings of each bit (if neither is better we pick one at random). If for any initial string S the value of $f(S')$ equals the highest possible value of f we shall say that f is *bit-setting optimizable* (BSO).

Now consider the function f_1 of three-bit strings shown in Figure 1. Each point of the lattice contains a bitstring together with the fitness (function value) for that string. The arrows indicate the direction of fitness change between points that differ by a single bit. By inspection, we see that f_1 is not bit-setting optimizable (NBSO) since if the optimizer happens to start with S = 001 it will end up with S' = 110, which is not the optimum. (Note in passing that if the fitnesses of 001 and 000 were changed to 1 and 0, respectively, the problem would be BSO.)

The results of Section 5 are of this sort. One sometimes hears the question: Is a GA more powerful than an equal population of hill-climbers, and if so, why? The answer seems to be yes, as noted. But why?

Perhaps the basic reason is that the hill-climber can get tripped up by an individual case, whereas on a problem that is GA-easy but not SAO, the GA is carried over such cases by the statistical power of the schema theorem. Note also that each of a population of hill-climbers is likely to get stuck somewhere, but in different places than its neighbors. In a problem like the one constructed in this paper, each hill-climber will at the same time be *correct* in a number of places (in the string), again at different places than its neighbors. Crossover would permit the good parts to be communicated and accumulated. But only the GA has crossover.

References

[1] D. E. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, chapter 6, pages 74-88. Morgan Kaufmann, Los Altos, CA, 1987.

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

5 IS THIS FAIR TO HILL-CLIMBERS?

One might suggest that the genetic algorithm has a built-in unfair advantage over a steepest-ascender since the GA works from a population of N initial strings whereas the steepest-ascender uses just one. What if there were N steepest-ascenders each starting with a random string and searching independently? The probability of finding the optimum somewhere among N such steepest-ascenders certainly increases with N. How does this compare with a population of size N under the GA?

Here we are on somewhat more complicated ground. Let us start by assuming we are trying to optimize the function F using N steepest-ascenders. Since a steepest-ascender starting at four of the eight possible starting strings for f_2 will reach the optimum of f_2, the probability of reaching the optimum of F, starting with a random $3m$-bit string, is $(4/8)^m$. The probability of *not* finding the optimum in such a string is then $[1-(.5)^m]$. If we have N random strings, the probability that the steepest-ascender will not find the optimum in *any* of the strings is $[1-(.5)^m]^N$. Consequently, the probability of finding at least one optimum in a population of N strings, written in terms of string length $n = 3m$, is $1 - [1 - (.5)^{n/3}]^N$.

Let us go one step further and suppose this probability equals some criterion value, say one-half. We could then set the above expression equal to one-half and solve for N. The result $N(n)$ would be the minimum size population for which the steepest-ascender would have probability .5 of finding the optimum. The manner in which $N(n)$ increased with n would suggest how the steepest-ascender's ability scaled with increasing problem size.

Our main purpose is to compare the N-string steepest-ascender with a GA having population N. As noted in Section 2, there is at present no formal analysis that would tell us, say, how large N has to be for the GA, or even the GA on a GA-easy problem such as this, to have probability .5 of finding the optimum. Current formal work on optimal population size, while important, does not yet directly address the question of convergence to the optimum. It is therefore necessary to proceed experimentally.

Our approach is to take a range of values for n, calculate the corresponding values of $N(n)$, then see if a GA using these values for N finds F's optimum substantially more often than half the time. Specifically, we chose values for n of 15, 18, 21, 24, 27, and 30 bits. The resulting values of $N(n)$ are shown in the table below; in each case, the GA's population size was the next lower even integer. The GA employed tournament selection (2-string tournaments), single-point crossover with probability 0.6, and point mutation with probability equal to the reciprocal of population size. For each value of N, the GA was run 10 times for 30 generations. The measure of the GA's success was the proportion of the 10 runs for which the optimum was present in the final generation. Also tabulated are measures of computational effort for the steepest-ascender and the GA. In both cases we simply use the number of string evaluations. For the GA, this is just N times the number of generations (30). For the steepest-ascender, note that to determine the best bit change for a single string requires n evaluations and the total "climb" for that string should average on the order of $n/2$ steps. Since there are N strings, the total number of evaluations is therefore about $(Nn^2)/2$.

n	$N(n)$	GA-success	GA-effort	SA-effort
15	21.83	0.8	600	2250
18	44.01	0.9	1320	7128
21	88.38	1.0	2640	19404
24	177.10	1.0	5280	50688
27	354.54	1.0	10620	129033
30	709.44	1.0	21240	318600

These results suggest that the GA is more effective, with much less effort, than the N-string steepest-ascender on function F. With exponentially rising population sizes, the steepest-ascender has only an even chance of finding the optimum, whereas the GA found the optimum with little difficulty in the small populations, and with near certainty in the larger ones. The GA in fact found the optimum in earlier generations as n increased.

The tentative implication of the results of this and the previous section is that within the class of functions that are GA-easy, and therefore presumably *actually* rather easy for a GA, there exists a quite large subclass of problems that are not easily solvable by "steepest-ascent", a basic hill-climbing technique. Furthermore, on these problems the GA would appear to be more powerful than an equal population of steepest-ascenders. Further research along this particular direction might compare the GA, again on a GA-easy but not SAO problem, with a population of more sophisticated hill-climbers that employ stochastic methods to escape local optima.

6 CONCLUSION

In this paper we compared the GA with simple but well-defined optimization methods on relatively simple problems. We found, perhaps somewhat surprisingly, that "GA-easy" problems can be difficult for these other methods, but not for the GA. The results add to our confidence that "GA-easy" is a non-trivial characterization. Formal analysis of genetic algorithm action on GA-easy problems should be relatively straightforward. If such problems are actually moderately interesting, the analysis is doubly motivated.

Our approach proposed that careful comparisons with methods that the GA beats could yield insight into the GA.

and determine the value of k and the bit value at k for which the highest fitness is found (breaking ties by some definite procedure). Call these k' and b', respectively. Next form a new string S' that is like S except that the bit at position k' is set to b'. Now replace S by S' and repeat this process until the fitness of S no longer increases. If for any initial string the fitness of the string finally arrived at equals the highest possible value of f we shall say that f is *steepest-ascent optimizable* (SAO).

Inspection of Figure 2 shows that f_2 is not steepest-ascent optimizable because 000, which is sub-optimal, will be reached starting from any of the four strings in the lower part of the figure. Let us next see if f_2 is GA-easy:

f(0**)	3/4	f(*0*)	3/4	f(**0)	3/4
f(1**)	5/4 <—	f(*1*)	5/4 <—	f(**1)	5/4 <—
f(00*)	1	f(0*0)	1	f(*00)	1
f(01*)	1/2	f(0*1)	1/2	f(*01)	1/2
f(10*)	1/2	f(1*0)	1/2	f(*10)	1/2
f(11*)	2 <—	f(1*1)	2 <—	f(*11)	2 <—

The function f_2 is indeed GA-easy. We have shown by this example that the proposition "GA-easy does not imply steepest-ascent optimizable" is true. This is stronger than our previous result, since apart from our intuition, it can be proved that functions that are bit-setting optimizable form a proper subset of functions that are steepest-ascent optimizable. Briefly, assume there exists a function f that is BSO but *not* SAO. If f is not SAO, it must have at least one local optimum (a maximum), where the steepest-ascender could get stuck. Now suppose the bit-setting optimizer happened to be started with initial string S precisely on one these local maxima. By definition of local maximum, any bit change in S would produce a string with non-increasing fitness, so that the bit-setting optimizer would also get stuck, contradicting the hypothesis that f is BSO. Thus BSO implies SAO. The converse is not true, since for example f_1 is SAO but not BSO.

4 LONG STRINGS

The previous main result, that GA-easy does not imply SAO, is certainly interesting but we need to check that it doesn't merely apply to short string problems like those used to prove it. Is the proposition also true for arbitrarily long strings? The answer is yes, as will now be shown.

Consider a function F of bit strings of length $n = 3m$ (m an integer >> 1). Further, let the value (fitness) of a string under F be the sum of the values obtained by applying f_2 to successive triples of the bits of the string. That is,

$$F(x_1, x_2, x_3, x_4,...,x_{3m}) = f_2(x_1, x_2, x_3) + f_2(x_4, x_5, x_6) + \ldots + f_2(x_{3m-2}, x_{3m-1}, x_{3m}).$$

Clearly, because f_2 is not steepest-ascent optimizable, F is also not SAO. To see this it is only necessary to assume a starting string for optimizing F in which at least one of the three-bit arguments to at least one of the f_2 terms above is a "bad" (unsuccessful) starting point for optimization of f_2. Then no matter how effective the optimizer on the rest of the string, those three bits will never arrive at 111, so the global optimum of F will not be reached.

Showing that F is GA-easy is more difficult. We shall again proceed by suggestive example, and not attempt a formal proof. First, shorten F, without loss of generality, to a function of six bits. We next need to be convinced that schema values are simply sums of the schema values for f_2, as in this example:

$$F(1***10) = f_2(1**) + f_2(*10).$$

But this equation is equivalent to the sum of the eight equations:

$$F(100010) = f_2(100) + f_2(010)$$
$$\ldots$$
$$F(111110) = f_2(111) + f_2(110),$$

all of which hold by definition of F. Therefore, we can say that F's schema fitnesses just sum f_2's schema fitnesses in the manner above.

Now, to demonstrate GA-easy, we must show that in every complete set of competing schemata, the schema with maximum fitness contains the optimum. In other words, e.g.,

$$F(1***11) > F(0***00),\ F(0***01),\ \ldots\ , F(0***11),$$

meaning that F(1***11) is greater than any schema value on the right. Consider the first inequality. Since we have just found that F's schema values are sums of f_2's, we can write

$$F(1***11) = f_2(1**) + f_2(*11)$$
and
$$F(0***00) = f_2(0**) + f_2(*00).$$

We are therefore asking whether

$$f_2(1**) + f_2(*11) > f_2(0**) + f_2(*00).$$

But because f_2 is GA-easy, this must be true. Since there was nothing unique about our choice of schemata, we conclude that F is GA-easy. Further, If F is GA-easy but not SAO, then we have demonstrated that the proposition "GA-easy does not imply steepest-ascent optimizable" is true for strings of arbitrary length.

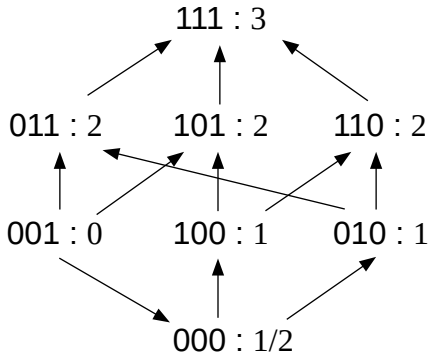

Figure 1: Function f_1

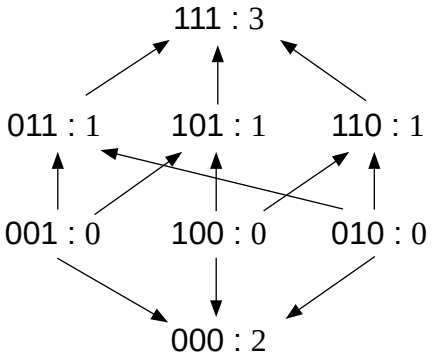

Figure 2: Function f_2

Let us now ask how hard f_1 is for a GA. Unfortunately, while we could frame a definition of "GA-optimizable" as being, say, optimizable by a GA starting from a random initial population of such-and-such a size, etc., it could not be usefully applied to a given problem because the precise action of a GA is not sufficiently well understood to allow prediction of the results. In effect, the only way currently to determine "GA-optimizable" is experimentally. However, optimization problems can be characterized in terms of their *deceptiveness* and there tends to be a correlation between deceptiveness and the difficulty in practice of optimization by a GA.

According to theory centered on the *schema theorem*, GAs "work well when *building blocks*—short, low-order schemata with above-average fitness values—combine to form optima or near-optima" [2]. Goldberg particularly has studied function-coding combinations that are deceptive to the GA in the sense that some or all short, low-order schemata whose fitness values are optimal within complete sets of *competing schemata* ([2], p. 39) do not in fact contain, or sample, the global optimum. For such functions, the schema theorem implies that the GA will be misled and may fail to find or retain the global optimum before converging to a suboptimal solution. Experimentally, this is the case (with some qualifications that are outside our present scope), and it can safely be said that deceptive problems will be difficult to optimize using a GA.

In this paper we focus on the converse proposition and suggest as a working hypothesis that the more *non*-deceptive a problem, the easier it will be to GA-optimize. In particular, let us define a problem to be *GA-easy* if the highest fitness schema in *every* complete set of competing schemata contains the optimum. Noting that our connection between GA-easy and GA-optimizable is reasonable but entirely heuristic, let us now ask whether of not f_1 is GA-easy. To do so we enumerate the problem's complete sets of competing schemata and see if the highest fitness schema in each set contains the optimum. The results are as follows (schema fitness values assume a uniform schema distribution, true on average at the start of search from a random initial population):

f(0**)	7/8	f(*0*)	7/8	f(**0)	9/8
f(1**)	2 <—	f(*1*)	2 <—	f(**1)	7/4 <—
f(00*)	1/4	f(0*0)	3/4	f(*00)	3/4
f(01*)	3/2	f(0*1)	1	f(*01)	1
f(10*)	3/2	f(1*0)	3/2	f(*10)	3/2
f(11*)	5/2 <—	f(1*1)	5/2 <—	f(*11)	5/2 <—

The maximum fitness schema (indicated by an arrow) in each competing set contains the optimum, 111, so that, according to our definition, f_1 is GA-easy. Earlier we found that f_1 is not bit-setting optimizable so that we have shown by example that the proposition "GA-easy does not imply bit-setting optimizable" is true.

This result is somewhat interesting, but not terribly surprising, since bit-setting optimization is a very weak method and should not often succeed. However, we now know that GA-easy, while making perhaps the strongest possible assumption about schema non-deceptiveness, nevertheless includes functions that cannot be bit-setting optimized. In the next section we develop a more interesting result.

3 STEEPEST-ASCENT OPTIMIZATION AND GA-EASY

Consider function f_2 (Figure 2) which is similar to f_1 except that it has a local maximum at 000. Like f_1, f_2 is not BSO. However, unlike f_1, f_2 cannot be optimized by a binary form of *steepest-ascent*. To define (binary) steepest-ascent, suppose we again start with an arbitary string S of length n and see which of the two possible settings of the kth bit gives the string a higher fitness. We perform this experiment, always starting with S, for each possible k,

From *Proceedings of The Fourth International Conference on Genetic Algorithms*,
R. K. Belew and L. B. Booker (eds.), San Mateo, CA: Morgan Kaufmann Publishers (1991)

GA-Easy Does Not Imply Steepest-Ascent Optimizable

Stewart W. Wilson
The Rowland Institute for Science
100 Cambridge Parkway
Cambridge, MA 02142
(wilson@think.com)

Abstract

It is shown that there are many functions which are GA-easy but not readily optimizable by a basic hill-climbing technique. The results, including a comparison of the genetic algorithm with a population of hill-climbers, provide insight into the operation of the GA and suggest further study of GA-easiness.

1 INTRODUCTION

Recent work beginning with Goldberg's [1] seminal paper on deceptiveness has aimed at characterizing functions (or more exactly, function-coding combinations) that are difficult to optimize using a genetic algorithm. The purpose of those investigations has been to explore the limits of GA power and, if possible, to exceed them via innovations in the algorithm. It would also be of interest to have good characterizations of functions that are *easy* for GAs, particularly if the functions cause difficulties for standard optimization methods. Identification of such functions and methods can give us a clearer picture of where and to what the GA is superior, and so aid in the practical selection of optimization techniques. But characterizing GA capability in terms of the methods it beats should also give insight into the still somewhat mysterious operation of the GA by suggesting—when we understand how the other methods work—which aspects of the GA could account for its superiority.

The present paper takes a few steps along this path by showing, through example problems, that there is a large class of functions which are *GA-easy* (according to a quite natural definition), but whose global optima cannot reliably be found by a basic hill-climbing technique. Further, an experiment is performed indicating that the GA is superior, on this class of functions, to a set of hill-climbers equal in number to the GA population size. These demonstrations contribute to the evidence that (1) the GA possesses local-optimum-avoiding capability exceeding that of basic hill-climbers, and (2) crossover gives the GA a distinct advantage over multiple independent local searches.

The rest of the paper is organized as follows. In the next section we warm up by defining what is perhaps the most elementary binary search technique, *bit-setting optimization*, and show that GA-easy does not imply bit-setting optimizable—in other words that there is a class of functions that are GA-easy but cannot be optimized by this technique. Then in Section 3 we strengthen this result by defining *steepest-ascent optimization* and showing that GA-easy does not imply steepest-ascent optimizable. In Section 4 we make sure this result applies to strings of arbitrary length. Then in Section 5 we demonstrate that on functions that are GA-easy but not steepest-ascent-optimizable, the GA should do better than an equal population of steepest-ascenders. Section 6 summarizes and comments on these results.

2 BIT-SETTING OPTIMIZATION AND GA-EASY

Let us consider an optimization problem in which the function f to be optimized is defined over a Hamming space so that points in the space are represented by binary strings. Suppose we begin with an arbitrary string S of length n and see which of the two possible settings of the kth bit gives the string a higher value for f (i.e., a higher fitness). We perform the same experiment, always starting with S, for each possible k, and then form a new string S' consisting of the better settings of each bit (if neither is better we pick one at random). If for any initial string S the value of $f(S')$ equals the highest possible value of f we shall say that f is *bit-setting optimizable* (BSO).

Now consider the function f_1 of three-bit strings shown in Figure 1. Each point of the lattice contains a bitstring together with the fitness (function value) for that string. The arrows indicate the direction of fitness change between points that differ by a single bit. By inspection, we see that f_1 is not bit-setting optimizable (NBSO) since if the optimizer happens to start with S = 001 it will end up with S' = 110, which is not the optimum. (Note in passing that if the fitnesses of 001 and 000 were changed to 1 and 0, respectively, the problem would be BSO.)