Long Path Problems

Jeffrey Horn, David E. Goldberg, and Kalyanmoy Deb

PREPRINT
(camera-ready)

as accepted for final publication in
Parallel Problem Solving from Nature–PPSN III
International Conference on Evolutionary Computation
The Third Conference on Parallel Problem Solving from Nature
Jerusalem, Israel, October 1994
Proceedings
volume 866 of *Lecture Notes in Computer Science*
Edited by Y. Davidor, H.-P. Schwefel, and R. Männer
Published by Springer-Verlag
Berlin, Germany
(pp. 149–158)

Long Path Problems

Jeffrey Horn[1], David E. Goldberg[1], and Kalyanmoy Deb[2] *

[1] Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 South Mathews Avenue, Urbana, IL 61801-2996, USA, (e-mail: jeffhorn@uiuc.edu, deg@uiuc.edu)
[2] Department of Mechanical Engineering, Indian Institute of Technology, Kanpur, UP, PIN 208016, India, (e-mail: deb@iitk.ernet.in)

Abstract. We demonstrate the interesting, counter-intuitive result that simple paths to the global optimum can be so long that climbing the path is intractable. This means that a unimodal search space, which consists of a single hill and in which each point in the space is on a simple path to the global optimum, can be difficult for a hillclimber to optimize. Various types of hillclimbing algorithms will make constant progress toward the global optimum on such long path problems. They will continuously improve their *best found* solutions, and be guaranteed to reach the global optimum. Yet we cannot wait for them to arrive. Early experimental results indicate that a genetic algorithm (GA) with crossover alone outperforms hillclimbers on one such long path problem. This suggests that GAs can climb hills faster than hillclimbers by exploiting *building blocks* when they are present. Although these problems are artificial, they introduce a new dimension of problem difficulty for evolutionary computation. Path length can be added to the ranks of multimodality, deception/misleadingness, noise, variance, etc., as a measure of fitness landscapes and their amenability to evolutionary optimization.

1 Introduction

In this paper we present a class of problems designed to be difficult for randomized search procedures that exploit local search space information. In particular, these problems challenge hillclimbers and mutation algorithms. The problems are difficult not because they contain local optima that capture the attention of the search procedure. Indeed, these problems are unimodal and "easy" in the sense that the simplest hillclimber will always find the global optimum, no matter where in the space it starts searching. Rather, these problems are difficult for hillclimbers because the only "path" up the hill to the global optimum is very

* The first author acknowledges support by NASA under contract number NGT-50873. The second and third authors acknowledge the support provided by the US Army under Contract DASG60-90-C-0153 and by the National Science Foundation under Grant ECS-9022007. We thank Joseph Culberson for pointing us to the literature on difference-preserving codes. Both Culberson and Greg J.E. Rawlins participated in early discussions with the second author on hillclimbing hard problems.

long and narrow. The length of the path grows exponentially with the size of the (binary) problem, ℓ.

Constructing hard problems for a class of algorithms is part of a recognized methodology for analyzing, understanding, and bounding complex algorithms. Three types of difficulties for hillclimbers are well-known [3]:

- Isolation or *needle-in-a-haystack* (NIAH)
- Full deception
- Multimodality

All three types of difficulties are also known to pose stiff challenges to genetic algorithms (GAs), and have been used to understand GAs [2, 1].

We propose a fourth type of problem that specifically targets the hillclimber's use of local information. We construct a path that leads a hillclimber to the global optimum, but on a path that is so long that for large problems ($\ell \gg 20$) we simply cannot wait for the algorithm to terminate. We might sit and watch our hillclimber making constant progress for years on an 80-bit problem.

We are motivated to construct and analyze these long path problems by several observations:

- Computational difficulty of an apparently easy problem
- Utility in analyzing hillclimbers and mutation algorithms
- Possible distinguishing problem for GAs versus hillclimbers
- Possible distinguishing problem for crossover versus mutation

These problems represent a dimension of difficulty that has been less obvious than noise, deception, local optima, etc. Unimodal, and with every point on a path to the global optimum, these problems are still intractable for hillclimbers of bounded step size. They might also be intractable for various kinds of mutation algorithms, and for algorithms incorporating recombination, such as GAs with high crossover rates. If they do offer as stiff a challenge to recombinative algorithms as they do to hillclimbers (and apparently to mutation algorithms as well), then we have found yet another class of GA-difficulty. If, on the other hand, the recombinative GA can perform significantly better (faster) on these problems than local-search procedures, then we have found a class of functions that distinguish these two types of algorithms. From this we can learn something about their different strengths and weaknesses relative to each other [6].

2 Definitions: Optima, Paths, and Algorithms

We use the paradigm of a *fitness landscape*, which consists of a search space, a metric, and a scalar fitness function $f(s)$ defined over elements s of the search space S. Assuming the goal is to maximize fitness, we can imagine the globally best solutions (the global optima, or "globals") as "peaks" in the search space. For the purposes of this paper, we define local optimality as follows. We first assume a real-valued, scalar fitness function $f(s)$ over fixed length ℓ-bit binary strings s, $f(s) \in \Re$. Without loss of generality, we assume f is to be maximized. A *local optimum* in a discrete search space S is a point, or region, with fitness function value strictly greater than those of *all* of its nearest neighbors. By

"region" we mean a set of interconnected points of equal fitness. That is, we treat as a single optimum the set of points related by the transitive closure of the nearest-neighbor relation such that all points are of equal fitness. This definition allows us to include flat plateaus and ridges as single optima, and to treat a flat fitness function as having no local optima. The "nearest neighbor" relation assumes a metric on the search space, call it d, where $d(s_1, s_2) \in \Re$ is the metric's distance between points s_1 and s_2. Then the nearest neighbors of a point s' are all points $s \in S, s \neq s'$ such that $d(s', s) \leq k$, for some neighborhood radius k. In this paper we use only *Hamming distance* (number of bit positions at which two binary strings differ) as the metric, and assume $k = 1$. Thus a point s with fitness $f(s)$ greater than that of all its immediate neighbors (strings differing from s in only one bit position) is a local optimum.

Although the term "hillclimber" denotes a specific set of algorithms for some researchers, we will use the term more loosely here to describe *all* algorithms that emphasize exploration of a local neighborhood around the current solution. In particular, we assume that our *hillclimber* explores a neighborhood of radius k-bits much better than it explores outside that neighborhood. Therefore, our hillclimber is much more likely to take "steps" of size $\leq k$ than it is to take steps $> k$. Examples of hillclimbers include steepest ascent [12], next ascent [4], and random mutation [10, 1, 8, 9]. GAs with high selective pressure and low crossover rates also exhibit strong local hillclimbing.

A path P of step size k is a sequence of points p_i such that any two points adjacent on the path are at most k-bits apart, and any two points not adjacent on the path are more than k-bits apart:

$$\forall p_i, p_j \in P,\ d(p_i, p_j) \begin{cases} \leq k, \text{ if } |i - j| = 1 \\ \\ > k, \text{ otherwise.} \end{cases}$$

where $d(p_i, p_j)$ is the Hamming distance between the two points and p_i is the ith point on the path P. Thus, our hillclimbing algorithm that takes steps of size $\leq k$ would tend to follow the path without taking any "shortcuts". The step size here is important, because we want to lead the algorithm up the path, but to make the path wind through the search space as much as possible, we will need to fold that path back many times. Earlier portions of the path may thus pass quite closely to later portions, within $k + 1$ bits, so we must assume that our algorithm has a very small, if not zero, probability of taking steps of size $> k$.

3 A Simple Construction: the Root2path

In this section we construct a long path that is not optimally long, but does have length exponential in ℓ, the size (order, or dimension) of the problem, and is simple in its construction. We call it the *Root2path*. Here we choose the smallest step size $k = 1$ to illustrate the construction. Each point on the path must be exactly one bit different from the point behind it and the point ahead of it, while also being at least two bits away from any other point on the path.

The construction of the path is intuitive. If we have a Root2path of dimension ℓ, call it P_ℓ, we can basically double it by moving up two dimensions to $\ell+2$ as follows. Make two copies of P_ℓ, say copy00 and copy11. Prepend "00" to each point in copy00, and "11" to each point in copy11[3]. Now each point in copy00 is at least two bits different from all points in copy11. Also, copy00 and copy11 are both paths of step size one and of dimension $\ell+2$. Furthermore, the endpoint of copy00 and the endpoint of copy11 differ only in their first two bit positions ("00" versus "11"). By adding a *bridge point* that is the same as the endpoint of copy00 but with "01" in the first two bit positions instead of "00", we can connect the end of copy00 and the end of copy11. Reversing the sequence of points in copy11, we concatenate copy00, the bridge point, and Reverse[copy11] to create the Root2path of dimension $\ell+2$, call it $P_{\ell+2}$, and length essentially twice that of P_ℓ:

$$|P_{\ell+2}| = 2\,|P_\ell| + 1 \tag{1}$$

As for the dimensions between the doublings, which are all of odd order, we can simply use the path P_ℓ by adding a "0" to each point in P_ℓ to create $P_{\ell+1}$. If $|P_\ell|$ is exponentional in ℓ, then $|P_{\ell+1}|$ is exponential in $\ell+1$.

For the base case $\ell = 1$, we have only two points in the search space: 0 and 1. We put them both on the Root2path $P_1 = \{0, 1\}$, where 0 is the beginning and 1 is the end (i.e., the global optimum).

With every other incremental increase in dimension, we have an effective doubling of the path length. Solving the recurrence relation in Equation 1, with $|P_1| = |P_2| = 2$, we obtain the path length as a function of ℓ:

$$|P_\ell| = 3 * 2^{\lfloor(\ell-1)/2\rfloor} - 1. \tag{2}$$

Path length increases in proportion to $2^{\ell/2}$ or $(\sqrt{2})^\ell$ and thus grows exponentially in ℓ with base ≈ 1.414, an ever-decreasing fraction of the total space 2^ℓ.

Since the path takes up only a small fraction of the search space for large ℓ, the entire Root2path approaches a Needle-in-a-Haystack (NIAH) problem as ℓ grows. We are interested in how long a hillclimber takes to climb a path, not how long it takes to find it. We therefore slope the remainder of the search space (i.e., all points not on the Root2path) towards the beginning of the path. The construction of the Root2path makes it easy to do this. Since the first point on the path is the all-zeroes point, we assign fitness values to all points *off* the path according to a function of *unitation*[4]. The fewer ones in a string, the higher its fitness. Thus, most of the search space should lead the hillclimber to the all-zeroes point, in at most ℓ steps. We call this landscape feature the *nilness*[5] *slope*. Together, the path P_ℓ and the nilness slope form a single *hill*, making the search space unimodal. A deterministic hillclimber, started *anywhere* in the space, is guaranteed to find the global optimum.

[3] Thus the point "001" in P_3 becomes "00001" in copy00 and "11001" in copy11.

[4] The unitation $u(s)$ of a string s is equal to the number of ones in s. For example, $u(\text{"0110110"}) = 4$.

[5] The nilness of a string s is simply $n(s) = \ell - u(s)$ (i.e., the number of zeroes in s).

To find the total number of "steps" from the bottom of the hill (the all-ones point) to the top (the global optimum), we add the "height" of the nilness slope, which is simply ℓ, to the path length (Equation 2), and substract one for the all zeroes point, which is on both the path and the slope:

$$\text{Hill-Height}(\ell) = 3 * 2^{\lfloor(\ell-1)/2\rfloor} + \ell - 2 \tag{3}$$

The recursive construction of the Root2path is illustrative, but inefficient for fitness evaluations in our simulations below. In Figure 1 we present pseudocode for a much more efficient *decoding algorithm*[6]. Note that the recursion in the pseudocode is (optimizable) tail recursion. The function *HillPosition* can be used directly as the objective fitness function for optimization (maximization)[7].

```
PathPosition[str] := CASE
                (str == "0") RETURN 0;  /* First step on path.  */
                (str == "1") RETURN 1;  /* Second step on path. */
                (str == "00—rest-of-string")  /* On 1st half of path.  Recur. */
                          RETURN PathPosition[rest-of-string];
                (str == "11—rest-of-string")  /* On 2nd half of path.  Recur. */
                          RETURN 3*2^Floor[(Length[str]-1)/2] - 2 -
                                PathPosition[rest-of-string];
                (str == "101" OR "1011—all-zeroes") /* At bridge pt. (halfway) */
                          RETURN  3*2^(Floor[(Length(str)-1)/2] - 1) - 1);
                OTHERWISE    RETURN false;      /* str is NOT ON PATH.  */

HillPosition[str] := IF (PathPosition[str])       /* If str is on path, */
                       /* return path position plus problem length (slope). */
                      THEN RETURN PathPosition[str] + Length[str];
                      ELSE RETURN Nilness[str];   /*  Else return position on slope, */
                                          /*   which is number of zeroes.    */
```

Fig. 1. The decoding algorithm for the Root2path function, $k = 1$.

4 Simulation Results

4.1 The Long Road for Hillclimbers

We restrict ourselves to five simple algorithms, analyzed in [10]:

- Steepest ascent hillclimber (SAHC)
- Next ascent hillclimber (NAHC)
- Fixed rate mutation algorithm (Mut)
- Mutation with steepest ascent (Mut+SAHC)
- Mutation with next ascent (Mut+NAHC)

All five algorithms work with one point, s, at a time. The first point, s_0, is chosen

[6] In the literature on coding theory, the development of such algorithms is an important followup to the existence proofs and constructions of long paths [11].

[7] Note that the decoding algorithm assumes odd(ℓ).

randomly. Thereafter, s_{n+1} is found by looking at a neighborhood of s_n. With steepest ascent, all of s_n's neighbors are compared with s_n. The point within that neighborhood that has the highest fitness becomes s_{n+1}. If $s_{n+1} = s_n$, then steepest ascent has converged to a local (perhaps global) optimum. Next ascent is similar to steepest ascent, the only difference being that next ascent compares neighbors to s_n in some fixed order, taking the first neighbor with fitness greater than s_n to be s_{n+1}. In our runs, we assume SAHC and NAHC explore a neighborhood of radius one (step size $k = 1$).

Mutation (Mut) flips each bit in s_n with probability p_m, independently. The resulting string is compared with s_n. If its fitness is greater, the mutated string becomes s_{n+1}, otherwise s_n does. Mühlenbein [10], and other researchers, have found that a bitwise mutation rate $p_m = 1/\ell$ is optimal for many classes of problems. The only mutation rate we use here is $1/\ell$.

The other two hillclimbing algorithms we run are combinations of mutation with steepest ascent (Mut+SAHC) and next ascent (Mut+NAHC). These combinations are implemented by simply mutating s_n, and allowing either next ascent or steepest ascent hillclimbing to explore the neighborhood around the mutated string. The resulting string, either the originally mutated string or a better neighbor, is then compared with s_n for the choice of s_{n+1}.

We test on Root2paths of dimension $\ell = 1$ to 20. We only consider paths of step size $k = 1$. In Figure 2, we plot the performance of the five hillclimbers. For each problem size ℓ, we ran each algorithm at random starting points. The plotted points are averages over five runs. We measure performance as the number of iterations (of the hillclimber's main update loop) required to reach the global optimum (i.e., the number of points in the sequence s_n). Note that this is less than the number of fitness evaluations used. For example, since steepest ascent searches a neighborhood of radius one everytime it updates s_n, its number of fitness evaluations is ℓ times the number of iterations.

As Figure 2 illustrates, at least three of the algorithms perform exponentially worse as ℓ increases. Mutation by itself tends to spend a long time finding the next step on the path. Steepest and next ascent tend to follow the path step by step. Steepest ascent with mutation exhibits linear performance, however. The superiority of this hillclimbing variant is explained by its tendency to take steps of size two. Mutation with $p_m = 1/\ell$ takes a single-bit step, in expectation. Steepest ascent then explores the immediate neighborhood around the mutated point. Since the Root2path contains many shortcuts of step size two[8], steepest ascent with mutation is able to skip several large segments of the path.

To force Mut+SAHC to stay on an exponentially long path, we clearly need paths of greater step size. A simple way of building such a path is to extend the construction of the Root2path as follows. For a step size of $k = 2$, we should double the path every *third* increment in dimension. Thus we prepend "111" and "000" to copies of P_ℓ to get $P_{\ell+3}$. We now have a path that grows in length in proportion to $2^{\ell/3}$. We could call these *CubeRoot2paths*. We can generalize to step size k, to get paths that grow as $2^{\ell/(k+1)}$, exponential in ℓ for $k \ll \ell$.

[8] For example, the beginning and end of the Root2path are only two bits apart!

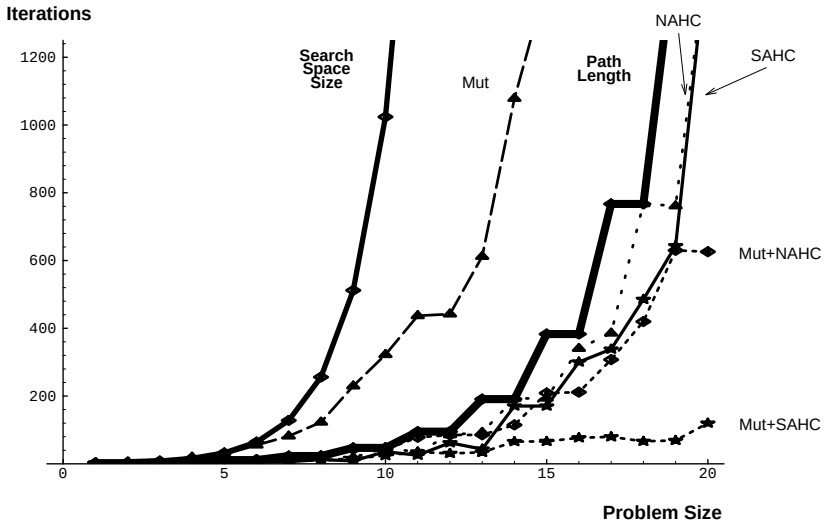

Fig. 2. The performance of five hillclimbing algorithms as a function of the problem size ℓ. Performance is averaged over five runs.

4.2 Crossover's Success

It is not obvious that the long path problems are amenable to recombinative search. From its inductive construction, it is clear that the Root2path has structure. The same basic subsequences of steps are used over and over again on larger scales, resulting in fractal self-similarity. But we do not know if such structure induces building blocks exploitable by crossover [5], as we have not yet applied a schema analysis to these functions. However, certain substrings such as "11" and "1011" (in certain positions) are common to many points on the path. And our first results, summarized in Table 1, indicate that a GA with crossover alone is an effective strategy for climbing long hills in a short time.

In Table 1 we compare three hillclimbers to a GA on three different size Root2path problems (all with stepsize $k = 1$). The GA is a simple, *generational* GA, using single point crossover with probability $p_c = 0.9$, no mutation ($p_m = 0$), binary tournament selection, and the population sizes indicated in Table 1. Random mutation hill climbing (RMHC) is described in [1, 8, 9]. RMHC is like mutation-only (Mut) above, except that one and only one bit flip takes place. Thus, RMHC starts with a random string s_0, and updates s_n by randomly flipping one bit in s_n to form s'_n. If s'_n is better than, or equal to s_n, then s'_n becomes s_{n+1}, otherwise s_n becomes s_{n+1}.

In [1, 8, 9], the authors found that RMHC optimized Royal Road (RR)

PERFORMANCE ON **Root2path**, stepsize $k = 1$			
	Number of Functions Evaluations to Global Optimum mean (std. dev.)		
	problem size		
algorithm	$\ell = 29$	$\ell = 39$	$\ell = 49$
SAHC	1,425,005 (789)	>15,000,000 (0)	>40,000,000 (0)
NAHC	1,359,856 (247)	>15,000,000 (0)	>40,000,000 (0)
RMHC	1,382,242 (112,335)	>15,000,000 (0)	>40,000,000 (0)
GA (Xover only)	25,920 (5830)	75,150 (53,500)	151,740 (140,226)
Pop. size	4000	5000	6000
Height of Hill (path + slope)	49,179 steps	1,572,901 steps	50,331,695 steps

Table 1. GA versus hillclimbers: results from 10 runs of each algorithm.

functions faster than SAHC, NAHC, and the GA. On the Root2path problems, however, the GA[9] seems to outperform the three hillclimbers, and RMHC apparently loses out to the *deterministic*[10] hillclimbers. Comparing the long path problems to the RR functions is not within the scope of this paper[11]. But our early results pointing to superior performance by crossover might be of particular interest to those looking at when GAs outperform hillclimbers [8, 9]. One answer might be "on a hill" (at least, a certain kind of hill).

5 Discussion

5.1 Extension: Longer Paths

It is certainly possible to construct paths longer than the Root2path. A *Fibonacci path* of step size $k = 1$ is constructed inductively like the Root2path. Our inductive step goes as follows. Given a Fibonacci path F_ℓ of dimension ℓ, and a Fibonacci path $F_{\ell+1}$ of dimension $\ell+1$, we construct a path of dimension $\ell+2$ by skipping a dimension and then doubling F_ℓ as with the Root2path. But rather than simply adding a single "bridge" point to connect the two copies of

[9] For the GA we estimate the number of fitness evaluations (to find the global) as numgens $*$ popsize $* p_c$, where *numgens* is the number of generations until the global optimum **first** appears. If the GA prematurely converges (i.e., to a non-optimal point), it is restarted **without** resetting *numgens*. Thus *numgens* is cummulative over multiple (unsuccessful) GA runs.

[10] SAHC and NAHC are deterministic in the sense that they are guaranteed to find the global optimum of the Root2path within a maximum time. The GA and RMHC, on the other hand, are stochastic.

[11] It is interesting to note, however, that both landscapes are unimodal (by our definition of local optimality).

F_ℓ, we use the path $F_{\ell-1}$ to connect them. We know we can use $F_{\ell-1}$ to connect the two copies of F_ℓ since $F_{\ell+1}$ is composed of an F_ℓ path coupled with an $F_{\ell-1}$ path.

The formal construction and inductive proof of existence of the Fibonacci path will have to be postponed. The important result to mention here is that the Fibonacci path grows faster in length than the Root2path. The sequence of Fibonacci path lengths, obtained by incrementing ℓ, is the Fibonacci sequence, {1,2,3,5,8,13,21,34,55,...}, since:

$$|F_\ell| = |F_{\ell-1}| + |F_{\ell-2}|$$

Solving the recurrence relation reveals exponential growth of $\approx 1.61803^\ell$, which has the golden ratio as its base. This base is larger than the base ≈ 1.414 in the exponential growth of the Root2path, but is still < 2. Thus, even the Fibonacci paths will asymptotically approach zero percent of the search space.

The problem of finding a maximally long path with minimal separation has some history, and is known as the "snake-in-the-box" problem, or the design of difference-preserving codes, in the literature on coding theory and combinatorics [7, 11]. Maximizing the length of paths with k-bits of separation is an open problem, even for $k = 1$. However, upper bounds have been found that are $< O(2^\ell)$. Thus the longest paths we can ever find[12] will be $O(2^{\ell/c})$ for some constant $c > 1$. For the Root2path, $c = 2$ for $k = 1$. For the Fibonacci path, $c = 1/(Log_2\Phi) \approx 1.44042$ when $k = 1$, where Φ is the golden ratio.

5.2 Extension: Analysis of Expected Performance

We need both empirical and analytical results for expected performance of various mutation algorithms and recombinative GAs. We wish to explore such apparent tradeoffs as step size k versus length of the path $|P|$. As k increases, $|P|$ decreases exponentially in k, but the number of fitness evaluations required to effectively search a neighborhood of radius k increases exponentially in k. A similar tradeoff involves the mutation rate p_m. As p_m increases, the mutation-only algorithm is more likely to take a larger shortcut across the path, but it is also less likely to find the next step on the path. These kinds of tradeoffs, involving parameters of the search space design and the algorithms themselves, are amenable to analysis of expectation.

5.3 Conclusions

Long path problems are clearly and demonstrably difficult for local searchers (that is, algorithms that search small neighborhoods with high probability, and

[12] It is easy to show that maximum path lengths must be $< 2^\ell$, at least for $k \geq 3$: divide the total volume of the search space by a volume $v(k)$ that is a lower bound on the number of off-path points that must "surround" each point on the path. This upper bound indicates that for $k \geq 3$, the optimal path length must approach zero exponentially fast as ℓ increases. This means that for $k \geq 3$, the best growth in path length for which we can hope is x^ℓ, where $x < 2$.

larger neighborhoods with vanishingly small probability). Such algorithms include hillclimbers and the GA's mutation operator. Surprisingly, some of these "intractable hills" can be solved efficiently by GA crossover. The fact that the GA solves problems specifically contrived for hillclimbers lends support to our intuition of GA robustness. Also able to solve long path problems of step size k are hillclimbers of step size $k' > k$. But long path problems of step size k' can be constructed to defeat such hillclimbers. The GA (with crossover) on the other hand *might* scale smoothly with increasing k. Such a result has implications for hybrid algorithms that perform hillclimbing during or after regular GA search: the addition of hillclimbing to the GA could make an "easy" problem intractable. Thus the long path problems reveal to us another dimension of problem difficulty for evolutionary computation; a dimension along which we can characterize, measure, and predict the performance of our algorithms.

References

1. Forrest, S., Mitchell, M.: Relative building-block fitness and the building-block hypothesis. In: L.D. Whitley (ed.): Foundations of Genetic Algorithms, 2. San Mateo, CA: Morgan Kaufmann (1993) 109–126
2. Goldberg, D. E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Reading, MA: Addison-Wesley (1989)
3. Goldberg, D. E.: Making genetic algorithms fly: a lesson from the Wright brothers. Advanced Technology for Developers. **2** February (1993) 1–8
4. Jones, T., Rawlins, G. J. E.: Reverse hillclimbing, genetic algorithms and the busy beaver problem. In: S. Forrest (ed.): Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann (1993) 70–75
5. Holland, J. H.: Adaptation in natural and artificial systems. Ann Arbor, MI: University of Michigan Press (1975)
6. Hoffmeister, F., Bäck, T.: Genetic algorithms and evolutionary strategies: similarities and differences. Technical Report "Grüne Reihe" No. 365. Department of Computer Science, University of Dortmund. November (1990)
7. MacWilliams, F. J., Sloane, N. J. A.: The Theory of Error Correcting Codes. Amsterdam, New York: North-Holland (1977)
8. Mitchell, M., Holland, J. H.: When will a genetic algorithm outperform hill climbing? In: S. Forrest (ed.): Proceedings of the Fifth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann (1993) 647
9. Mitchell, M., Holland, J. H., Forrest, S.: When will a genetic algorithm outperform hill climbing? Advances in Neural Information Processing Systems 6. San Mateo, CA: Morgan Kaufmann (to appear)
10. Mühlenbein, H.: How genetic algorithms really work, I. fundamentals. In: R. Männer, B. Manderick (eds.): Parallel Problem Solving From Nature, 2. Amsterdam: North-Holland (1992) 15–26
11. Preparata, F. P., Niervergelt, J.: Difference-preserving codes. IEEE Transactions on Information Theory. **IT-20:5** (1974) 643–649
12. Wilson, S. W.: GA-easy does not imply steepest-ascent optimizable. In: R.K. Belew, L.B. Booker (eds.): Proceedings of the Fourth International Conference on Genetic Algorithms. San Mateo, CA: Morgan Kaufmann (1991) 85–89