UNIVERSITY OF DORTMUND

REIHE COMPUTATIONAL INTELLIGENCE

COLLABORATIVE RESEARCH CENTER 531

Design and Management of Complex Technical Processes and Systems by means of Computational Intelligence Methods

On the Analysis of the $(1+1)$ Evolutionary Algorithm

Stefan Droste, Thomas Jansen, and Ingo Wegener

No. CI-21/98

Technical Report ISSN 1433-3325 February 1998

Secretary of the SFB 531 · University of Dortmund · Dept. of Computer Science/XI
44221 Dortmund · Germany

This work is a product of the Collaborative Research Center 531, "Computational Intelligence", at the University of Dortmund and was printed with financial support of the Deutsche Forschungsgemeinschaft.

On the Analysis of the $(1+1)$ Evolutionary Algorithm*

Stefan Droste Thomas Jansen Ingo Wegener
FB Informatik, LS 2, Univ. Dortmund, 44221 Dortmund, Germany
droste, jansen, wegener@ls2.cs.uni-dortmund.de

Abstract

Many experimental results are reported on all types of Evolutionary Algorithms but only few results have been proved. A step towards a theory on Evolutionary Algorithms, in particular, the so-called $(1+1)$ Evolutionary Algorithm is performed. Linear functions are proved to be optimized in expected time $O(n \ln n)$ but only mutation rates of size $\Theta(1/n)$ can ensure this behaviour. For some polynomial of degree 2 the optimization needs exponential time. The same is proved for a unimodal function. Both results were not expected by several other authors. Finally, a hierarchy result is proved. Moreover, methods are presented to analyze the behaviour of the $(1+1)$ Evolutionary Algorithm.

1 Introduction

Evolutionary Algorithms are a class of search algorithms that are often used as function optimizers for static objective functions. There are a lot of different types of Evolutionary Algorithms, the best known are Evolution Strategies (Rechenberg (1994), Schwefel (1995)), Evolutionary Programming (Fogel (1995)), Genetic Algorithms (Holland (1975), Goldberg (1989)), and Genetic Programming (Koza (1992)). Each type of Evolutionary Algorithm itself is an algorithm paradigm that has many different concrete instances. The field of Evolutionary Algorithms, though the first origins can be found in the early 1960s, is still a quite young and evolving area of mostly practical efforts. The very successful application of Evolutionary Algorithms in very different domains led to strongly practical oriented interests. Today, theory is far behind "experimental knowledge". There are, of course, theoretical investigations about some properties of Evolutionary Algorithms, though rigorous research is hard to find. This implies that even the best known results are still subject to controversial discussions. To put an end to this, a solid theory has to be build up that starts with simple examples and shows how results can be obtained in a rigorous fashion. The most important questions concerning Evolutionary Algorithms, as long as function optimization is the objective, are how efficient an Evolutionary Algorithm will optimize a given objective function, which classes of objective functions can be optimized efficiently and which cannot.

In order to make a step towards this goal we investigate the running time behavior of a very simple variant, that is called $(1+1)$ Evolutionary Algorithm ($(1+1)$ EA). As the

*This work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (531).

name "Evolutionary Algorithm" suggests, evolution as it is observed in nature is imitated. It is believed that the repeated process of recombination, mutation, and selection leads to individuals that are increasingly adapted to their environment, i. e., they are assumed to be of increasing fitness. Therefore, in EAs a possible solution to the optimization task is called an *individual* and a set of individuals is called a *population*. From this population in one step or *generation* a subset of *parents* is *selected* according to their function values under the objective function, i. e., their *fitness*. The individuals are *recombined* by application of *crossover* and the resulting individuals are *mutated*. Then some *replacement strategy* is applied to determine the next population that may contain some of the newly generated *children* as well as some of their *parents*. The choice of concrete algorithms to perform selection, crossover, mutation, and replacement as well as the choice of the concrete representation of the individuals offers a great variety of different EAs.

The analysis here concentrates on the most simple variant of an EA that is still of theoretical and practical interest (Juels and Wattenberg (1994), Rudolph (1997)). We restrict the size of the population to just one individual and do not use crossover. Since we assume the objective function to have Boolean inputs we represent the current individual as a bit string. This is the usual choice for Genetic Algorithms. We use a bitwise mutation operator that flips each bit independently of the others with some probability p_m that depends on the length of the bit string. We replace the current bit string by the new one if the fitness of the current bit string is not superior to the fitness of the new string. This replacement strategy is taken from Evolution Strategies and is called $(1+1)$-strategy there: the new generation is chosen as the best individual of one parent and one child. We always assume that the objective or fitness function $f : \{0,1\}^n \to \mathbb{R}$ has to be maximized here. Therefore, we can formalize the $(1+1)$ EA as follows.

Algorithm 1.1:
1. Set $p_m := 1/n$.
2. Choose randomly an initial bit string $x \in \{0,1\}^n$.
3. Repeat the following mutation step: Compute x' by flipping independently each bit x_i with probability p_m. Replace x by x' iff $f(x') \geq f(x)$.

Algorithm 1.1 is sometimes regarded as a special variant of a hillclimber and is denoted as randomized or stochastic hillclimber then (compare e. g. Ackley (1987)). Like a hillclimber it uses only one current point in the search space and never accepts a new point with inferior function value. Unlike normal hillclimbers the $(1+1)$ EA has no clearly defined neighborhood, or, stated otherwise, it can reach in one single step any point in the search space, while the probability of reaching a point decreases with increasing Hamming distance to the current point.

One may consider the $(1+1)$ EA as a degenerated kind of Simulated Annealing (Kirkpatrick, Gelatt, and Vecchi (1983)), where the cooling scheme is trivial since the temperature is constantly zero. In this case, again, the probabilistic kind of neighborhood is quite unusual.

We start off here with giving some basic definitions that will be helpful during the analysis. More concepts and notions are introduced in the sections, when they are needed.

Definition 1.2: A function $f : \{0,1\}^n \to \mathbb{R}$ is called *fitness function*. We assume that f has to be maximized.

Definition 1.3: For two bit strings $x, y \in \{0,1\}^n$ we define the *Hamming distance* of x and y by

$$H(x,y) := \sum_{i=1}^{n} |x_i - y_i|.$$

Definition 1.4: A fitness function $f : \{0,1\}^n \to \mathbb{R}$ can be written as polynomial

$$f(x_1, \ldots, x_n) = \sum_{I \subseteq \{1,\ldots,n\}} c_f(I) \cdot \prod_{i \in I} x_i$$

with coefficients $c_f(I) \in \mathbb{R}$. The *degree* of f is defined as

$$\deg(f) := \max\{i \in \{0, \ldots, n\} \mid \exists I \text{ with } |I| = i \text{ and } c_f(I) \neq 0\}.$$

Definition 1.5: The *expected running time* $E(T)$ of the $(1+1)$ EA on a fitness function f is defined as the mean of the number of function evaluations of Algorithm 1.1 before the current bit string is a global optimum of f. The number of function evaluations is given by the number of times line 3 is executed increased by 1 for the initial bit string.

The *expected running time* of the $(1+1)$ EA on a class of fitness functions F is defined as the supremum of the expected running times on f for all $f \in F$.

The $(1+1)$ EA, under many different names as we mentioned above, has already been subject to various studies. As examples for theoretical investigations we mention three publications, though undoubtly a lot of other papers can be found. Bäck (1992) investigates the question which mutation rate should be chosen. Mühlenbein (1992) gives a sharp upper bound on the expected running time on a very simple linear fitness function. A number of different upper bounds on the expected running time, also for more interesting and challenging functions, can be found in Rudolph (1997).

In the following section we give a general upper bound on the expected running time of the $(1+1)$ EA for all fitness functions. We demonstrate that fitness functions exist, that require that amount of time asymptotically. We present a fitness function of degree 2 that has this worst case property. In Section 3 we prove that fitness functions of degree one, i. e., linear functions, are always solvable in expected time $O(n \log n)$. In Section 4 we deal with unimodal functions which are a superclass of the linear functions discussed in Section 3. We derive lower bounds on the expected running time for two concrete unimodal fitness functions and show that this class has exponential expected running time in the worst case. After dealing with extreme running times only, in Section 5 we give a definition of n functions where the expected running time for the m-th function is $\Theta(n \log n + n^m)$ for $1 \leq m \leq n$. Finally, in Section 6 we discuss a variant of Algorithm 1.1 and demonstrate that very little changes in the algorithm can cause huge differences in the expected running time.

2 Worst case bounds and worst case examples

As we are interested in the efficiency of the $(1+1)$ EA for different fitness functions, we should compare the expected running time of the $(1+1)$ EA with other optimization algorithms. The most simple algorithm for function optimization, complete enumeration, needs $O(2^n)$ steps to find the global optimum of an arbitrary fitness function over n Boolean variables. In Subsection 2.1 we show, that the $(1+1)$ EA finds the global optimum of every fitness function after at most n^n steps, which is worse than $O(2^n)$.

Because this upper bound does not imply the existence of fitness functions, so that the $(1+1)$ EA has an expected running time of $\Theta(n^n)$, we explicitly present two fitness functions with expected running time $\Theta(n^n)$ in Subsections 2.2 and 2.3. While the first fitness function is of degree n, the second one has degree 2, showing that the separation of the set of all fitness functions according to the degree of a function is not reasonable with respect to its indications on the running time of the $(1+1)$ EA.

2.1 A general upper bound for the expected running time

Theorem 2.1: The expected running time of the $(1+1)$ EA for an arbitrary fitness function is at most n^n.

Proof: Let $x \in \{0,1\}^n$ be an arbitrary bit string and x^* a global optimum of the fitness function. As $H(x,x^*) \leq n$, the probability of mutating from x to x^* in one step is $(1/n)^{H(x,x^*)} \cdot (1-1/n)^{n-H(x,x^*)} \geq n^{-n}$. Hence, the expected time until this event occurs is at most n^n. Because this event implies, that a global optimum of the fitness function is found, the expected running time of the $(1+1)$ EA is at most n^n. □

As in this proof the exact expected running time of the $(1+1)$ EA is only roughly upper bounded, it is not clear, if any fitness function exists, so that the $(1+1)$ EA needs on average $\Theta(n^n)$ steps for optimizing it. The next subsection presents an explicit function with this property.

2.2 A function of degree n with expected running time $\Theta(n^n)$

A fitness function is assumed to be difficult to be optimized, if it gives "misleading hints" regarding the position of its global optimum. The $(1+1)$ EA most often makes only small mutations, which are accepted if they do not decrease the fitness function value. Therefore, the expected running time of the $(1+1)$ EA should be high, if for most bit strings their "neighbors", i.e. bit strings with small Hamming distance, with higher fitness lead away from the global optimum.

The function $\text{TRAP}:\{0,1\}^n \to \mathbb{R}$ (see Ackley (1987)), defined by

$$\text{TRAP}(x_1,\ldots,x_n) := \sum_{i=1}^{n} x_i + (n+1)\cdot\prod_{i=1}^{n}(1-x_i),$$

fulfills these requirements: the global optimum is $(0, \ldots, 0)$, but for every other $x \in \{0,1\}^n$ all bit strings with more ones than x have higher fitness, leading away from the global optimum. Hence, the expected running time of the $(1+1)$ EA when starting in a bit string x should increase with $|x|$, the number of ones in x. The following theorem rigorously proves that these argumentations are correct.

Theorem 2.2: The expected running time of the $(1+1)$ EA for TRAP is $\Theta(n^n)$.

Proof: The basic idea of the proof is, that for almost all initial bit strings it is very likely that the $(1+1)$ EA will run into the local optimum $(1, \ldots, 1)$, because all mutations increasing the number of ones are accepted. But as the only possible successful mutation from $(1, \ldots, 1)$ leads to the global optimum $(0, \ldots, 0)$ and has probability n^{-n}, the expected waiting time for this mutation is n^n.

Let $E(T_k)$ for $k \in \{0, \ldots, n\}$ be the expected running time of the $(1+1)$ EA, when starting with a bit string with k ones, which is properly defined, because TRAP is a symmetric function, i.e., its function value depends only on the number of ones in its input. Because the initial bit string is chosen randomly, $E(T)$, the expected running time of the $(1+1)$ EA, is

$$E(T) = 2^{-n} \cdot \sum_{k=0}^{n} \binom{n}{k} \cdot E(T_k).$$

Because of Theorem 2.1, it is sufficient to show that the expected running time for TRAP is $\Omega(n^n)$ in order to show that it is $\Theta(n^n)$. In the following we show that $\mathrm{Prob}(Z_k \leadsto Z_n)$, the probability of reaching the bit string with n ones, when starting from an arbitrary bit string with $k \in \{2, \ldots, n\}$ ones and using arbitrarily many steps in between, is at least a constant $c > 0$. As the probability of mutating from $(1, \ldots, 1)$ to $(0, \ldots, 0)$ is n^{-n}, $E(T_n)$ is n^n, implying:

$$\begin{aligned}
E(T) &= 2^{-n} \cdot \sum_{k=0}^{n} \binom{n}{k} \cdot E(T_k) \\
&\geq 2^{-n} \cdot \sum_{k=2}^{n} \binom{n}{k} \cdot (\mathrm{Prob}(Z_k \leadsto Z_n) E(T_k | Z_k \leadsto Z_n) + \\
&\qquad\qquad\qquad\qquad \mathrm{Prob}(Z_k \not\leadsto Z_n) E(T_k | Z_k \not\leadsto Z_n)) \\
&\geq 2^{-n} \cdot \sum_{k=2}^{n} \binom{n}{k} \cdot c \cdot n^n = c \cdot n^n \cdot \frac{2^n - 1 - n}{2^n} = \Omega(n^n).
\end{aligned}$$

So we have to show, that $\mathrm{Prob}(Z_k \leadsto Z_n) \geq c$, i.e. $\mathrm{Prob}(Z_k \not\leadsto Z_n) \leq 1 - c$ for all $k \in \{2, \ldots, n\}$. $\mathrm{Prob}(Z_k \not\leadsto Z_n)$ is the probability of not reaching $(1, \ldots, 1)$, while starting from a bit string with k ones. This event is equivalent to reaching a bit string with $i \in \{k, \ldots, n-1\}$ ones, but no bit string with $i' > i$ ones. Let $\mathrm{Prob}(Z_i \not\leadsto Z_{i+})$ be the probability of not reaching any bit string with $i' > i$ ones, while starting in a bit string with i ones, in arbitrarily many steps. As the probability of reaching a bit string with i ones, while starting from a bit string with k ones is at most one, $\mathrm{Prob}(Z_k \not\leadsto Z_n)$ can be bounded above by $\sum_{i=k}^{n-1} \mathrm{Prob}(Z_i \not\leadsto Z_{i+})$.

Let p_i^- be the probability of mutating from a bit string with i ones in one step to $(0,\ldots,0)$ and p_i^+ the probability of mutating from a bit string with i ones in one step to a bit string with more than i ones. Then the following equation holds:

$$\text{Prob}(Z_i \not\rightsquigarrow Z_{i+}) = \sum_{t=0}^{\infty}\left(1-p_i^+-p_i^-\right)^t \cdot p_i^- = \frac{p_i^-}{p_i^-+p_i^+} = \left(1+\frac{p_i^+}{p_i^-}\right)^{-1}.$$

Using an upper bound $\tilde{p}_i^-$ for p_i^- and a lower bound $\tilde{p}_i^+$ for p_i^+ the probability $\text{Prob}(Z_i \not\rightsquigarrow Z_{i+})$ can be bounded above in the following way:

$$\text{Prob}(Z_i \not\rightsquigarrow Z_{i+}) = \left(1+\frac{p_i^+}{p_i^-}\right)^{-1} \leq \left(1+\frac{\tilde{p}_i^+}{\tilde{p}_i^-}\right)^{-1} = \frac{\tilde{p}_i^-}{\tilde{p}_i^-+\tilde{p}_i^+} \leq \frac{\tilde{p}_i^-}{\tilde{p}_i^+}.$$

The probability of mutating a bit string with i ones to $(0,\ldots,0)$ is exactly $(1/n)^i \cdot (1-1/n)^{n-i}$. Assuming that only the mutation of exactly 1 of the $n-i$ zeros leads to a bit string with $i+1$ ones, the lower bound $\binom{n-i}{1}\cdot 1/n\cdot(1-1/n)^{n-1}$ on p_i^+ follows. Putting together these bounds, we obtain the following upper bound for $\text{Prob}(Z_i \not\rightsquigarrow Z_{i+})$:

$$\frac{\left(\frac{1}{n}\right)^i \cdot \left(1-\frac{1}{n}\right)^{n-i}}{\binom{n-i}{1}\cdot\frac{1}{n}\cdot\left(1-\frac{1}{n}\right)^{n-1}}.$$

Hence, $\text{Prob}(Z_2 \not\rightsquigarrow Z_n)$ is bounded above by the following sum:

$$\sum_{i=2}^{n-1}\frac{\left(\frac{1}{n}\right)^i \cdot \left(1-\frac{1}{n}\right)^{n-i}}{\frac{n-i}{n}\cdot\left(1-\frac{1}{n}\right)^{n-1}} \leq \sum_{i=2}^{n-1}\left(\frac{1}{n}\right)^{i-1}\cdot\left(1-\frac{1}{n}\right)^{1-i} \quad (\text{as } (1-1/n)^{n-2} \geq \exp(-1))$$

$$\leq \exp(1)\cdot\sum_{i=1}^{\infty}\left(\frac{1}{n}\right)^i \leq \exp(1)\cdot\frac{1}{n}\cdot\frac{1}{1-1/n} = \exp(1)\cdot\frac{1}{n-1} \to 0 \text{ (for } n\to\infty).$$

Therefore, $\text{Prob}(Z_2 \rightsquigarrow Z_n)$ and, in general, $\text{Prob}(Z_k \rightsquigarrow Z_n)$ for $k \in \{2,\ldots,n\}$ is at least a constant greater 0. So the expected number of steps of the $(1+1)$ EA for TRAP is $\Theta(n^n)$. □

2.3 A function of degree two with expected running time $\Theta(n^n)$

So far we have shown in this section that the $(1+1)$ EA needs running time at most n^n and that a function of degree n with expected running time $\Theta(n^n)$ exists. As the degree of a function is a measure for the degree of dependency between the variables, one might assume that the degree of a fitness function is an indicator for the expected running time of the $(1+1)$ EA while optimizing it. But in the following we show that this assumption is not correct in general.

To do this, we explicitly present a function of degree 2 with expected running time $\Theta(n^n)$, showing that functions of degree 2 are in fact hard to optimize for the $(1+1)$ EA.

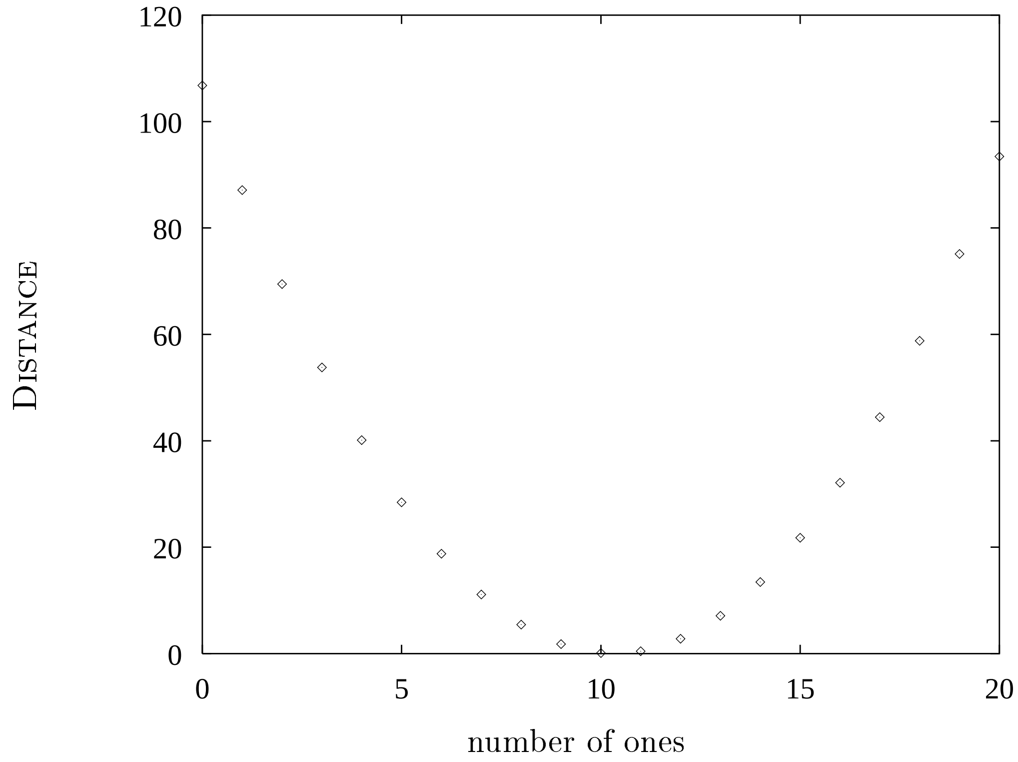

Figure 1: The function DISTANCE for $n = 20$

Theorem 2.3: The $(1+1)$ EA has an expected running time of $\Theta(n^n)$ for the function DISTANCE, defined by

$$\text{DISTANCE}(x_1, \ldots, x_n) = \left(\sum_{i=1}^{n} x_i - \left(\frac{n}{2} + \frac{1}{3}\right)\right)^2.$$

Proof: This function is of degree 2 and to get a graphical impression, it is plotted in Figure 1 for $n = 20$. The basic idea of the proof is the same as for Theorem 2.2: for a constant-sized part of all initial bit strings the probability of reaching the local optimum $(1, \ldots, 1)$ is constant. As the only bit string with higher fitness is the global optimum $(0, \ldots, 0)$, the expected runtime of the $(1+1)$ EA, when starting in $(1, \ldots, 1)$ is n^n. Hence, the expected running time of the $(1+1)$ EA is $\Theta(n^n)$.

As the value of DISTANCE is only dependent on the number of ones in its input, let DISTANCE(k) be its function value for an arbitrary input with $k \in \{0, \ldots, n\}$ ones. As its function value is the square of the distance between the number of ones in the input and $n/2 + 1/3$, it can be rewritten as (where $i \geq 0$):

$$\text{DISTANCE}(k) = \begin{cases} (i - \frac{1}{3})^2, & \text{if } k = \frac{n}{2} + i \\ (i + \frac{1}{3})^2, & \text{if } k = \frac{n}{2} - i \end{cases}.$$

Hence, the order of the function values is as follows (w.l.o.g. n is even):

$$\textsc{Distance}(\frac{n}{2}) < \textsc{Distance}(\frac{n}{2}+1) < \textsc{Distance}(\frac{n}{2}-1) < \textsc{Distance}(\frac{n}{2}+2) < \dots$$

$$\textsc{Distance}(n-1) < \textsc{Distance}(n) < \textsc{Distance}(0),$$

i.e., starting from a bit string with $n/2+i$ ones every bit string with at most $n/2-i$ and every bit string with at least $n/2+i$ can be reached. This implies, that starting from $(1,\dots,1)$ only $(0,\dots,0)$ (besides $(1,\dots,1)$ itself) can be reached. As $(0,\dots,0)$ is the global optimum, the expected running time of the $(1+1)$ EA, when starting in $(1,\dots,1)$, is n^n. Analogously to Section 2.2 we will now prove that the probability $\text{Prob}(Z_k \rightsquigarrow Z_n)$ of reaching $(1,\dots,1)$, while starting with a bit string with k ones and using arbitrarily many steps, is at least a constant $c > 0$ for all $k \geq n/2+1$.

What is the probability of randomly initializing with a bit string having at least $n/2+1$ ones? The number of bit strings with at least $n/2+1$ ones is $\sum_{k=n/2+1}^{n} \binom{n}{k} \geq 2^{n-1} - \binom{n}{n/2} = \Omega(2^n)$. Hence, the initialization leads to a bit string having at least $n/2+1$ ones with constant probability.

To get a lower bound for $\text{Prob}(Z_k \rightsquigarrow Z_n)$, let us consider a sufficient condition for the (1+1) EA to reach $(1,\dots,1)$: starting from a bit string with k ones a bit string with $i > k$ ones has to be reached, while only visiting bit strings with k ones in between. Then a bit string with $i' > i$ ones has to be reached, while only visiting bit strings with i ones in between, and so on, until $(1,\dots,1)$ has been reached. This neglects that mutating to a bit string with less ones than the current bit string does not in general imply that the (1+1) EA cannot reach $(1,\dots,1)$ anymore.

Let $\text{Prob}(Z_i \rightsquigarrow Z_{i+})$ be the probability of reaching a bit string with $i' > i$ ones when starting in a bit string with i ones and using only bit strings with i ones in between. Then the following inequality holds

$$\text{Prob}(Z_k \rightsquigarrow Z_n) \;\; \geq \;\; \prod_{i=k}^{n-1} \text{Prob}(Z_i \rightsquigarrow Z_{i+}).$$

Hence, it is sufficient to show that $\prod_{i=k}^{n-1} \text{Prob}(Z_i \rightsquigarrow Z_{i+})$ is at least a constant greater than 0. To get a lower bound for $\text{Prob}(Z_i \rightsquigarrow Z_{i+})$, the probabilities p_i^+ (to mutate from a bit string with i ones to a bit string with more ones) and p_i^- (to mutate successfully from a bit string with i ones to a bit string with less ones) are bounded: if at least one of the $n-i$ zeros mutates, but no bit with value one, then a bit string with more ones is reached (hence, $p_i^+ \geq (1-1/n)^i \cdot (1-(1-1/n)^{n-i})$); in order to switch to a bit string with less ones, it is necessary that $2(i-n/2) = 2i-n$ ones mutate (hence, $p_i^- \leq \binom{i}{2i-n} \cdot (1/n)^{2i-n}$). Analogously to Subsection 2.2 this leads to the following lower bound

$$\text{Prob}(Z_i \rightsquigarrow Z_{i+}) = \frac{p_i^+}{p_i^+ + p_i^-} \geq \frac{\left(1-\left(1-\frac{1}{n}\right)^{n-i}\right) \cdot \left(1-\frac{1}{n}\right)^{i}}{\left(1-\left(1-\frac{1}{n}\right)^{n-i}\right) \cdot \left(1-\frac{1}{n}\right)^{i} + \binom{i}{2i-n} \cdot \left(\frac{1}{n}\right)^{2i-n}} =: a_i^{-1}.$$

Hence, we get the lower bound $\left(\prod_{i=n/2+1}^{n-1} a_i\right)^{-1}$ for $\text{Prob}(Z_{n/2+1} \rightsquigarrow Z_n)$. This expression is at least a constant greater than 0, if $\prod_{i=n/2+1}^{n-1} a_i$ is bounded above by a constant, which is the case iff $\sum_{i=n/2+1}^{n-1} \ln(a_i)$ is bounded above by a constant. Since $\ln(1+x) \leq x$ and a_i has the form $1 + b_i$, where

$$b_i = \frac{\binom{i}{2i-n} \cdot \left(\frac{1}{n}\right)^{2i-n}}{\left(1 - \left(1 - \frac{1}{n}\right)^{n-i}\right) \cdot \left(1 - \frac{1}{n}\right)^{i}},$$

it is sufficient to show that $\sum_{i=n/2+1}^{n-1} b_i$ is bounded above by a constant.
If $i \leq 2n/3$ (w.l.o.g. n is divisible by 6), the denominator of b_i is at least $(1-\exp(-1/3)) \cdot \exp(-1)$, while for the numerator we use the estimation

$$\binom{i}{2i-n} \cdot \left(\frac{1}{n}\right)^{2i-n} \leq \left(\frac{i}{n}\right)^{2i-n} \leq \left(\frac{4}{9}\right)^{i-n/2}.$$

Hence, the sum $\sum_{i=n/2+1}^{2n/3} b_i$ is bounded above by

$$\frac{1}{(1-\exp(-1/3)) \cdot \exp(-1)} \cdot \sum_{i=1}^{n/6} \left(\frac{4}{9}\right)^{i} \leq \frac{4}{5 \cdot (1-\exp(-1/3)) \cdot \exp(-1)}.$$

If $i \in \{2n/3+1, \ldots, n-1\}$, then $(1-1/n)^{n-i} \leq 1 - 1/n$ and the denominator of b_i is at least $\exp(-1)/n$. For the nominator we use the estimation $\binom{i}{2i-n} = \binom{i}{n-i} \leq n^{n-i}$. Hence, the whole numerator is at most n^{2n-3i}. Since $i \geq 2n/3+1$, this can be estimated above by n^{-3}. Therefore, $\sum_{i=2n/3+1}^{n-1} b_i$ is bounded above by

$$\exp(1) \cdot \sum_{i=2n/3+1}^{n-1} \frac{1}{n^2} \leq \frac{\exp(1)}{n}.$$

Altogether, $\sum_{i=n/2+1}^{n-1} b_i$ is bounded above by

$$\frac{4}{5 \cdot (1-\exp(-1/3)) \cdot \exp(-1)} + \frac{\exp(1)}{n}.$$

Hence, the expected number of steps of the $(1+1)$ EA for the DISTANCE-function is $\Theta(n^n)$. □

3 Linear functions*

We know from the previous section that fitness functions of degree at least 2 can be very difficult for the $(1+1)$ EA. Fitness functions of degree 0 are, of course, trivial, since they

*The results of this section have been presented at the ICEC '98 (Droste, Jansen, and Wegener (1998)).

are constant. Fitness functions of degree 1 remain as an interesting class of functions, and they are in fact easy for the $(1+1)$ EA as we will prove here. We call these functions *linear*, since they can be written as

$$f(x) = \sum_{i=1}^{n} g_i(x_i),$$

where $g_i(x_i) = w_i x_i + \tau_i$ for i with $1 \leq i \leq n$ and $w_i, \tau_i \in \mathbb{R}$ holds.

For the analysis in this section we make w. l. o. g. some assumptions. Constant additive terms have no influence on the behavior of the $(1+1)$ EA, so we assume $\tau_i = 0$ for all i. We assume that all weights are non-negative. Otherwise one may replace x_i by $1 - x_i$. Furthermore, the weights w_i are assumed to be integers. Finally, we assume that the weights are sorted, i. e., $w_1 \geq \cdots \geq w_n$. It follows that the all one string is always a global optimum. Moreover, if all weights are positive, the all one string is the only global optimum.

Before we consider an arbitrary linear function f, we introduce two special linear functions that are of particular interest.

Definition 3.1: The linear function OneMax has all weights set to 1, i. e.,

$$\text{OneMax}(x) = \sum_{i=1}^{n} x_i.$$

The linear function Bin has set the i-th weight according to $w_i := 2^{n-i}$, i. e., Bin interprets a bit string as binary representation of an integer.

These two functions are in some sense two extreme examples from the class of linear functions. The weights of Bin are so strongly decreasing that $w_i > \sum_{j=i+1}^{n} w_j$ holds for all i with $1 \leq i < n$. This implies that it is always the leftmost flipping bit alone that decides, whether a mutation is accepted. The weights of OneMax are all equal, so that it is only the number of bits flipping to one compared with the number of bits flipping to zero that decides, whether a mutation is accepted.

The function OneMax is easy to analyze, upper bounds for the expected running time have been presented, e. g., by Mühlenbein (1992). As for all symmetric functions, mutation steps to bit strings with equal number of ones can be ignored. In successful steps the number of ones is increased by at least 1, so at most n successful steps are sufficient. If the number of zeros in the current bit string equals i, the probability for a successful step is bounded below by $\binom{i}{1} n^{-1} (1 - 1/n)^{n-1}$, so we get

$$\sum_{i=1}^{n} \left(\binom{i}{1} \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} \right)^{-1} \leq \exp(1) n \sum_{i=1}^{n} \frac{1}{i} = O(n \ln n)$$

as upper bound for the expected running time. A lower bound of equal order of growth is easy to find, too. Furthermore, it is valid for all linear functions with all non-zero weights.

Lemma 3.2: The expected number of steps the $(1+1)$ EA takes to optimize a linear function with all non-zero weights is $\Omega(n \ln n)$.

Proof: By our assumptions in this section all weights w_i are positive. Since the all one bit string is the only optimum, it is necessary that each bit that is zero after random initialization flips at least once. Hence, the average time until each of these bits has tried to flip at least once is a lower bound on the considered expected time. The following considerations are similar to the example called "coupons collector problem" by Motwani and Raghavan (1995).

Let T denote the random variable describing the first point of time where each of these bits has tried to flip at least once. Since T takes only positive integers, we have

$$E(T) = \sum_{t=1}^{\infty} t \cdot \text{Prob}(T = t) = \sum_{t=1}^{\infty} \text{Prob}(T \geq t).$$

W. l. o. g. n is even. With probability at least $1/2$ at least half of the bits are zero after random initialization. $(1-1/n)^{t-1}$ describes the probability that one bit does not flip at all in $t-1$ steps. So, $1-(1-1/n)^{t-1}$ is the probability that it flips at least once in $t-1$ steps. Therefore, we have $(1-(1-1/n)^{t-1})^{n/2}$ as probability that this is the case with $n/2$ bits. Finally, $1-(1-(1-1/n)^{t-1})^{n/2}$ is the probability for the event that at least one of $n/2$ bits never flips in $t-1$ steps. So, we have

$$\begin{aligned}
E(T) &\geq \frac{1}{2}\sum_{t=1}^{\infty}\left(1-\left(1-\left(1-\frac{1}{n}\right)^{t-1}\right)^{n/2}\right) \\
&\geq \frac{1}{2}(n-1)(\ln n)\left(1-\left(1-\left(1-\frac{1}{n}\right)^{(n-1)\ln n}\right)^{n/2}\right) \\
&\geq \frac{1}{2}(n-1)(\ln n)\left(1-\exp(-1/2)\right) = \Omega(n \ln n).
\end{aligned}$$

$\square$

For Bin an upper bound is harder to find. Since it is only the leftmost flipping bit that decides, whether a mutation is accepted, the Hamming distance to the optimal all one string may be increased during optimization. Even extreme steps like the one from $(0, 1, 1, \ldots, 1)$ to $(1, 0, 0, \ldots, 0)$ are possible though extremely unlikely. In order to measure the progress during optimization we distinguish between the bits in the left half (i. e., $x_1, \ldots, x_{n/2}$) and the bits in the right half of the current bit string. We consider the bits in the left half to be more important and analyze the optimization in two phases. In the first phase only the bits in the left half are considered. We derive an upper bound on the expected time before these bits all are ones. Such leading ones are never replaced by zeros, so in the second phase, when we consider the bits of the right half, we are sure that nothing changes in the left half. Adding up the expected time for the left and the right half yields an upper bound for the expected running time of the $(1+1)$ EA on Bin.

Lemma 3.3: The expected number of steps of the $(1+1)$ EA on BIN is $\Theta(n \ln n)$.

Proof: The lower bound is given in Lemma 3.2, so we only prove an upper bound.

W. l. o. g. n is even. Let T be the random variable describing the first point of time where the $(1+1)$ EA reaches the optimal all one bit string. As described above we distinguish two phases and denote the first point of time where the $n/2$ bits of the left half are all one by T_1. The rest of the time is given by T_2, so we have $T = T_1 + T_2$.

In order to derive an upper bound on $E(T_1)$ we distinguish the mutations: steps that change at least one of the bits in the left half are called successful. Unsuccessful steps may only change bits in the right half and are not important for us.

Let X_i be the random variable describing the first point of time where the left half of the current bit string x contains at least i ones, in particular $X_0 = 0$. Then we have

$$T_1 = X_{n/2} = (X_1 - X_0) + (X_2 - X_1) + \cdots + (X_{n/2} - X_{n/2-1}).$$

We distinguish between successful and unsuccessful mutations and therefore introduce the random variables Y_i that describe the random number of successful steps in the interval $[X_{i-1}+1, \ldots, X_i]$, as well as Z_i that describe the random number of unsuccessful steps during that period of time. This yields

$$T_1 = Y_1 + Z_1 + Y_2 + Z_2 + \cdots + Y_{n/2} + Z_{n/2},$$

and we can investigate successful and unsuccessful steps separately.

We begin with Y_i. In a successful step the leftmost flipping bit flips from zero to one. In order to get an upper bound on Y_i we pessimistically assume that all other flipping bits flip from one to zero. The expected number of further flipping bits is maximal, if the leftmost flipping bit is the leftmost bit in the bit string. If we look at the bits on positions $2, \ldots, n/2$, we see that they may all flip independently of each other and the first bit. So, the distribution of the random number of flipping bits among these positions is a binomial distribution with parameters $n/2 - 1$ and $1/n$. We conclude that regardless of the actual position of the leftmost flipping bit the expected number of further flipping bits is bounded above by $1/2$. Let D_k describe the random difference in the number of ones among the bits of the left half of x after and before the k-th successful step among the steps $X_{i-1}+1, \ldots, X_i$. Then under our pessimistic assumptions, $D_k \leq 1$ and $E(D_k) \geq 1 - 1/2 = 1/2$. Let $S_k = D_1 + D_2 + \cdots + D_k$. Then Y_i is bounded above by k^*, the smallest index k where $S_k = 1$. The process $S_0, S_1, S_2, \ldots$ is a random walk on the line starting at $S_0 = 0$. In the first step we move to $S_0 + D_1 = S_1$, then to $S_1 + D_2 = S_2$, and so on. Since $D_k \leq 1$, we reach one as first point to the right of zero. Then the process stops and we are interested in the average stopping time $E(k^*)$.

Our random walk is homogeneous with respect to time and place. After the first step the distance to 1 equals $1 - D_1$. Besides the first step we have to wait on average $(1-D_1)E(k^*)$ further successful steps until we reach 1 for the first time. Hence,

$$E(k^*) \;=\; 1 + \sum_{d=-n/2+2}^{1} \operatorname{Prob}(D_1 = d)(1-d)E(k^*)$$

$$
\begin{aligned}
&= 1 + E(k^*) - E(k^*) \sum_{d=-n/2+2}^{1} d\mathrm{Prob}(D_1 = d) \\
&= 1 + E(k^*) - E(k^*)E(D_1).
\end{aligned}
$$

We conclude that $E(k^*) = 1/E(D_1)$, and with $E(D_k) \geq 1/2$ for all k we have $E(k^*) \leq 2$. This implies $E(Y_i) \leq 2$ for all i. The equality $E(k^*)E(D_1) = 1$ is known in more general form in probability theory as Wald's identity (Feller (1971)).

Now, we look for some lower bound for p, the probability of a successful mutation. Given p, the average time until we have k successful mutations equals k/p, so we have

$$
\begin{aligned}
E(Y_i + Z_i) &= \sum_k \mathrm{Prob}(Y_i = k) \cdot E(Y_i + Z_i \mid Y_i = k) \\
&= \sum_k \mathrm{Prob}(Y_i = k) \cdot \frac{k}{p} = \frac{E(Y_i)}{p} \leq \frac{2}{p}.
\end{aligned}
$$

During the steps $X_{i-1} + 1, \ldots, X_i$ the number of ones in the left half of x is bounded above by $i - 1$. A sufficient condition for a successful step is that all bits equal to one do not try to flip and exactly one of the bits equal to zero tries to flip. The probability of this event is bounded below by

$$
\binom{n/2 - (i-1)}{1} \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n/2-1} \geq \left(\frac{n}{2} - i + 1\right) \frac{\exp(-1/2)}{n}.
$$

Altogether we have for $n \geq 10$

$$
\begin{aligned}
&\sum_{i=1}^{n/2} E(Y_i + Z_i) \leq 2 \sum_{i=0}^{n/2-1} \left(\left(\frac{n}{2} - i\right) \frac{\exp(-1/2)}{n}\right)^{-1} \\
&= 2\exp(1/2) n \sum_{i=1}^{n/2} \frac{1}{i} \leq 2\exp(1/2) n \ln n = O(n \ln n).
\end{aligned}
$$

The investigation of the second phase can be carried out analogously. Only the probability of a successful step changes, since it is necessary to guarantee that no bit of the left half of x tries to flip. We achieve this by replacing $(1-1/n)^{n/2-1}$ by $(1-1/n)^{n-1}$. This yields the upper bound $2\exp(1) n \ln n$, so we have for $n \geq 10$

$$
E(T) = E(T_1) + E(T_2) \leq 2(\exp(1) + \exp(1/2)) n \ln n = O(n \ln n).
$$

□

For an arbitrary linear function things are a little more complicated. On the one hand a bit that flips from zero to one can have such a large weight that it allows several other bits to flip from one to zero simultaneously, as it is the case for BIN. On the other hand, different to BIN, it may well be true that leading ones are not guaranteed to remain unchanged, as it is the case with ONEMAX. But the main idea from the proof of the upper bound on the expected running time for BIN can be carried over: one can distinguish more and less important bits according to their weights.

Theorem 3.4: The expected running time of the $(1+1)$ EA on the class of linear functions is $\Theta(n \ln n)$.

Proof: In order to prove the statement a lower bound and an upper bound have to be proven. The lower bound follows from Lemma 3.2, so we only have to prove an upper bound on the expected running time of the $(1+1)$ EA on an arbitrary linear function of size $O(n \ln n)$.

Let f be an arbitrary linear function. W. l. o. g. we assume that n is even, all weights w_i are positive integers and the weights are sorted, i. e., $w_1 \geq \cdots \geq w_n$.

We measure the progress the $(1+1)$ EA makes in some value $val : \{0,1\}^n \to \mathbb{R}$, such that val becomes maximal for the optimal all one string. We define

$$val(x) := 2\sum_{i=1}^{n/2} x_i + \sum_{i=n/2+1}^{n} x_i.$$

So, we distinguish between the bits in the left and right half of the current bit string x and consider the bits of the left half to be more important.

Like in the proof of Lemma 3.3 we consider successful steps, but we call every accepted mutation successful, here. We describe a random walk on the line such that the variable describing this random walk grows slower than the function val. We want to estimate the average time until starting at x a bit string x^* with $val(x^*) = val(x) + 1$ is reached. In order to ensure that such a bit string is reached at all, we have to ensure that val increases at most by 1. We can apply Wald's identity if we additionally can prove that the expected value of val increases in each step at least by a constant $c > 0$. It will turn out that $c = 0.08$ is appropriate.

For successful steps there must be at least one bit that flips from zero to one. We consider the leftmost bit x_i flipping from zero to one and distinguish two cases according to the position of this bit.

Case 1: $i \leq n/2$. The bit x_i that belongs to the left half of the bit string flips from zero to one. Taking only this into account the value of val is increased by 2. Since we want to apply Wald's identity and since we are interested in the first point of time when our random walk with respect to val reaches the next point to the right, we like to ensure that val increases at most by 1. Therefore, we let no other bit flip from zero to one, and if no bit flips from one to zero, we choose a bit from the right half and let it flip from one to zero. Because we have $f(x') \geq f(x)$ in successful steps, we are overestimating the number of new zeros.

We assume that each zero bit may flip to one with probability $1/n$. If there are j_1 zeros in the left half and j_2 zeros in the right half, the number of flipping zeros is on the average j_1/n in the left and j_2/n in the right half. This decreases the value of val on average by $(2j_1 + j_2)/n$. Only if no bit flips, val increases by 2. In this case we decrease val by 1 by flipping one bit in the right half from one to zero. This happens with probability

$(1-1/n)^{j_1+j_2}$. Altogether

$$E\left(val(x')-val(x)\right) \geq 2-\frac{2j_1+j_2}{n}-\left(1-\frac{1}{n}\right)^{j_1+j_2}.$$

This function takes its minimum for maximal values of j_1 and j_2. Since $j_1 \leq n/2-1$ and $j_2 \leq n/2$, we obtain for $n \geq 5$

$$E\left(val(x')-val(x)\right) \geq 2-\frac{3}{2}-\left(1-\frac{1}{n}\right)^{n-1} > 0.08.$$

Case 2: $i > n/2$. In this case there is no bit in the left half that flips from zero to one. Since we consider an accepted mutation we have $f(x') \geq f(x)$. We assume the weights w_i to be sorted, so it follows that the number of bits of the right half that flip from zero to one is an upper bound on the number of bits of the left half that may flip from one to zero. From the right half all bits may flip from one to zero.

If k denotes the number of zeros in x, we see that the worst case with respect to val is given by

$$x_{n/2+1} = \cdots = x_{n/2+k} = 0,$$
$$w_1 = \cdots = w_{n/2} = \cdots = w_{n/2+k} = n, \quad w_{n/2+k+1} = \cdots = w_n = 1.$$

Let $j+1$ be the number of bits flipping from zero to one. Then we have the following two upper bounds on the number of bits that flip from one to zero.

1. At most $j+1$ bits from the left half can flip but no bit from the right half.
2. At most j bits from the left half can flip and all bits from the right half.

We work under the assumption that $x_{n/2+1}$ flips from zero to one. Let L_0 be the random number of bits from the left half that flip from one to zero, let R_0 be the random number of bits from the right half that flip from one to zero. Finally, let Z be the random number of bits among $x_{n/2+2}, \ldots, x_{n/2+k}$ that flip from zero to one. Since we assume that the leftmost bit that flips from zero to one belongs to the right half, we may assume that all zeros in the current bit string are among the bits of the right half and $k \leq n/2$ holds. Hence,

$$E\left(val(x')-val(x)\right) = \sum_{j=0}^{n/2-1} \mathrm{Prob}(Z=j) \cdot E\left(val(x')-val(x) \mid Z=j\right).$$

We distinguish the two cases according to L_0 as described above.

In the first case we have

$$E\left(val(x')-val(x) \mid Z=j\right) = j+1-2E\left(L_0 \mid L_0 \leq j+1\right).$$

Obviously,

$$E\left(L_0 \mid L_0 \leq j+1\right) \leq E(L_0) \leq \frac{1}{2}$$

and

$$E(L_0 \mid L_0 \leq 0) = 0.$$

We know that

$$\begin{aligned}
E(L_0 \mid L_0 \leq 1) &= \text{Prob}(L_0 = 1 \mid L_0 \leq 1) \\
&= \frac{\text{Prob}(L_0 = 1)}{\text{Prob}(L_0 \leq 1)} \\
&= \frac{\binom{n/2}{1}(1/n)(1-1/n)^{n/2-1}}{(1-1/n)^{n/2} + \binom{n/2}{1}(1/n)(1-1/n)^{n/2-1}} \\
&= \frac{1}{2(1-1/n)+1} \leq 0.43
\end{aligned}$$

if $n \geq 3$. So, we have

$$\begin{aligned}
E\left(val(x') - val(x)\right) &= \sum_{j=0}^{n/2-1} \text{Prob}(Z=j) E\left(val(x') - val(x) \mid Z = j\right) \\
&\geq \text{Prob}(Z=0)(1 - 2 \cdot 0.43) + \sum_{j=1}^{n/2-1} \text{Prob}(Z=j) \cdot 0 \\
&= 0.14 \cdot \text{Prob}(Z=0) \geq 0.14 \cdot \exp(-1/2) > 0.08.
\end{aligned}$$

In the second case we have

$$E\left(val(x') - val(x) \mid Z = j\right) = j + 1 - 2E(L_0 \mid L_0 \leq j) - E(R_0).$$

Obviously,

$$E(L_0 \mid L_0 \leq j) \leq E(L_0) \leq \frac{1}{2}$$

and

$$E(R_0) \leq \frac{1}{2}.$$

So, we have

$$\begin{aligned}
E\left(val(x') - val(x)\right) &= \sum_{j=0}^{n/2-1} \text{Prob}(Z=j) E\left(val(x') - val(x) \mid Z = j\right) \\
&\geq \text{Prob}(Z=0)(1-0.5) \\
&\quad + \sum_{j=1}^{n/2-1} \text{Prob}(Z=j) \cdot (j + 1 - 2E(L_0 \mid L_0 \leq j) - 0.5) \\
&\geq 0.5 \cdot \exp(-1/2) + 0 > 0.08.
\end{aligned}$$

Now, by Wald's identity the average time that our random walk starting at some point $a = val(x)$ reaches $a+1$ for the first time is bounded by a constant. The value val has to increase at most by $(3/2)n$, since 0 is the minimal and $(3/2)n$ the maximal function value of val. The probability for a successful step can be estimated like in Lemma 3.3. If the current value of x is d away from the maximal value $(3/2)n$, there are at least $d/2$ zeros in x. Hence, we have to consider each of the cases of k zeros, $0 \leq k \leq n$, twice. So, altogether we have the upper bound $O(n \ln n)$. □

We see that the $(1+1)$ EA optimizes linear functions quite efficiently. One may ask whether the performance depends on the mutation probability p_m, whether the performance can be improved substantially by using other values then $1/n$ for p_m.

The "correct" mutation probability has already been subject to some research, see, e. g., Bäck (1993). The choice of $p_m = 1/n$ is the most often recommended one, but the reasoning is basically based on experimental experience. We prove here that mutation probabilities of c/n for positive constants c are optimal for linear functions and that much smaller or larger probabilities increase the expected running time.

Setting p_m to c/n for any positive constant c does not change the order of growth of the expected running time. For $c \leq 1$ the proof of Theorem 3.4 remains valid without any change. For $c > 1$ we can carry out an analogous analysis using $\lceil c \rceil + 1$ phases instead of two.

For much smaller mutation probabilities we expect that the waiting time until all bits have tried to flip becomes too long. For much larger probabilities in each step on average a lot of bits try to flip, so we expect the probabilities of successful steps to become too small. We justify this reasoning by two formal statements.

Theorem 3.5: The $(1+1)$ EA with mutation probability $p_m = (\alpha(n)n)^{-1}$, where $\alpha(n) \to \infty$ for $n \to \infty$, needs on average $\Omega(\alpha(n)n \ln n)$ steps until it reaches the optimal value of a linear function with positive weights, if it starts at $x = (0, \ldots, 0)$.

Proof: Like in the proof of Lemma 3.2 let T be the random variable describing the first point of time where each bit has tried to flip. If $\text{Prob}(T \geq t) \geq c$ for some constant c and all $t \leq \alpha(n)n \ln n - \ln n$, then the theorem follows. Let $t = (n\alpha(n) - 1) \ln n$. Then $(1 - 1/n\alpha(n))^t \geq e^{-\ln n} = 1/n$ and

$$\text{Prob}(T \geq t+1) = 1 - \left(1 - \left(1 - \frac{1}{n\alpha(n)}\right)^t\right)^n \geq 1 - \left(1 - \frac{1}{n}\right)^n \geq 1 - \exp(-1).$$

□

Theorem 3.6: The $(1+1)$ EA with mutation probability $p_m = \alpha(n)/n$, where $\alpha(n) \to \infty$ for $n \to \infty$, needs on average $\exp(\Omega(\alpha(n)))n \ln n = \Omega(\alpha(n)n \ln n)$ steps until it reaches for BIN the optimal value, if it starts at $x = (x_1, \ldots, x_n)$ where $x_i = 1$, if $i \leq n/2$, and $x_i = 0$, if $i > n/2$.

Proof: Because of the definition of BIN a step only can be successful, if none of the leading $n/2$ bits tries to flip. The probability that a step is successful is, therefore, bounded above by $(1-\alpha(n)/n)^{n/2} = \exp\left(-\Omega\left(\alpha(n)\right)\right)$. We prove that the average number of successful steps is $\Omega\left(n(\ln n)/\alpha(n)\right)$. Hence, the average total run time is

$$\Omega\left(\left(\exp\left(\Omega\left(\alpha(n)\right)\right) n \ln n\right)/\alpha(n)\right) = \exp\left(\Omega\left(\alpha(n)\right)\right) n \ln n.$$

For the lower bound on the number of successful steps it is sufficient to remark that each of the $n/2$ not leading bits has to flip in at least one successful step. Similarly to the proof of Theorem 3.5 we choose $t = (n/\alpha(n) - 1)\ln n$. Then $(1-\alpha(n)/n)^t \geq 1/n$ and

$$\mathrm{Prob}(T \geq t+1) = 1 - \left(1 - \left(1 - \frac{\alpha(n)}{n}\right)^t\right)^{n/2} \geq 1 - \left(1 - \frac{1}{n}\right)^{n/2} \geq 1 - \exp(-1/2).$$

Since we only are considering $n/2$ bits the outer exponent is here $n/2$ instead of n leading to the constant $1-\exp(-1/2)$ instead of $1-\exp(-1)$. □

4 Unimodal functions

Linear functions are our first example of a class of functions where the $(1+1)$ EA is expected to find the global optimum quite efficiently. Of course, we would like to identify more and larger classes of functions where some polynomial upper bound on the expected running time can be given. Since we know from Section 2 that already functions with degree 2 can be most difficult for the $(1+1)$ EA, we need some other criterion to recognize a fitness function as easy.

In this section we consider unimodal functions. Since we are considering fitness functions with Boolean inputs, it is not totally obvious how "unimodal" should be defined. In fact, there are different definitions in the literature which yield quite different classes of functions. Here, we use that definition which appears to be the most natural one.

Definition 4.1: Let $f : \{0,1\}^n \to \mathbb{R}$ be a fitness function. We call $x \in \{0,1\}^n$ a *local maximum*, iff

$$\forall y \in \{0,1\}^n : H(x,y) = 1 \Rightarrow f(y) \leq f(x).$$

The function f is *unimodal* iff f has exactly one local maximum.

Obviously, all linear functions with all nonzero weights are unimodal. Moreover, the most striking similarity between unimodal and linear functions is that for all $x \in \{0,1\}^n$ that are not the global optimum there is always another point with Hamming distance one with greater fitness. So there is always a mutation of exactly one bit that improves the function value. This leads Mühlenbein (1992) to the remark that all unimodal functions can be optimized by the $(1+1)$ EA in expected $O(n \log n)$ steps. We disprove this claim in this section.

4.1 A quadratic lower bound for a unimodal fitness function

Already Rudolph (1997) doubts Mühlenbein's claim and presents a fitness function and believes that the $(1+1)$ EA has expected running time $\Theta(n^2)$ on that function. He proves an upper bound of $O(n^2)$ steps and presents experiments that confirm the lower bound, but does not give a formal proof. We use his example and present the lower bound.

Definition 4.2: The function $\textsc{LeadingOnes} : \{0,1\}^n \to \mathbb{R}$ is defined by

$$\textsc{LeadingOnes}(x_1, \ldots, x_n) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j.$$

The function value of $\textsc{LeadingOnes}(x)$ equals the number of leading ones in x. Since the function value can always be increased by appending a single one to the leading ones, $\textsc{LeadingOnes}$ is obviously unimodal. The $(1+1)$ EA accepts a mutation iff the number of leading ones is not decreased by this mutation. It is quite obvious that the expected number of steps for the $(1+1)$ EA on $\textsc{LeadingOnes}$ is $O(n^2)$. There is always exactly one position in the bit string that has to flip to increase the number of leading ones, namely the bit with value zero that follows immediately after the leading bits with value one. Waiting for this zero to flip while simultaneously no bit of the leading ones flips takes on average time $\Theta(n)$, so the upper bound $O(n^2)$ follows. Since after random initialization the expected number of ones is $n/2$ and in successful steps the number of ones in the bit string is on average not further increased, it may be supposed that we need about $n/2$ steps, so the lower bound of $\Omega(n^2)$ would follow. We make this idea precise.

Theorem 4.3: The expected running time for the $(1+1)$ EA on $\textsc{LeadingOnes}$ is $\Theta(n^2)$.

Proof: We repeat the easy proof of the upper bound. The number of leading ones is never decreased. So we ignore steps where the number of leading ones remains unchanged and assume that no other ones are in the bit string. This can only increase the expected running time. We have to wait for at most n steps where the number of leading bits is increased by at least 1, each with probability at least $(1/n)(1-1/n)^i$, if i denotes the current number of leading bits. As upper bound on the expected running time we get

$$\sum_{i=0}^{n-1} n(1-1/n)^{-i} < \exp(1)n^2 = O(n^2).$$

Now we need a lower bound. We distinguish two different types of accepted mutations. If the number of leading ones is increased, we call such a mutation important. The other accepted mutations are called unimportant.

Let T denote the number of steps until the $(1+1)$ EA reaches the optimal all one string. Let T_i denote the number of steps between the $(i-1)$-th and i-th important mutation. Let M denote the number of important mutations until the optimum is reached. Obviously,

$M \leq n$ holds. So we have

$$\begin{aligned} E(T) &= E\left(\sum_{i=1}^{M} T_i\right) = \sum_{j=1}^{n} E\left(\sum_{i=1}^{M} T_i \mid M = j\right) \cdot \mathrm{Prob}(M = j) \\ &= \sum_{j=1}^{n}\sum_{i=1}^{j} E(T_i \mid M = j)\mathrm{Prob}(M = j) \end{aligned}$$

for the expected number of steps. The probability for an important mutation is at most $1/n$, so $E(T_i \mid M = j) \geq n$ holds for all values of i. We conclude that

$$E(T) \geq \sum_{j=1}^{n}\sum_{i=1}^{j} n\mathrm{Prob}(M = j) \geq cn^2\mathrm{Prob}(M \geq cn)$$

holds for any constant c with $0 < c < 1$. If we can prove, that after random initialization and cn important mutations the probability, that there is at least one zero in the current bit string, is bounded below by some constant greater than 0, then we have proven that $E(T) = \Omega(n^2)$.

We distinguish the ones in the current bit string after i important mutations according to their origins.

(1) The initially chosen bit string contains ones.

(2) Important mutations create new ones. First, at least one new leading one is appended. Second, more new ones that may or may not belong to the leading ones can be created.

(3) Finally, unimportant mutations can create eventually new ones that do not belong to the leading ones.

We now derive upper bounds on the number of ones created in the different cases.

Lemma 4.4: After random initialization the number of ones is less than $(1/2+c_1)n$ with probability at least $1-(\exp(2c_1)/(1+2c_1)^{1+2c_1})^{n/2}$ for any constant c_1 with $0 < c_1 < 1/2$.

Proof: All bits are chosen independently as one or zero with probability 1/2. Therefore, the expected number of ones is $n/2$. We use Chernoff's bounds (Hagerub and Rüb (1989)) and see that the probability, that the number of ones a is less than $(1/2 + c_1)n$, can be bounded by

$$\mathrm{Prob}(a < (1/2 + c_1)n) = 1 - \mathrm{Prob}(a \geq (1 + 2c_1)n/2) \geq 1 - \left(\frac{\exp(2c_1)}{(1 + 2c_1)^{1+2c_1}}\right)^{n/2}.$$

□

Lemma 4.5: In c_2n important mutations the number of ones is increased by less than $(c_2 + c_3)n$ with probability at least

$$1 - \left(\frac{\exp(c_3/c_2 - 1)}{(c_3/c_2)^{c_3/c_2}}\right)^{c_2 n}$$

for any constants c_2 and c_3 with $c_3 > c_2 > 0$.

Proof: In each important mutation the number of leading ones is increased by at least 1. All the bits behind that new leading one may flip. We are at most overestimating the number of new ones if we assume every flipping bit to become a one. We denote the number of such flipping bits in c_2n steps by m. Since these bits are independent of each other, the expectation of m is bounded above by c_2n. We use Chernoff's bounds to derive a lower bound for the probability that m is less than c_3n. We use $c_3n = (1+(c_3/c_2)-1)c_2n$ and therefore need $(c_3/c_2) - 1$ to be a positive number. Since we assume $c_3 > c_2$ to hold, this is always true. So we have

$$\mathrm{Prob}(m < c_3 n) = 1 - \mathrm{Prob}(m \geq (1 + c_3/c_2 - 1)c_2 n) \geq 1 - \left(\frac{\exp(c_3/c_2 - 1)}{(c_3/c_2)^{c_3/c_2}}\right)^{c_2 n}.$$

□

Lemma 4.6: The probability, that k bits that are initializied randomly and only are subject to unimportant mutations, all have the value one, is 2^{-k}.

Proof: After random initialization k bits all have the value one with probability 2^{-k}. Since unimportant mutations are always accepted, this probability does not change. □

We now combine the different bounds to obtain the desired result. We choose the constants c_1, c_2, and c_3 in such a way that $1/2 + c_1 + c_2 + c_3 < 1$ and $c_2 < c_3$ hold. We see that after c_2n important mutations we still have at least $(1/2 - c_1 - c_2 - c_3)n$ zeros with probability

$$\left(1 - \left(\frac{\exp(2c_1)}{(1+2c_1)^{1+2c_1}}\right)^{n/2}\right) \cdot \left(1 - \left(\frac{\exp(c_3/c_2 - 1)}{(c_3/c_2)^{c_3/c_2}}\right)^{c_2 n}\right),$$

if we neglect the ones that are created in unimportant mutations. In the worst case these are all leading ones. Then we have $(1/2 - c_1 - c_2 - c_3)n - 1$ bits, which are candidates for new ones due to unimportant steps. So according to Lemma 4.6 with probability $1 - 2^{-(1/2 - c_1 - c_2 - c_3 - 1/n)n}$ there is at least one zero in these bits. It follows that with probability converging exponentially fast to 1, after c_2n important steps there is still at least one zero in the current bit string. This proves $E(T) = \Omega(n^2)$, so together with the upper bound we have $E(T) = \Theta(n^2)$. □

4.2 An exponential lower bound for a unimodal fitness function

It is obvious that the $(1+1)$ EA reaches the optimum of a unimodal function with k different function values after $O(kn)$ steps (Rudolph (1997)). For unimodal functions there is always at least one possible mutation that increases the function value and requires only the mutation of exactly one bit. Since such mutations have probability $(1/n)(1-1/n)^{n-1}$, the expected time until such a mutation occurs is bounded above by $\exp(1)n$. At most k such mutations are sufficient, so the bound $O(kn)$ follows. To enforce large expected running times on unimodal functions one needs a fitness function where only a large number of 1 bit mutations is sufficient to reach the optimum by such "small steps". The idea is to construct functions where only a small "path" to the optimum exists: a path is a sequence of bit strings that are all reachable via mutations of exactly one bit such that this mutation is the only mutation of exactly one bit that is accepted. If this path is exponentially long, one may hope that the $(1+1)$ EA needs exponentially many steps to find the optimum in the expected case.

Such long paths were introduced by Horn, Goldberg, and Deb (1994). They performed several experiments and compared the $(1+1)$ EA and some other hillclimbers with a Genetic Algorithm. Though they were convinced to observe exponential running times of the $(1+1)$ EA, Rudolph (1997b) proved that the expected running time is only $O(n^3)$. The problem is that already a mutation of two bits simultaneously enables the $(1+1)$ EA to take a shortcut and this reduces the number of required steps dramatically. To overcome this problem Rudolph (1997) formally defines a more general version of long paths (already informally described by Horn, Goldberg, and Deb (1994)), such that no mutation of at most k bits is sufficient to take a short cut. These paths are called long k-paths. The parameter k can be chosen, though with increasing k the length of the path decreases. We start with defining long k-paths and a few simple statements about them that are taken from Rudolph (1997).

Definition 4.7: Let $n \geq 1$ hold. For all $1 < k$ that fulfill $(n-1)/k \in \mathbb{N}$ the long k-path of dimension n is a sequence of bit strings from $\{0,1\}^n$. The long k-path of dimension 1 is defined as $P_1^k := (0,1)$. The long k-path of dimension n is defined using the long k-path of dimension $n-k$ as basis as follows. Let the long k-path of dimension $n-k$ be given by $P_{n-k}^k = (v_1, \ldots, v_l)$. Then we define the sequences of bit strings S_0, B_n, and S_1 from $\{0,1\}^n$, where $S_0 := (0^k v_1,\ 0^k v_2,\ \ldots, 0^k v_l)$, $S_1 := (1^k v_l,\ 1^k v_{l-1},\ \ldots, 1^k v_1)$, and $B_n := (0^{k-1}1 v_l, 0^{k-2}11 v_l, \ldots, 01^{k-1} v_l)$. The points in B_n build a bridge between the points in S_0 and S_1, that differ in the k leading bits. Therefore, the points in B_n are called *bridge points*. The resulting long k-path P_n^k is constructed by appending S_0, B_n, and S_1, so P_n^k is a sequence of $\left|P_n^k\right| = |S_0| + |B_n| + |S_1|$ points. We call $\left|P_n^k\right|$ the length of P_n^k. The i-th point on the path P_n^k is denoted as p_i, p_{i+j} is called the j-th successor of p_i.

The recursive definition of long k-paths allows us to determine the length of the paths easily.

Lemma 4.8: The long k-path of dimension n has length $\left|P_n^k\right| = (k+1)2^{(n-1)/k} - k + 1$. All points of the path are different.

Proof: For $n = 1$ the length is $(k+1)2^0 - k + 1 = 2$. Let the statement hold for values smaller than n. By definition of long k-paths we have $\left|P_n^k\right| = 2\left|P_{n-k}^k\right| + k - 1 = 2(k+1)2^{(n-k-1)/k} - 2k + 2 + k - 1 = (k+1)2^{(n-1)/k} - k + 1$. By definition all points on the path are different. $\square$

The most important property of long k-paths are the regular rules that hold for the Hamming distances between each point and its successors.

Lemma 4.9: Let n and k be given such that the long k-path P_n^k is well defined. For all i with $0 < i < k$ the following holds. If $x \in P_n^k$ has at least i different successors on the path then the i-th successor of x has Hamming distance i of x and all other points on the path that are successors of x have Hamming distances different from i.

Proof: The statement is obviously true for $n = 1$ and all values of k. Assume that it holds for P_{n-k}^k. We know that P_n^k is constructed by appending S_0, B_n, and S_1, with $\left|P_n^k\right| = (k+1)2^{(n-1)/k} - k + 1$, $|S_0| = |S_1| = (k+1)2^{(n-k-1)/k} - k + 1$, and $|B_n| = k - 1$. Let x be the p-th point of P_n^k. We distinguish several cases according to the value of p.

If $p < |S_0|$ and $p + i \leq |S_0|$, then the statement holds by assumption, since S_0 has the same structure as P_{n-k}^k.

If $p \leq |S_0|$ and $|S_0| < p + i \leq |S_0| + |B_n|$, then the Hamming distance from x to the last point in S_0 is $|S_0| - p$ which is at least 0 and less than k by the assumption $0 < i < k$. By definition of B_n the j-th point in B_n differs from the last point of S_0 in j bits, so there is exactly one point in B_n with Hamming distance i. All points in S_1 have greater Hamming distance, since the first k bits of points in S_0 and S_1 are all different.

If $|S_0| < p < |S_0| + |B_n|$ and $p + i \leq |S_0| + |B_n|$, the statement is obviously true. All points in B_n just differ on the fist k bits, the Hamming distance of any point to its j-th successor is j so x has exactly one successor in B_n with Hamming distance i. All points in S_1 have greater Hamming distance.

If $|S_0| < p \leq |S_0| + |B_n|$ and $|S_0| + |B_n| < p + i$ hold, then the situation is essentially the same as for the second case. The parts S_1 and S_0 have the same structure, only the k leading bits differ and the ordering of S_1 is reversed. So the same remarks apply.

Finally, if $|S_0| + |B_n| < p$, the statement holds by assumption, since S_1 has the same structure as P_{n-k}^k. $\square$

We are interested in unimodal fitness functions, so for a fitness function $f : \{0,1\}^n \to \mathbb{R}$ exactly 2^n function values have to be defined. Since the long k-path of dimension n consists of only $(k+1)2^{(n-1)/k} - k + 1$ points, we have to embed this path in a unimodal function. The following definition differs from the one given by Rudolph (1997) in the way that bit strings not belonging to the long k-path are treated. This little modification helps us to establish the lower bound while it does not matter for the upper bounds that were already given by Rudolph.

Definition 4.10: Let $n \geq 1$ hold, let $1 < k$ be given such that k fulfills $(n-1)/k \in \mathbb{N}$. The long k-path function of dimension n is called $\text{PathFunction}_k : \{0,1\}^n \to \mathbb{N}$ and is defined by

$$\text{PathFunction}_k(x) = \begin{cases} n^2 + l & \text{if } x \text{ is the } l\text{-th point of } P_n^k \\ n^2 - n\sum_{i=1}^{k} x_i - \sum_{i=k+1}^{n} x_i & \text{if } x \notin P_n^k \end{cases} .$$

We have already mentioned that a fitness function f is unimodal iff for all points $x \in \{0,1\}^n$ we have that either x is the global maximum or there exists $y \in \{0,1\}^n$ with $H(x,y) = 1$, such that $f(x) < f(y)$ holds. For PathFunction_k this is obviously the case. For all points on the path except the last one, which is the global optimum, this holds, since there is always 1 successor on the path with Hamming distance exactly 1 according to Lemma 4.9. For points not on the path decreasing the number of ones by 1 always yields a bit string with increased function value. By Definition 4.7 it is obvious, that the all zero bit string is the first point on the path.

Rudolph (1997) establishes two different upper bounds on the expected running time that both yield exponential values for $k = \sqrt{n-1}$, so he speculates that the expected running time may in fact be exponential for this choice of k. We prove this here, and thereby answer the open question, whether unimodal functions exists, on which the $(1+1)$ EA has exponential expected running time.

Lemma 4.11 (Rudolph (1997)): The expected running time of the $(1+1)$ EA on PathFunction_k is bounded above by $O\left(n^{k+1}/k\right)$ and $O\left(n\left|P_n^k\right|\right)$.

Proof: We distinguish two different ways to reach the optimum. If we assume that the $(1+1)$ EA advances by mutations of exactly one bit only, we see that at most $\left|P_n^k\right| + n$ such mutations are necessary, so the upper bound $O\left(n\left|P_n^k\right|\right)$ follows. If we assume that mutations of k bits flipping simultaneously are taken to reach the optimum, i. e., the $(1+1)$ EA takes shortcuts, then we notice that n/k such mutations are sufficient, implying the other upper bound. □

Theorem 4.12: The expected running time of the $(1+1)$ EA on $\text{PathFunction}_{\sqrt{n-1}}$ is $\Theta\left(n^{3/2}2^{\sqrt{n}}\right)$.

Proof: For $k = \sqrt{n-1}$ we have $\left|P_n^{\sqrt{n-1}}\right| = (\sqrt{n-1}+1)2^{\sqrt{n-1}} - \sqrt{n-1} + 1$ according to Lemma 4.8, so the upper bound follows directly from Lemma 4.11.

The idea for the proof of the lower bound is roughly speaking the following. We assume that the $(1+1)$ EA reaches the path somewhere in the first half of the bridge points. If only mutations with at most $\sqrt{n-1}-1$ bits flipping simultaneously occur, the $(1+1)$ EA has to follow the long path, so the expected number of steps is about $n\left|P_{\sqrt{n-1}}^n\right|/2$.

Let T denote the number of steps until the $(1+1)$ EA reaches the optimum. Let T_i denote the number of steps until the $(1+1)$ EA reaches the optimum, if it is started in the i-th point on the path. Let E_t be the event that in t steps no mutation of at least $\sqrt{n-1}$

simultaneously flipping bits occurs. Since the future steps of the $(1+1)$ EA depend only on the current state and not on the "history" of the current run, T_i describes the number of steps until the optimum is reached after the i-th point of the path is reached, too. We use the notation "$i \in I$" to describe the event that the $(1+1)$ EA reaches at least one of the points on the path with index in I. By I_m we denote the set of integers from 1 to m, i. e., $I_m = \{1, \ldots, m\}$.

By Definition 4.7 we have that $\left|P_n^{\sqrt{n-1}}\right| = 2\left|P_{n-\sqrt{n-1}}^{\sqrt{n-1}}\right| + \sqrt{n-1} - 1$ holds for the length of $P_n^{\sqrt{n-1}}$. We want to estimate the expected running time of the $(1+1)$ EA if the first point on the path does not belong to S_1, i. e., it does belong to the first $a := \left|P_{n-\sqrt{n-1}}^{\sqrt{n-1}}\right| + \sqrt{n-1} - 1$ points on $P_n^{\sqrt{n-1}}$.

Therefore, we have

$$E(T) \quad \geq \quad \text{Prob}\,(i \in I_a) \cdot \min\left\{E\left(T_i \mid E_t\right) \mid i \in I_a\right\} \cdot \text{Prob}\,(E_t)\,.$$

We start with estimations for the two probabilities. For any k, the probability that at least k bits mutate simultaneously is bounded above by $\binom{n}{k}n^{-k}$, so this happens at least once in t steps with probability at most $t\binom{n}{k}n^{-k}$. We can use $\binom{n}{k} \leq n^k/k!$ and Stirling's formula for the bound $\sqrt{2\pi k}k^k/\exp(k) \leq k!$ to get $t\exp(k)/(\sqrt{2\pi k}k^k)$ as upper bound. So we have

$$\text{Prob}\,(E_t) \geq 1 - \frac{t\exp(\sqrt{n-1})}{\sqrt{2\pi\sqrt{n-1}}\sqrt{n-1}^{\sqrt{n-1}}}$$

for $k = \sqrt{n-1}$.

We know that P_n^k is constructed from S_0, S_1, and B_n as described in Definition 4.7. We are interested in the first point on the path that the $(1+1)$ EA reaches after random initialization. In particular we are looking for a lower bound on the probability that this point belongs to S_0 or B_n.

If the initial bit string is on the path, then the probability that this string does not belong to S_1 is $1 - \left|P_{n-k}^k\right| / \left|P_n^k\right| > 1/2$. Now we are left with the case that the initial bit string is off the path. In this case a mutation is accepted iff

1. the path is reached,
2. the number of ones in the first k bits is decreased, or
3. the number of ones in the first k bits remains unchanged, and the number of ones in the bit string is not increased.

Since the probability of reaching a point in S_1 as first point on the path decreases with increasing number of zeros in the first k bits, we are overestimating the probability of first reaching S_1, if we assume that no mutation steps off the path are accepted. After random initialization before the first step the probability of reaching S_0 is obviously equal to the probability of reaching S_1 for symmetry reasons. It follows that the probability that the first bit string on the path belongs to S_0 or B_n is greater than 1/2. So in any case we have $\text{Prob}\,(i \in I_a) > 1/2$.

Now we need a lower bound for $E(T_i \mid E_t)$ for $i \in I_a$. We call the number of points on the path between the i-th point (our starting point) and the global optimum the distance d and have $d = \left|P_n^{\sqrt{n-1}}\right| - i + 1$. We denote the number of points between the i-th point on the path and the current point on the path after j steps by a_j. Using these notions we have

$$\begin{aligned} E\,(T_i \mid E_t) &\geq t \cdot \text{Prob}\,(T_i \geq t \mid E_t) \\ &= t \cdot \text{Prob}\,(a_t < d \mid E_t) \\ &= t \cdot (1 - \text{Prob}\,(a_t \geq d \mid E_t)) \,. \end{aligned}$$

Using Markoff's inequality we get

$$E\,(T_i \mid E_t) \geq t \cdot \left(1 - \frac{E\,(a_t \mid E_t)}{d}\right).$$

The most important property of long $\sqrt{n-1}$-paths is that a mutation of j bits simultaneously (with $j < \sqrt{n-1}$) implies an advance of at most j points (Lemma 4.9). This yields

$$E\,(a_t \mid E_t) \leq t \cdot E\,(a_1 \mid E_t)\,,$$

so we have

$$E\,(T_i \mid E_t) \geq t \cdot \left(1 - \frac{t \cdot E\,(a_1 \mid E_t)}{d}\right).$$

We use

$$E\,(a_1 \mid E_t) \leq \frac{2}{n},$$

which can be proven as follows.

Since we want an upper bound for the expected gain in one step under the condition that only mutations with less than $\sqrt{n-1}$ simultaneously flipping bits occur, we have

$$E\,(a_1 \mid E_t) = \sum_{i=1}^{\sqrt{n-1}-1} i \cdot \text{Prob}(M_i),$$

if M_i denotes the event that the only accepted mutation of exactly i simultaneously flipping bits occurs. According to Lemma 4.9 there is always exactly one accepted mutation of exactly i bits. It follows that the probability of the "correct" i-bit mutation equals $n^{-i}(1-1/n)^{n-i}$. This implies

$$E\,(a_1 \mid E_t) \leq \sum_{i=1}^{\sqrt{n-1}-1} \frac{i}{n^i} < \sum_{i=1}^{\infty} \frac{i}{n^i} = \sum_{i=1}^{\infty}\sum_{j=i}^{\infty} n^{-j} = \frac{n}{(n-1)^2} \leq \frac{2}{n},$$

for $n \geq 4$. The last equality follows from an easy calculation using the fact that $\sum_{s=0}^{t-1} q^s = (1-q^t)/(1-q)$ holds for $|q| < 1$ and all integers $t > 1$.

So we have

$$E(T_i \mid E_t) \geq t \cdot \left(1 - \frac{2t}{nd}\right).$$

We know that $d = \left|P_n^{\sqrt{n-1}}\right| - i + 1$ and $i \leq \left|P_{n-\sqrt{n-1}}^{\sqrt{n-1}}\right| + \sqrt{n-1} - 1$, so we have

$$d \geq \left|P_{n-\sqrt{n-1}}^{\sqrt{n-1}}\right| + 2 = \left(\sqrt{n-1}+1\right) 2^{\sqrt{n-1}-1} - \sqrt{n-1} + 1.$$

For $t = n^{3/2} 2^{\sqrt{n}-5}$ we get

$$E(T_i \mid E_t) = \Omega\left(n^{3/2} 2^{\sqrt{n}}\right)$$

and together with the upper bound this finishes the proof. $\square$

5 Enforcing expected running times

We have a class of fitness functions which the $(1+1)$ EA optimizes efficiently, and we know several examples where the expected running time of the $(1+1)$ EA is exponential. In this section we define n different fitness functions $\text{JUMP}_1, \ldots, \text{JUMP}_n$, where the expected running time is $\Theta(n^m + n \log n)$ for JUMP_m. So we can enforce a large variety of different expected running times. This result can be interpreted as a hierarchy result for the performance of the $(1+1)$ EA. Each additional factor n for the expected time enlarges the class of functions which can be optimized. The construction of the functions JUMP_m is done in a way that shows the understanding of the way Algorithm 1.1 works. The main idea is to use a construction similar to the TRAP-function, but to adjust the distance between the local and the global maximum, so that the expected running time, that is dominated by the time for the jump over that distance, can be controlled. The function JUMP_m, which is given in the following definition, is visualized in Figure 2 for $n = 40$ and $m = 6$. With growing m the size of the gap widens, for $m = n$ the function JUMP_n equals TRAP. For $m = 1$ we get the linear function ONEMAX.

Definition 5.1: Given n with $n > 1$ and $m \in \{1, 2, \ldots, n\}$, let the function $\text{JUMP}_m : \{0,1\}^n \to \mathbb{R}$ be defined by

$$\text{JUMP}_m(x) := \begin{cases} m + \sum_{i=1}^n x_i & \text{if } \sum_{i=1}^n x_i \leq n - m \text{ or } \sum_{i=1}^n x_i = n \\ n - \sum_{i=1}^n x_i & \text{otherwise} \end{cases}.$$

Theorem 5.2: The expected running time of the $(1+1)$ EA on JUMP_m is $\Theta(n^m + n \log n)$ for $m \in \{1, 2, \ldots, n\}$.

Proof: As we mentioned above, for $m = 1$ the function JUMP_1 essentially equals ONEMAX (in fact it gives ONEMAX $+ 1$), so the expected running time $\Theta(n \log n)$ follows from the results of Section 3.

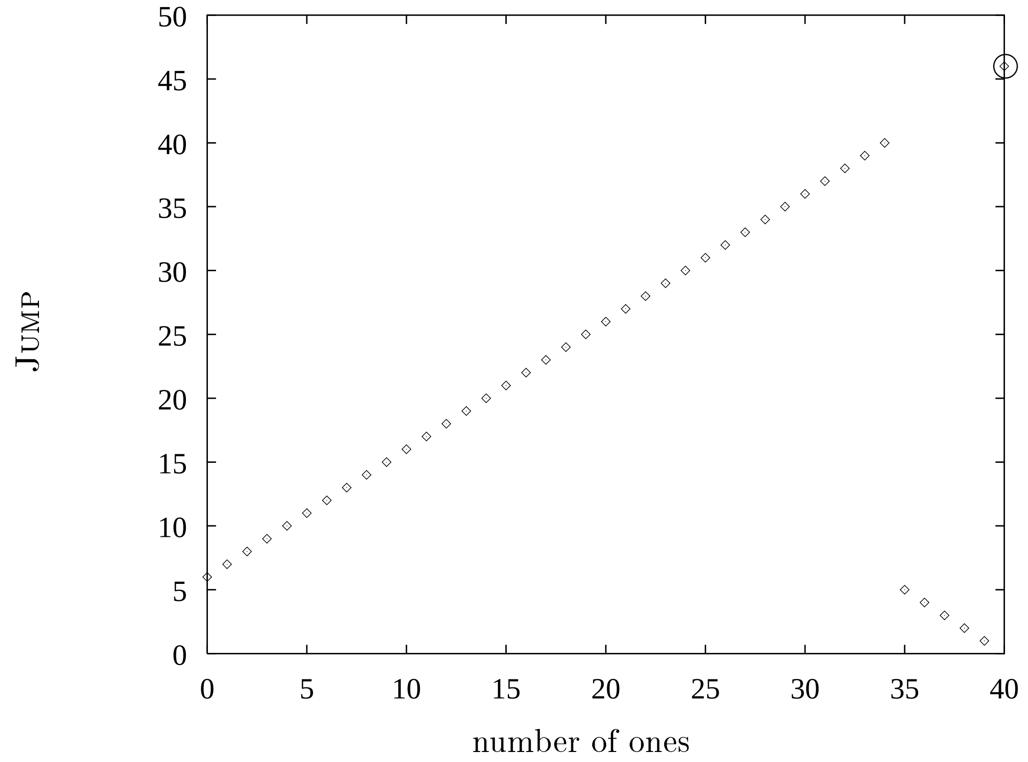

Figure 2: The function JUMP_m

Now we assume that $m > 1$ holds. We partition the search space into three disjoint sets A_1, A_2, and A_3 with

$$
\begin{aligned}
A_1 &:= \left\{x \in \{0,1\}^n \mid n - m < \sum_{i=1}^{n} x_i < n\right\} \\
A_2 &:= \left\{x \in \{0,1\}^n \mid \sum_{i=1}^{n} x_i \leq n - m\right\} \\
A_3 &:= \{(1, \ldots, 1)\}\,.
\end{aligned}
$$

The definition of the sets is done in a way that ensures that bit strings in A_i have greater function values as bit strings in A_j, iff $i > j$ holds. So after reaching some bit string in A_i the $(1+1)$ EA can only reach bit strings in A_j with $j \geq i$. Since JUMP_m is a symmetric function, we can ignore steps where the number of ones in the bit string remains unchanged.

We begin with an upper bound on the expected running time. The expected number of steps until the $(1+1)$ EA reaches a bit string with exactly m zeros or the global optimum is $O(n \log n)$. If the current bit string belongs to A_1, then steps towards bit strings with more zeros are accepted. The situation is similar to the linear function $f(x) = -\sum_{i=1}^{n} x_i$, except for steps to the global optimum, which are, of course, accepted. We conclude that after $O(n \log n)$ steps either the global maximum or a bit string from A_2 is reached. So we now assume that the current bit string belongs to A_2. The situation is similar to OneMax, as long as the number of zeros in the current bit string is at least m. We conclude that again after $O(n \log n)$ steps either the global maximum or a bit string with exactly m zeros is reached. Once the $(1+1)$ EA reaches a bit string with exactly m zeros, only mutations to other bit strings with exactly m zeros and mutations to the global maximum are accepted. The probability for reaching the global maximum by an m-bit mutation equals $n^{-m}(1-1/n)^{n-m}$, so the expected number of steps until this happens is bounded below by n^m and bounded above by $\exp(1)n^m$. We conclude that we have an upper bound of $O(n^m)$ until the optimum is reached.

Now we derive the lower bound. Let T denote the number of steps until the optimum is reached. If q is the probability that some bit string with exactly m zeros is reached, then

$$E(T) \geq q \cdot n^m$$

holds.

Let X_t denote the event that the current bit string x of the $(1+1)$ EA after t steps contains exactly m zeros. Let Y_t denote the event that the $(1+1)$ EA has reached the optimal all one string after at most t steps. Let Z_t denote the event that after t steps the current bit string of the $(1+1)$ EA contains neither exactly m zeros nor no zeros at all. We then have

$$\text{Prob}\left(\overline{X_t}\right) = \text{Prob}\left(Y_t\right) + \text{Prob}\left(Z_t\right).$$

We begin with an upper bound on $\text{Prob}(Y_t) =: y_t$. Obviously, after random initialization we have $y_0 = 2^{-n}$. If w_t denotes the probability for a mutation from a non optimal bit string to the optimum in one step under the probability distribution after $t-1$ steps, we have $y_t = y_{t-1} + (1-y_{t-1})w_t$. Given an upper bound w on w_t this implies

$$y_t \leq 1 - (1-w)^t\,(1-y_0) = 1 - (1-w)^t\left(1-2^{-n}\right) < 1 - (1-w)^t.$$

We have $w_t = \sum_{i=1}^{n} q_{i,t} n^{-i}(1-1/n)^{n-i}$, if $q_{i,t}$ denotes the probability that the current bit string after $t-1$ steps contains exactly i zeros. After random initialization we have $q_{i,1} = 2^{-n}\binom{n}{i}$. In the following steps the probabilities $q_{i,t}$ change. We distinguish the bit strings according to our partition of the search space.

For bit strings that belong to A_1, accepted mutations can only lead to bit strings with at least the same Hamming distance to the optimum or the optimum itself. So the probability for a direct jump to the optimal all one string can only decrease. We see that we are overestimating the probability w_t, if we assume that the probability for direct jumps does not change.

For bit strings that belong to A_2, things are just the other way round. Accepted mutations can only lead to bit strings with at most the same Hamming distance to the optimum. So the probability for a direct jump to the optimal all one string can only increase. We see that we are overestimating the probability w_t, if we assume that the probability for a direct jump to the optimum is given by $n^{-m}(1-1/n)^{n-m}$ for all bit strings. Furthermore, one is a trivial upper bound on the probability to be in one of the bit strings from A_2, i. e., on $\sum_{i=m}^{n} q_{i,t}$, for all t.

So we altogether have

$$\begin{aligned} w_t &\leq \sum_{i=1}^{m-1} 2^{-n}\binom{n}{i} n^{-i}(1-1/n)^{n-i} + n^{-m}(1-1/n)^{n-m} \\ &< \sum_{i=0}^{n} 2^{-n}\binom{n}{i} n^{-i}(1-1/n)^{n-i} + n^{-m}(1-1/n)^{n-m} \\ &= 2^{-n} + n^{-m}(1-1/n)^{n-m}, \end{aligned}$$

and we can use $w := 2^{-n} + n^{-m}(1-1/n)^{n-m}$ as an upper bound for w_t. Using this upper bound we get $y_t \leq 1-(1-2^{-n}-n^{-m}(1-1/n)^{n-m})^t$ as upper bound for $\text{Prob}(Y_t)$. Since the expression $n^{-m}(1-1/n)^{n-m}$ decreases strongly monotone with m, we get an upper bound on w for $m=2$. We set t to $n\log^2 n$ and get

$$\begin{aligned} \text{Prob}\left(Y_{n\log^2 n}\right) = y_{n\log^2 n} &\leq 1-\left(1-2^{-n}-n^{-2}(1-1/n)^{n-2}\right)^{n\log^2 n} \\ &\leq 1-\left(1-2^{-n}-\left(\frac{n}{n-1}\right)^2 \frac{1}{\exp(1)n^2}\right)^{n\log^2 n} \\ &\leq 1-\left(1-\frac{1}{\exp(1)(n-1)^2}\right)^{\left(\exp(1)(n-1)^2-1\right)\cdot\frac{n\log^2 n}{\exp(1)(n-1)^2-1}} \\ &\leq 1-\exp\left(-n\log^2 n/\left(\exp(1)(n-1)^2-1\right)\right). \end{aligned}$$

The last bound converges to 0.

Now we need an upper bound on $\text{Prob}(Z_t)$. As we saw earlier, the expected number of steps before a bit string with exactly m zeros is reached, is bounded above by $cn\log n$ for some constant $c>0$. So using Markoff's inequality it follows that

$$\text{Prob}\left(Z_{n\log^2 n}\right) \leq \frac{cn\log n}{n\log^2 n} = \frac{c}{\log n}.$$

By combining the bounds we get

$$\begin{aligned} E(T) &\geq \text{Prob}\left(X_{n^2}\right)\cdot n^m \\ &\geq \left(\exp\left(-n\log^2 n/\left(\exp(1)(n-1)^2-1\right)\right) - \frac{c}{\log n}\right)\cdot n^m = \Omega\left(n^m\right), \end{aligned}$$

since the lower bound for the probability converges to 1 with increasing values of n. Together with the upper bound this yields $\Theta(n^m)$ for the expected running time of the (1 + 1) EA on JUMP_m with $m>1$. □

6 A variant of the (1 + 1) EA

When we take a closer look at the (1 + 1) EA, i.e. Algorithm 1.1, we see that the old bit string x is replaced by a new bit string x', even if $f(x)=f(x')$. This strategy of *accepting* *equal-valued bit strings* is often used in many Evolutionary Algorithms, as it is assumed to help the algorithm to escape from plateaus, i.e. sets of neighboring bit strings with equal fitness value: if the actual bit string is "surrounded" by bit strings of the same fitness value and Algorithm 1.1 would only accept bit strings with higher fitness, it would need a long time to make an improvement. By accepting bit strings with the same fitness, the $(1+1)$ EA can make random steps on this plateau, which can bring it nearer to bit strings with higher fitness, therefore making it more likely to escape from this plateau.

Now we will show that this common-sense argumentation for the strategy of accepting equal-valued bit strings can be rigorously proven for the PEAK-function to lower the growth of the expected running time of the $(1+1)$ EA substantially. The PEAK-function is defined by

$$\textsc{Peak}(x_1, \ldots, x_n) := \prod_{i=1}^{n} x_i.$$

The PEAK-function should be well suited for our purpose, as it has only one peak, while all other bit strings form one big plateau, therefore giving no "hints" about the peak.

Theorem 6.1: The $(1+1)$ EA has an expected running time of $O(\exp(2n + \ln(n)/2))$ for PEAK. If the $(1+1)$ EA only accepts bit strings with higher fitness, the expected running time for PEAK is $\Theta(\exp(n\ln(n) - n\ln(2)))$.

Proof: Let us first take a look at the expected running time of the original form of the $(1+1)$ EA. In order to upper bound this for PEAK, we lower bound the probability of reaching the global optimum $(1, \ldots, 1)$ after n steps independent of the initial bit string. A sufficient condition to mutate from a bit string with $k < n$ ones to $(1, \ldots, 1)$ is that in every step exactly one of the zeros mutates, but no other bit. This event has probability

$$\prod_{i=k}^{n-1} \binom{n-i}{1} \cdot \frac{1}{n} \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geq \frac{n!}{n^n} \cdot \left(1 - \frac{1}{n}\right)^{n(n-1)} \geq \left(\frac{n}{\exp(1)}\right)^n \cdot \frac{\sqrt{2\pi n}}{\exp(n) \cdot n^n} = \frac{\sqrt{2\pi n}}{\exp(2n)}.$$

So we have to wait at most $\exp(2n)/\sqrt{2\pi n}$ blocks of n steps each in the expected case until the $(1+1)$ EA reaches the global optimum, independent of the initial bit string. Hence, the expected running time of the $(1+1)$ EA for PEAK is at most

$$\sqrt{\frac{n}{2\pi}} \cdot \exp(2n) = O(\exp(2n + \ln(n)/2)).$$

If the $(1+1)$ EA is changed in such a way that it only accepts bit strings with higher fitness the expected running time can be computed exactly. Because now the expected running time of the $(1+1)$ EA is $n^k \cdot (n/(n-1))^{n-k}$, if the initial bit string has k zeros. As the initial bit string is chosen randomly, the expected running time is now

$$\begin{aligned}
&\sum_{k=1}^{n} \binom{n}{k} \cdot 2^{-n} \cdot n^k \cdot \left(\frac{n}{n-1}\right)^{n-k} = 2^{-n} \cdot \left(\left(\sum_{k=0}^{n} \binom{n}{k} \cdot n^k \cdot \left(\frac{n}{n-1}\right)^{n-k}\right) - \left(\frac{n}{n-1}\right)^n\right) \\
= \;& 2^{-n} \cdot \left(\left(n + \frac{n}{n-1}\right)^n - \left(\frac{n}{n-1}\right)^n\right) = \Theta\left(\left(\frac{n}{2}\right)^n\right) = \Theta(\exp(n\ln(n) - n\ln(2))).
\end{aligned}$$

□

So we have proven that the strategy of accepting equal-valued bit strings can improve the order of growth of the expected running time. But do functions exist, where the expected running time of the $(1+1)$ EA increases, when equal-valued bit strings are accepted? It is known that there are functions where for explicit values of n the expected running time is worse when equal-valued bit strings are accepted, but it is an open question, if there are functions so that the order of growth of the expected running time increases, when accepting equal-valued bit strings, too.

7 Conclusion

We have presented several methods to analyze the $(1+1)$ EA. These methods yield results for the classes of linear functions, polynomials of degree 2, and unimodal functions, and they can be used to obtain a hierarchy result. The next step is to analyze Evolutionary Algorithms which allow populations of subjects and crossover.

References

Ackley, D. H. (1987). A Connectionist Machine for Genetic Hillclimbers. Kluwer Academic Publishers, Boston.

Bäck, Th. (1993). Optimal mutation rates in genetic search. In S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms (ICGA). Morgan Kaufman, San Mateo CA, 2–8.

Droste, S., Jansen, Th., and Wegener, I. (1998). A rigorous complexity analysis of the $(1+1)$ Evolutionary Algorithm for linear functions with Boolean inputs. To appear in: Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC'98).

Feller, W. (1971). An Introduction to Probability Theory and Its Applications. Volume II. Wiley, New York.

Fogel, D. B. (1995). Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. IEEE Press, Piscataway, NJ.

Goldberg, D. E. (1989). Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, Mass.

Hagerup, T. and Rüb, C. R. (1989). A guided tour of Chernoff bounds. Information Processing Letters (33), 305–308.

Holland, J. H. (1975). Adaption in Natural and Artificial Systems. University of Michigan, Michigan.

Horn, J., Goldberg, D. E., and Deb, K. (1994). Long path problems. In Y. Davidor, H.-P. Schwefel, and R. Männer (Eds.): Parallel Problem Solving From Nature (PPSN III), 149–158. Springer, Berlin. LNCS 866.

Juels, A. and Wattenberg, M. (1994). Stochastic hillclimbing as a baseline method for evaluating Genetic Algorithms. Technical Report, University of California, Computer Science Department, CSD-04-834.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. Science 220, 671–680.

Koza, J. R. (1992). Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, Mass.

Motwani, R. and Raghavan, P. (1995). Randomized Algorithms. Cambridge University Press, Cambridge.

Mühlenbein, H. (1992). How Genetic Algorithms really work. Mutation and hill-climbing. In R. Männer and R. Manderick (Eds.), Parallel Problem Solving from Nature (PPSN II), 15–25. North-Holland, Amsterdam.

Rechenberg, I. (1994). Evolutionsstrategie '94. Frommann-Holzboog, Stuttgart.

Rudolph, G. (1997). Convergence Properties of Evolutionary Algorithms. Ph. D. Thesis. Verlag Dr. Kovač, Hamburg.

Rudolph, G. (1997b). How mutation and selection solve long-path problems in polynomial expected time. Evolutionary Computation 4(2), 195–205.

Schwefel, H.-P. (1995). Evolution and Optimum Seeking. Wiley, New York.