Technical Note:

How Mutation and Selection Solve Long Path Problems in Polynomial Expected Time

Günter Rudolph

rudolph@icd.de

Informatik Centrum Dortmund (ICD)

Center for Applied Systems Analysis (CASA)

Joseph–von–Fraunhofer–Str. 20

D-44227 Dortmund / Germany

Abstract

It is shown by means of Markov chain analysis that unimodal binary long path problems can be solved by mutation and elitist selection in a polynomially bounded number of trials on average.

1 Unimodality of Binary Functions

The notion of *unimodal* functions usually appears in the theory of optimization in $\mathbb{R}^1$. Elster et al. (1977), pp. 228–230, provide a precise definition that is specialized to functions in $\mathbb{R}^1$ whereas the definition in Bronstein and Semendjajew (1988), p. 137, for functions in $\mathbb{R}^\ell$ with $\ell \geq 1$ presupposes differentiability. Here, the following definition for functions over $\mathbb{B}^\ell$ will be used:

DEFINITION 1

Let f be a real–valued function with domain $\mathbb{B}^\ell$ where $\mathbb{B} = \{0, 1\}$. A point $x^* \in \mathbb{B}^\ell$ is called a *local solution* of f if

$$f(x^*) \leq f(x) \text{ for all } x \in \{y \in \mathbb{B}^\ell : \| y - x^* \|_1 = 1\} \tag{1}$$

where $\| x \|_1 = \sum_{i=1}^{\ell} | x_i |$ is the *Hamming norm*. If the inequality in (1) is strict, then x^* is termed a *strictly local solution*. The value $f(x^*)$ at a (strictly) local solution is called a *(strictly) local minimum* of f. A function $f : \mathbb{B}^\ell \to \mathbb{R}$ is said to be *unimodal*, if there exists exactly one local solution. □

The above definition is very close to that in Antamoshkin et al. (1990), p. 433 and Horn et al. (1994), p. 150. But there is an important difference: In their definition a local solution is always strict. Consequently, a function that is constantly 1 except for one point with function value 0 would be unimodal and the minimization problem would be NP–hard in general — a situation that is quite counter–intuitive to the notion of unimodal functions in $\mathbb{R}^\ell$. Definition 1 excludes such cases.

2 Unimodal Long Path Problems

By definition, for each point $x \in \mathbb{B}^\ell \setminus \{x^*\}$ of an unimodal problem there exists at least one path to the optimum with strictly decreasing function values, where consecutive points on the path differ in one bit only. If the expected number of trials of some probabilistic algorithm to invert a single specific bit is of order $O(\ell)$, an upper bound on the expected number of trials to reach the minimum is given by the length of the longest path times $O(\ell)$. Horn et al. (1994) succeeded in constructing paths that grow exponentially in ℓ and can be used to build unimodal problems. Consequently, the upper bound derived by the above reasoning either is too rough or indicates that polynomial bounds do not exist. It is clear that such a long path must possess much structure, because the 1–bit path has to be folded several times to fit into the "box" $\mathbb{B}^\ell$. One might expect that there exist many shortcuts by appropriate 2–bit steps, that decrease the order of the upper bound considerably.

Before checking this hypothesis it is necessary to know how to construct a long path. Horn et al. (1994), p. 152, proposed the following blueprint: Let P_ℓ be a long path in odd dimension ℓ. Create subpath S_{00} by prepending "`00`" to each point in path P_ℓ and subpath S_{11} by prepending "`11`" to each point in the reverse of path P_ℓ. The bridge point is built from the last point in path P_ℓ prepended by "`01`". Finally, concatenate subpath S_{00}, the bridge point and subpath S_{11} to obtain the long path $P_{\ell+2}$ of dimension $\ell+2$. Using the initial long path $P_1 = (\mathtt{0}, \mathtt{1})$, the long path of dimension 3 is $P_3 = (\mathtt{000}, \mathtt{001}, \mathtt{011}, \mathtt{111}, \mathtt{110})$. The length of the paths is described by the recurrence

equations

$$|P_1| = 2$$

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

whose solution is $|P_\ell| = 3 \cdot 2^{(\ell-1)/2} - 1 = O(2^\ell)$ for odd $\ell \geq 1$. Thus, the length of the path grows exponentially in ℓ. Figure 1 shows a long path in dimension $\ell = 9$.

Pos(x)	x	Pos(x)	x
0	000000000	46	110000000
1	000000001	45	110000001
2	000000011	44	110000011
3	000000111	43	110000111
4	000000110	42	110000110
5	000001110	41	110001110
6	000011110	40	110011110
7	000011111	39	110011111
8	000011011	38	110011011
9	000011001	37	110011001
10	000011000	36	110011000
11	000111000	35	110111000
12	001111000	34	111111000
13	001111001	33	111111001
14	001111011	32	111111011
15	001111111	31	111111111
16	001111110	30	111111110
17	001101110	29	111101110
18	001100110	28	111100110
19	001100111	27	111100111
20	001100011	26	111100011
21	001100001	25	111100001
22	001100000	24	111100000
23	011100000		

Fig. 1: A long 1–bit path for dimension $\ell = 9$.

Of course, one needs a fast algorithm that decides whether a point is on or off the path. Since each point on the path can be represented by the regular language

$$(00|11)^*((0\$|1\$)|01(1\$|10(00)^*0\$)),$$

where \$ is the end symbol of the string and the string is accepted from left to right, it is clear that the on/off–path decision can be done by a finite state machine in $O(\ell)$

time. Horn et al. (1994), p. 153, presented a recursive algorithm that either returns the position of the point in the path or indicates that the point is off the path after $O(\ell)$ steps. An iterative, table–driven version is given in Fig. 2: The program either returns the position on the path or indicates by a negative return value that the string is off the path. The end symbol \$ is replaced by a 2.

```
static int GetToken[2][3]  = {{0,1,4}, {2,3,5}};
static int GetState[3][6]  = {{0,1,3,0,4,4}, {3,3,3,2,3,4}, {2,3,3,3,4,3}};
static int GetAction[3][6] = {{0,5,1,4,2,3}, {1,1,1,0,1,2}, {0,1,1,1,2,1}};

int Pos(int Length, int *Str)
{
  register i, Sign, Pos, Token, State, Action;

  Sign = 1;
  Pos  = State = 0;
  for (i = Length - 1; i > 0; i -= 2) {
    Token  = GetToken[Str[i]][Str[i-1]];
    Action = GetAction[State][Token];
    switch (Action) {
      case 0: break;
      case 1: return(-1);
      case 2: return(Pos);
      case 3: return(Pos + Sign);
      case 4: Pos += (3 * (1 << ((i - 1) / 2)) - 2) * Sign;
              Sign = -Sign;
              break;
      case 5: Pos += (3 * (1 << ((i - 1) / 2 - 1)) - 1) * Sign;
              break;
    }
    State = GetState[State][Token];
  }
}
```

Fig. 2: An ANSI C program to determine the position on the path. Note that `Length` $= \ell + 1$ since `Str` is vector x with symbol `2` appended. A negative return value indicates that the point is off the path.

To construct the unimodal function let $\texttt{Pos}(x) \in \{-1, 0, 1, \ldots, 3 \cdot 2^{(\ell-1)/2} - 2\}$ return the position of string $x \in \mathbb{B}^\ell$ on the path for odd ℓ where $\texttt{Pos}(x) < 0$ indicates that x is off the path. Then

$$f(x) = 3 \cdot 2^{(\ell-1)/2} - 2 - \begin{cases} \texttt{Pos}(x) & \text{if } \texttt{Pos}(x) \geq 0 \\ -\|x\|_1 & \text{otherwise} \end{cases} \tag{2}$$

is an unimodal function: Each point $x \neq x^*$ *on* the path has exactly one better point within Hamming distance one, namely its successor on the path, whereas its predecessor and all remaining points within Hamming distance one do have worse function values. Each point $x \in \mathbb{B}^\ell$ *off* the path has at least one better point is its neighborhood: If an arbitrary 1 in x is inverted, then the resulting point is either off the path with better objective function value or on the path, where all function values are better than those off the path.

3 Optimization by Mutation and Elitist Selection: The (1+1)–EA

Consider the following algorithm, hereinafter called $(1 + 1)$–EA: Let $X^{(t)} \in \mathbb{B}^\ell$ at some iteration $t \geq 0$. The mutated point $Y^{(t)}$ is obtained by inverting each bit with mutation probability $p = 1/\ell$ independently. Thus, the probability to generate some point $y \in \mathbb{B}^\ell$ from $x \in \mathbb{B}^\ell$ by mutation is $\mathsf{P}\{ x \stackrel{M}{\to} y \} = p^k \, (1-p)^{\ell-k}$ where $k = \| x-y \|_1$. If $f(Y^{(t)}) < f(X^{(t)})$ then the mutated point is accepted (i.e., $X^{(t+1)} = Y^{(t)}$) otherwise it is rejected (i.e., $X^{(t+1)} = X^{(t)}$).

This method may be regarded as the discrete version of the $(1 + 1)$ evolution strategy that optimizes over continuous variables (Rechenberg 1973). There are many deterministic as well as probabilistic optimization algorithms that are related to the discrete $(1 + 1)$–EA.

1. The classical *steepest descent algorithm* calculates the objective function values of all points in the 1–bit neighborhood of the current point and moves to the point with the best objective function value. The algorithm halts if no better point was found in the neighborhood.

2. The *first improvement* or *next descent algorithm* (Horn et al. 1994) is a variation of the steepest descent algorithm: Again, the points in the 1–bit neighborhood are

successively tested in some order, but as soon as a better point than the current one is detected, the algorithm moves to this point.

3. The *random bit–climbing algorithm* used in Davis (1991) is a randomized version of the first improvement algorithm: All points in the 1–bit neighborhood to be tested are put into a list that is permuted at random. The point in this list that offers the first improvement is accepted.

4. The *random mutation hill–climbing algorithm* (Mitchell and Holland 1993) chooses a point from the 1–bit neighborhood at random and accepts this point if it is improving, otherwise it is rejected.

Since each of the above algorithms only searches in the 1–bit neighborhood of the current solution they require exponentially many function evaluations to reach the optimum of the long path problem if being started with the zero string. But as soon as a 2–bit neighborhood is used in lieu of the 1–bit neighborhood, these local search methods require considerably fewer function evaluations when searching for the minimum of the long path problem. The proof of this claim will not be given here — rather, it is shown in the next section that the $(1+1)$–EA requires $O(\ell^3)$ function evaluation on average to find the optimum of long path problems.

4 A Polynomial Bound for the Number of Objective Function Evaluations

To derive an upper bound on the expected number of trials the $(1+1)$–EA is approximated by a simplified Markov chain that has provable worse performance than the exact Markov chain. The basic idea is as follows: The search space $\mathbb{B}^\ell$ can be decomposed into a partition of disjoint subsets $S_0, \ldots, S_k$ such that the inequality $f(x) > f(y)$ is valid for every $x \in S_i$ and $y \in S_j$ with $0 \leq i < j \leq k$. Since worse points are not accepted

it is impossible to move from S_j to S_i with $i < j$. Notice that a fictitious $(1+1)$–EA that does not accept jumps over better sets must have worse performance than the original $(1+1)$–EA on average. In the worst case the $(1+1)$–EA must successively move through all sets S_0 to S_k (in this order). Consequently, if the expected number of mutations that are necessary to transition from S_i to S_{i+1} for each $i = 0, \ldots, k-1$ are known then the sum of these numbers is just an upper bound on the expected number of mutations that are needed to reach the optimum. The proof of the theorem below offers a more detailed description.

THEOREM 1

The $(1+1)$–EA minimizes function (2) in $O(\ell^3)$ expected trials with mutation probability $p = 1/\ell$ regardless of the initial point.

PROOF:

Suppose that the initial point is not on the path. Then at most $O(\ell \log \ell)$ trials are necessary to reach position 0 on the path (i.e., the zero vector) ignoring potential shortcuts to the path (Mühlenbein 1992).

Next assume that the current point is on the path. Note that any accepted point is necessarily on the path from now on. The set $\mathcal{O}_\ell$ of points on the path can be decomposed in the following manner:

$$
\begin{array}{lcl}
S_0 & = & \{(\texttt{00**} \quad \cdots \quad \texttt{*}) \in \mathcal{O}_\ell\} \cup \{\texttt{bridge point}\} \\
S_1 & = & \{(\texttt{1111**} \quad \cdots \quad \texttt{*}) \in \mathcal{O}_\ell\} \cup \{\texttt{bridge point}\} \\
S_2 & = & \{(\texttt{110011**} \quad \cdots \quad \texttt{*}) \in \mathcal{O}_\ell\} \cup \{\texttt{bridge point}\} \\
S_3 & = & \{(\texttt{11000011**} \quad \cdots \quad \texttt{*}) \in \mathcal{O}_\ell\} \cup \{\texttt{bridge point}\} \\
 & \vdots & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \vdots \\
S_{(\ell-3)/2} & = & \{(\texttt{1100} \quad \cdots \quad \texttt{0011*}) \in \mathcal{O}_\ell\} \cup \{\texttt{bridge point}\} \\
S_{(\ell-1)/2} & = & \{(\texttt{1100} \quad \cdots \quad \texttt{0000*}) \in \mathcal{O}_\ell\}
\end{array}
$$

These disjoint sets define a partial ordering of the points with respect to their position on the path: If $x \in S_i$ and $y \in S_j$ with $0 \leq i < j \leq (\ell-1)/2$ then $\texttt{Pos(x)} < \texttt{Pos(y)}$ and therefore $f(x) > f(y)$. Under the assumption that only 1–bit improvements are possible the Markov chain must follow the long 1–bit path that passes through all sets S_i in ascending order so that the path length is $|S_0| + \cdots + |S_{(\ell-1)/2}| = 3 \cdot 2^{(\ell-1)/2} - 1$ provided that the zero vector was the starting point. But as it is evident from the decomposition of $\mathcal{O}_\ell$, for any point $x \in S_i$ (except the bridge point) there exists a 2–bit shortcut to a set S_j with $0 \leq i < j \leq (\ell-1)/2$. For example, if $x \in S_0$ and the two leftmost bits are inverted simultaneously, then the resulting point is on the path and therefore in some set S_j with $j > 0$. The worst case is of course a shortcut to the set S_1. A similar argumentation applies to the other sets. If $x \in S_i$ is the bridge point then the set S_{i+1} is entered by an 1–bit improvement.

These observations lead to the following simplified Markov chain: Only 1–bit and 2–bit mutations will be considered. It will be assumed that 1–bit improvements are solely caused by 1–bit mutations, whereas 2–bit mutations can only cause the shortcut. Moreover, a shortcut from set S_i always leads to the set S_{i+1}. Under this setting the simplified Markov chain will have worse performance than the original one.

To calculate the absorption time of the simplified Markov chain, it is sufficient to determine the expected time to transition from set S_i to S_{i+1} for $i = 0, \ldots, (\ell-3)/2$. For this purpose the simplified Markov chain is split into $(\ell-1)/2$ segments, each segment representing an absorbing Markov chain. Evidently, the sum of the absorption times of these Markov chains is just the absorption time of the simplified Markov chain and thereby an upper bound of the original Markov chain.

Thus, a Markov chain of a segment can take three different actions at every iteration prior to absorption:

1. It can follow the path via an 1–bit improvement caused by an 1–bit mutation with probability a.

2. It can take a shortcut to the next subset via an appropriate 2–bit mutation with probability b.

3. It remains at the current position with probability $r = 1 - (a + b)$.

An example of the transition table of Markov chain $\mathcal{C}_1$ with path length $d = 4$ is given below:

$\mathcal{C}_1$	0	1	2	3	4
0	r	a	0	0	b
1	0	r	a	0	b
2	0	0	r	a	b
3	0	0	0	$1-a$	a
4	0	0	0	0	1

The Markov chain $\mathcal{C}_1$ has reached the next subpath if it is in state 4. As soon as it is in state 3 a shortcut cannot occur any longer. Thus, the transition to state 4 must be realized by a 1–bit mutation. The expected time for this event is of course $1/a$. This suggests a further simplification of the Markov chain $\mathcal{C}_1$: State 3 may be considered as an additional absorbing state, because at most $1/a$ iterations are necessary on average to transition from state 3 to state 4 of the Markov chain $\mathcal{C}_1$. Thus, if $\mathsf{E}[\,T_2\,]$ is the absorption time of the further simplified Markov chain $\mathcal{C}_2$ then the absorption time of $\mathcal{C}_1$ can be bounded by $\mathsf{E}[\,T_1\,] \leq \mathsf{E}[\,T_2\,] + 1/a$. Therefore, the transition matrix P of Markov chain $\mathcal{C}_2$ with path length d is given by

$$P = \begin{pmatrix} r & a & 0 & \cdots & 0 & 0 & b \\ 0 & r & a & & & & b \\ 0 & 0 & r & \ddots & & & b \\ \vdots & & & \ddots & \ddots & & \vdots \\ 0 & & & & r & a & b \\ 0 & & & & 0 & 1 & 0 \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{pmatrix}$$

of size $(d+1) \times (d+1)$. To determine the the absorption time delete the last two rows and the two rightmost columns of matrix P which yields submatrix Q. Then set $A = I - Q$ where I is the unit matrix. Evidently, matrix

$$A = \begin{pmatrix} 1-r & -a & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1-r & -a & & & & 0 \\ 0 & 0 & 1-r & \ddots & & & 0 \\ \vdots & & & \ddots & \ddots & & \vdots \\ 0 & & & & 1-r & -a & 0 \\ 0 & & & & 0 & 1-r & -a \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1-r \end{pmatrix}$$

is of size $(d-1) \times (d-1)$. Let matrix $B = A^{-1}$ be the inverse of A and let T_2 be the random time until absorption of Markov chain $\mathcal{C}_2$ when starting at state 0. As known from finite Markov chain theory (Iosifescu 1980, p. 104) the expectation of T_2 is given by the row sum of the top row of matrix $B = (b_{ij})$, i.e.,

$$\mathsf{E}[\, T_2 \,] = \sum_{k=0}^{d-2} b_{0k} \; . \tag{3}$$

To obtain these values note that the inverse of A can be computed via its adjugate:

$$B = A^{-1} = \frac{1}{\det A} \operatorname{adj} A \, .$$

Let A_{ij} be the submatrix of A where row i and column j has been deleted and set $A(i,j) = (-1)^{i+j} \det A_{ij}$. Then the adjugate of A is given by $\operatorname{adj} A = (A(i,j))'$. Consequently,

$$b_{0k} = A(k,0) \,/\, \det A = (-1)^k \cdot \det A_{k0} \,/\, \det A \tag{4}$$

for $k = 0, \ldots, d-2$. Evidently, the determinant of A is $\det A = (1-r)^{d-1}$. It remains to derive the determinants of submatrices A_{k0}. To this end delete the leftmost column and the kth row of A with $k = 0, \ldots, d-2$. It is easily seen that

$$\det A_{k0} = (-a)^k \, (1-r)^{d-2-k}. \tag{5}$$

Insertion of (4) and (5) into (3) leads to

$$\mathsf{E}[\, T_2 \,] = \sum_{k=0}^{d-2} b_{0k} = \frac{1}{1-r} \sum_{k=0}^{d-2} \left(\frac{a}{1-r} \right)^k = \frac{1}{b} \left[1 - \left(\frac{a}{a+b} \right)^{d-1} \right]$$

and finally, since $a = p \, (1-p)^{\ell-1}$ and $b = p^2 \, (1-p)^{\ell-2}$, to

$$\mathsf{E}[\, T_2 \,] = \frac{1-(1-p)^{d-1}}{p^2 (1-p)^{\ell-2}} \leq \ell^2 \;\; \exp(1) \left[1 - \left(1 - \frac{1}{\ell} \right)^{d-1} \right] \leq \ell^2 \;\; \exp(1) \tag{6}$$

with $p = 1/\ell$. Note that the rightmost bound in (6) is independent from the length d of the segment. Since $\mathsf{E}[\, T_1 \,] \leq \mathsf{E}[\, T_2 \,] + 1/a$ and since the simplified Markov chain was split into $(\ell - 1)/2$ segments, the absorption time of the original Markov chain can be bounded by

$$\mathsf{E}[\, T \,] \leq \frac{\ell - 1}{2} \, \mathsf{E}[\, T_1 \,] \leq \frac{\ell - 1}{2} \; \exp(1) \, (\ell^2 + \ell) + \exp(1) \, \ell \, \log \ell = O(\ell^3) \tag{7}$$

by insertion of (6) and adding the expected time to reach a point on the path. □

Table 1 summarizes the statistics of the first hitting time T obtained from 1,000 independent runs per dimension ℓ. When taking into account the constants in (7) one obtains the bound $\mathsf{E}[\, T \,] \leq \lceil 1.4 \, \ell^3 \rceil$ for $\ell \geq 15$ which is about three times as large as the sample mean. Since the (empirical) skewness and excess of T deviates considerably from zero in general, it cannot be expected that T is approximately normally distributed. Therefore the standard model of (weighted) regression analysis to estimate the true constants cannot be applied. Indeed, the relative frequencies of the *observed*

first hitting times given in fig. 3 clearly illustrate why the empirical standard deviation is so large and why the skewness as well as the excess is not close to zero.

ℓ	$\lceil 1.4\,\ell^3 \rceil$	mean	std. dev.	skewness	excess
15	4725	1139.295	756.86	0.5530	-0.1288
17	6879	1716.263	1174.36	0.5592	-0.1276
19	9603	2335.859	1635.294	0.7569	0.6716
21	12966	3519.185	2280.433	0.4617	-0.1246
23	17034	4675.412	3078.554	0.5437	-0.0779
25	21875	5977.070	4029.706	0.6533	0.4521
27	27557	7879.390	4910.918	0.4165	-0.1011
29	34145	9756.216	6176.630	0.3728	-0.2976
31	41708	12203.919	7329.924	0.5070	0.3285
33	50312	15323.342	9241.186	0.3388	-0.1998
35	60025	19166.404	10989.280	0.1929	-0.3539
37	70915	21545.718	12723.423	0.2538	-0.3052
39	83047	27991.581	15244.430	0.1436	-0.2718
41	96490	32228.906	17309.202	0.2491	-0.0786
43	111310	37623.133	21057.253	0.2954	0.0870
45	127575	42537.464	22450.626	0.1556	-0.1897
47	145353	50015.188	27072.361	0.1306	-0.2241
49	164709	56741.727	30508.643	0.1699	-0.1470

Table 1: Empirical mean, standard deviation, skewness and excess of the first hitting time T based on 1,000 independent runs per dimension ℓ.

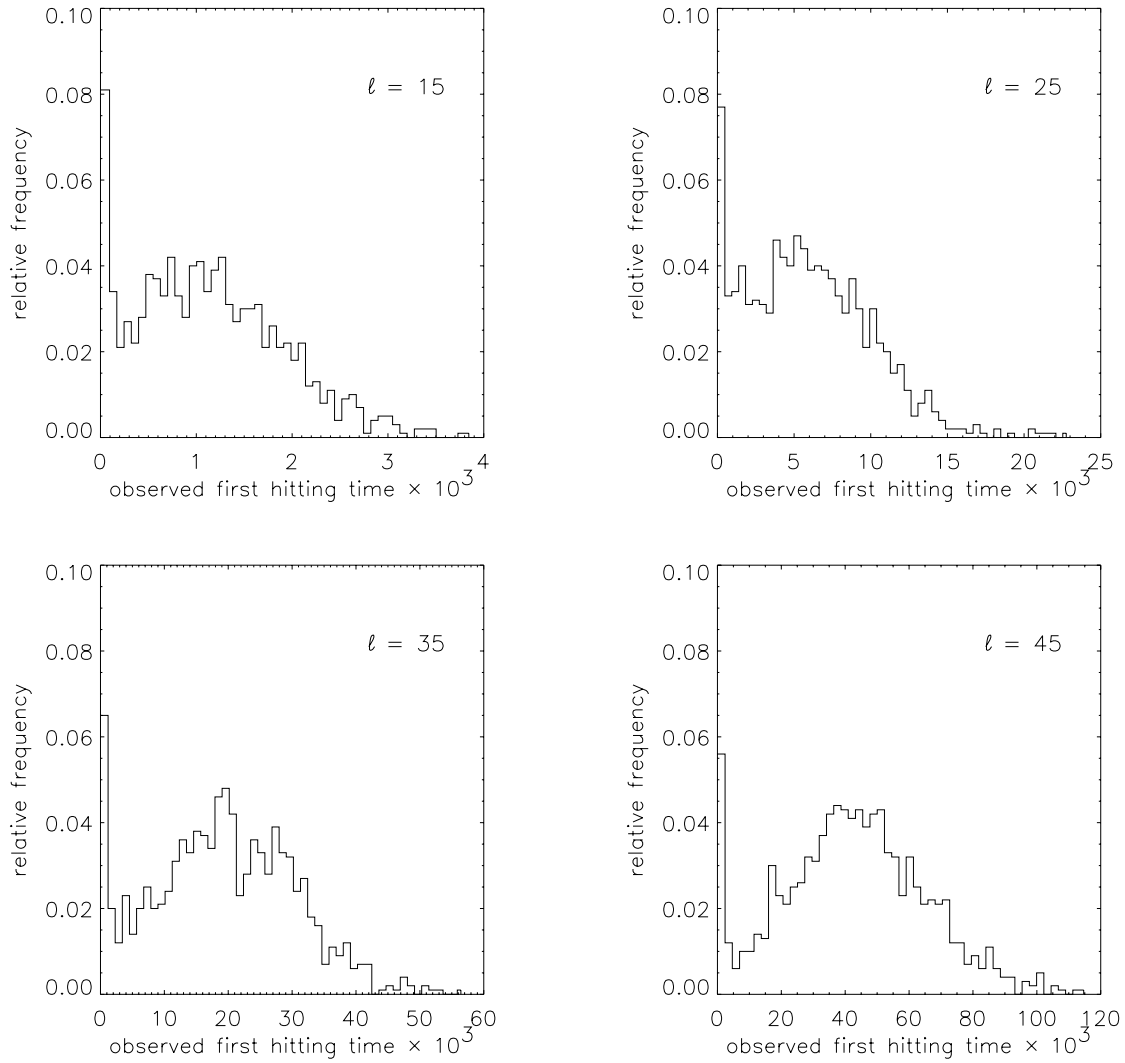

Figure 3: Relative frequencies of the observed first hitting time for dimensions $\ell = 15, 25, 35, 45$. Note the number of "outliers" close to zero.

5 Shorter but more difficult path problems

The long path problem is not a real challenge for a (1+1)–EA with mutation probability $p = 1/\ell$. Horn et al. (1994), p. 156–157, discussed some ideas how to construct longer paths. But since it is likely that longer paths have more structure that can eventually be exploited by an EA to take shortcuts, an unimodal problem does not necessarily become

more difficult this way. In contrast, shorter paths (but still of exponential length) might be more difficult, because the structure/regularity in the path can be decreased. Another route will be considered here: The structure of the path will essentially remain the same but to take a shortcut many bits must be altered simultaneously. Horn et al. (1994), p. 157, also mentioned this approach but they did not offer a method to construct those paths. The method given below is a straightforward generalization of the original method for usual long paths.

Let P_ℓ be a long path of dimension ℓ. Create subpath S_0 by prepending $k \geq 2$ zeroes to each point in path P_ℓ and subpath S_1 by prepending $k \geq 2$ ones to each point in the reverse of path P_ℓ. The bridge path consists of $k-1$ points, each of them built from the last point in P_ℓ prepended by substring $(\mathtt{0}\ldots\mathtt{01})$, $(\mathtt{0}\ldots\mathtt{011})$, $(\mathtt{001}\ldots\mathtt{1})$ and $(\mathtt{01}\ldots\mathtt{1}) \in \mathbb{B}^k$ respectively. Finally, concatenate substring S_0, the bridge path and substring S_1 to obtain a long path of dimension $\ell + k$. Long paths constructed in this manner will be called *long k–paths*. Note that long 2–paths are equivalent to long paths in Horn et al. (1994). The length of a long k–path is determined by the recursive equations

$$\begin{aligned} |P_1| &= 2 \\ |P_{\ell+k}| &= 2\,|P_\ell| + (k-1) \end{aligned}$$

whose solution is

$$|P_\ell| = (k+1)\,2^{(\ell-1)/k} - k + 1 \tag{8}$$

for $k \geq 2$ and where $(\ell-1)/k \in \mathbb{N}$.

Are these problems more difficult? Consider the (1+1)–EA as in Theorem 1 and assume that the current position is on the k–path, where the first k bits are zeros. If the first k bits are flipped while the others remain unaltered, then a successful k–bit shortcut has occurred. Since the probability for this event is $p^k\,(1-p)^{\ell-k}$, the expected time is less

than $e\,\ell^k$. In the worst case the bits $k+1$ to $2\,k$ are ones. Again, for a k–bit shortcut these k bits must be flipped simultaneously whereas the others remain unaltered. The expected time for this event is less than $e\,\ell^k$. Since there are $(\ell-1)/k$ such shortcuts, the expected time to reach the end of the path at x^* (with k ones and $\ell-k$ zeros from left to right) can be bounded by $O(\ell^{k+1}/k)$. Thus, it was proven:

THEOREM 2

Let $k \geq 2$ and $(\ell-1)/k \in \mathbb{N}$. The $(1+1)$–EA traverses a long k–path in dimension ℓ in $O(\ell^{k+1}/k)$ expected trials when using mutation probability $p = 1/\ell$. □

Note that $O(\ell^{k+1}/k)$ is an *upper bound* on the expected number of trials. For $k = \ell-1$ one obtains $O(\ell^{\ell-1})$ which is an exponential bound. But insertion of $k = \ell-1$ in (8) reveals that the path length reduces to $\ell+2$ which can be traversed by appropriate 1–bit improvements in $O(\ell^2)$ time. In general, choosing k proportional to ℓ yields a path length of $O(\ell)$ so that the bound on the expected number of trials is $O(\ell^2)$.
The choice of $k = (\ell-1)^{1/2}$ is more interesting: The path length reduces to $O(\sqrt{\ell}\,2^{\sqrt{\ell}})$ so that the bound on appropriate 1–bit improvements is not polynomial. Moreover, the argumentation in Theorem 2 leads to the bound of $O(\ell^{\sqrt{\ell}})$ trials. Both bounds are not polynomial. But since they are upper bounds it may be that a more detailed analysis yields lower (polynomial) bounds. It is, however, not obvious how the structure in the long $O(\sqrt{\ell})$–path can be exploited in another manner to achieve such bounds. Varying mutation rates like $p^{(t)} = (1 + t \bmod \ell)/\ell$ may be a solution. But this requires a more detailed investigation. Thus, the question, whether long $O(\sqrt{\ell})$–path problems can be solved in polynomial expected time by a $(1+1)$–EA must be left unanswered in this note.

6 Final Remark

It was claimed in the title of this note that mutation and selection can *solve* long path problems in polynomially bounded expected time. As a consequence, the $(1+1)$–EA must output the correct answer whenever it *terminates*. Since local and hence global optimality can be checked by considering the objective function values of the points in the 1–bit neighborhood of the current solution, the expected runtime of the EA increases to $O(\ell^4)$. But note that it is sufficient to check local optimality after every ℓth trial: This increases that number of trials by a factor of two, so that the bound $O(\ell^3)$ remains valid.

References

Antamoshkin, A.N., V.N. Saraev, and E.S. Semenkin (1990). Optimization of unimodal monotone pseudoboolean functions. *Kybernetika 26*(5), 432–441.

Bronstein, I.N. and K.A. Semendjajew (1988). *Taschenbuch der Mathematik; Ergänzende Kapitel* (5th ed.). Thun: Verlag Harri Deutsch.

Davis, L. (1991). Bit–climbing, representational bias, and test suite design. In R. Belew and L. Booker (Eds.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 18–23. San Mateo, CA: Morgan Kaufmann.

Elster, K.-H., R. Reinhardt, M. Schäuble, and G. Donath (1977). *Einführung in die nichtlineare Optimierung.* Leipzig: Teubner.

Horn, J., D.E. Goldberg, and K. Deb (1994). Long path problems. In Y. Davidor, H.-P. Schwefel, and R. Männer (Eds.), *Parallel Problem Solving from Nature, 3*, pp. 149–158. Berlin and Heidelberg: Springer.

Iosifescu, M. (1980). *Finite Markov Processes and Their Applications.* Chichester: Wiley.

Mitchell, M. and J.H. Holland (1993). When will a genetic algorithm outperform hill climbing? In S. Forrest (Ed.), *Proceedings of the Fifth International Conference on Genetic Algorithms*, p. 647. San Mateo, CA: Morgan Kaufmann.

Mühlenbein, H. (1992). How genetic algorithms really work I: Mutation and hill-climbing. In R. Männer and B. Manderick (Eds.), *Parallel Problem Solving from Nature, 2*, pp. 15–25. Amsterdam: North Holland.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Frommann–Holzboog Verlag.