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

Dynamic Parameter Control in Simple Evolutionary Algorithms

Stefan Droste   Thomas Jansen   Ingo Wegener

No. CI-89/00

Technical Report   ISSN 1433-3325   August 2000

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.

Dynamic Parameter Control in Simple Evolutionary Algorithms

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

Abstract

Evolutionary algorithms are general, randomized search heuristics that are influenced by many parameters. Though evolutionary algorithms are assumed to be robust, it is well-known that choosing the parameters appropriately is crucial for success and efficiency of the search. It has been shown in many experiments, that non-static parameter settings can be by far superior to static ones but theoretical verifications are hard to find. We investigate a very simple evolutionary algorithm and rigorously prove that employing dynamic parameter control can greatly speed-up optimization.

1 INTRODUCTION

Evolutionary algorithms are a class of general, randomized search heuristics that can be applied to many different tasks. They are controlled by a number of different parameters which are crucial for success and efficiency of the search. Though rough guidelines mainly based on empirical experience exist, it remains a difficult task to find appropriate settings. One way to overcome this problem is to employ non-static parameter control. Bäck (Bäck 1998) distinguishes three different ways of non-static parameter control: dynamic parameter control is the simplest variant. The parameters are set according to some (maybe randomized) scheme, that depends on the number of generations. In adaptive parameter control the control scheme can take the individuals and their function values encountered so far also into account. Finally, when self-adaptive parameter control is used, the parameters are evolved by application of the same search operators as used by evolutionary algorithms, namely mutation, crossover, and selection. All three variants are used in practice, but there is little theoretically confirmed knowledge about them. This holds especially as far as optimization of discrete objective functions is concerned. In the field of evolution strategies (Schwefel 1995) on continuous domains some theoretical studies are known (Beyer 1996; Rudolph 1999).

Here, we concentrate on the exact maximization of fitness functions $f\colon \{0,1\}^n \to \mathbb{R}$ by means of a very simple evolutionary algorithm. In its basic form it uses static parameter control, of course, and is well-known as (1+1) EA ((1+1) evolutionary algorithm) (Mühlenbein 1992; Rudolph 1997; Droste, Jansen, and Wegener 1998b; Garnier, Kallel, and Schoenauer 1999). In Section 2 we introduce the (1+1) EA. In Section 3 we consider a modified selection scheme that is parameterized and subject to dynamic parameter control. We employ a simplified mutation operator leading to the Metropolis algorithm (Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller 1953) in the static and to simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983) in the dynamic case. For an appropriate fitness function serving as an example we prove that appropriate dynamic parameter control schemes can reduce the average time needed for optimization from exponential to polynomial compared to an optimal static setting. In Section 4 we employ a very simple dynamic parameter control of the mutation probability and show how this enhances the robustness of the algorithm: in cases where already a static setting is efficient it typically slows down the optimization only by the factor $\log n$. Furthermore, we prove for an appropriately chosen fitness function f that it efficiently optimizes f which cannot be achieved using the most recommended static choice for the mutation probability. On the other hand, we present a function where this special dynamic variant of the (1+1) EA is by far outperformed by its static counterpart. In Section 5 we finish with some concluding remarks.

2 THE (1+1) EA

Theoretical results about evolutionary algorithms are in general difficult to obtain. This is mainly due to their stochastic character. Especially, crossover leads to the analysis of quadratical dynamic systems, which is of extreme difficulty (Rabani, Rabinovich, and Sinclair 1998). Therefore, it is a common approach to consider simplified evolutionary algorithms, which (hopefully) still contain interesting, typical, and important features of evolutionary algorithms in general. The maybe simplest and best known such algorithm is the so-called (1+1) evolutionary algorithm ((1+1) EA). It has been subject to intense research, Mühlenbein (1992), Rudolph (1997), Droste, Jansen, and Wegener (1998b), and Garnier, Kallel, and Schoenauer (1999) are just a few examples. It can be formally defined as follows, where $f\colon \{0,1\}^n \to \mathbb{R}$ is the objective function to be maximized:

Algorithm 1 ((1+1) EA).
1. *Choose* $p(n) \in (0; 1/2]$.
2. *Choose* $x \in \{0,1\}^n$ *uniformly at random.*
3. *Create* y *by flipping each bit in* x *independently with probability* $p(n)$.
4. *If* $f(y) \geq f(x)$, *set* $x := y$.
5. *Continue at line 3.*

The probability $p(n)$ is called the mutation probability. The usual and recommended static choice is $p(n) = 1/n$ (Bäck 1993), which implies that on average one bit is flipped in each generation. All the studies mentioned above investigate the case $p(n) = 1/n$. In the next section we modify the selection in line 4 such that with some probability strings y with

$f(y) < f(x)$ are accepted, too. In Section 4 we modify the (1+1) EA by changing the mutation probability $p(n)$ in each step.

3 DYNAMIC PARAMETER CONTROL IN SELECTION

In this section we compare a variant of the (1+1) EA which uses a simplified mutation operator and a probabilistic selection mechanism. Mutation consists of flipping exactly one randomly chosen bit. While this makes an analysis much easier, the selection is now more complicated: if the new search point is y and the old one x, the new point y is selected with probability $\min(1, \alpha^{f(y)-f(x)})$, where the *selection parameter* α is an element of $[1, \infty[$. So worsenings are now accepted with some probability, which decreases for large worsenings, while improvements are always accepted.

The only parameter for which we consider static and non-static settings is the selection parameter α. To avoid any misunderstandings we present the algorithm more formally now.

Algorithm 2.
1. *Set* $t := 1$. *Choose* $x \in \{0,1\}^n$ *uniformly at random.*
2. *Create* y *by flipping one randomly (under the uniform distribution) chosen bit of* x.
3. *With probability* $\min\{1, \alpha(t)^{f(y)-f(x)}\}$ *set* $x := y$.
4. *Set* $t := t + 1$. *Continue at line 2.*

The function $\alpha \colon \mathbb{N} \to [1; \infty[$ is usually denoted as *selection schedule*. If $\alpha(t)$ is constant with respect to t the algorithm is called static, otherwise dynamic. We compare static variants of this algorithm with dynamic ones with respect to the expected running time, i. e., the expected number of steps the algorithms make until $f(x)$ is the maximum of f for the first time.

We note that choosing a fixed value for α yields the Metropolis algorithm (see Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller (1953)), while otherwise we get a simulated annealing algorithm, where the neighborhood of a search point consists of all points with Hamming distance one. Hence, our approach can also be seen as a step to answer the question raised by Jerrum and Sinclair (1997): Is there a natural cooling schedule (which corresponds to our selection schedule), such that simulated annealing outperforms the Metropolis algorithm for a natural problem? There are various attempts to answer this question (see Jerrum and Sorkin (1998) and Sorkin (1991)). In particular, Sorkin (1991) proves that simulated annealing is superior to the Metropolis algorithm on a carefully designed fractal function. He proves his results using the method of rapidly mixing Markov chains (see Sinclair (1993) for an introduction). Note, that our proof has a much simpler structure and is easier to understand. Furthermore, we derive our results using quite elementary methods. Namely, our proofs do mainly use Markov bounds.

In the following we will present some equations for the expected number of steps the static algorithm needs to find a maximum. If we can bound the value of $\alpha(t)$, these equations will be also helpful to bound the expected number of steps in the dynamic case. We assume that our objective functions are symmetric and have their only global maximum at the all ones bit string $(1, \ldots, 1)$. A symmetric function $f \colon \{0,1\}^n \to \mathbb{R}$ depends on the number of ones in the input only.

So, when trying to maximize a symmetric function, the expected number of steps the algorithm needs to reach the maximum depends only on the number of ones the actual bit string x contains, but not on their positions. Therefore, we can model the process by a Markov chain with exactly $n+1$ states. Let the random variable T_i (for $i \in \{0, \dots, n\}$) be the random number of steps Algorithm 2 with constant α needs to reach the maximum for the first time, when starting in a bit string with i ones. As the initial bit string is chosen randomly with equal probability, the expected value of the number T of steps, the whole algorithm needs, is

$$\mathrm{E}\left(T\right) = \sum_{i=0}^{n} \frac{\binom{n}{i}}{2^n} \cdot \mathrm{E}\left(T_i\right).$$

Hence, by bounding $\mathrm{E}\left(T_i\right)$ for all $i \in \{0, \dots, n\}$ we can bound $\mathrm{E}\left(T\right)$. As the algorithm can only change the number of ones in its actual bit string by one, the number T_i of steps to reach the maximum $(1, \dots, 1)$ is the sum of the numbers T_j^+ of steps to reach $j+1$ ones, when starting with j ones, over all $j \in \{i, \dots, n-1\}$.

Let p_i^+ resp. p_i^- be the transition probability, that the algorithm goes to a state with $i+1$ resp. $i-1$ ones when being in a state with $i \in \{0, \dots, n\}$ ones. Then the following lemma is an immediate consequence.

Lemma 3. *The expected number $\mathrm{E}\left(T_i^+\right)$ of steps to reach a state with $i+1$ ones for the first time, when starting in a state with $i \in \{1, \dots, n-1\}$ ones, is*

$$\mathrm{E}\left(T_i^+\right) = \frac{1}{p_i^+} + \frac{p_i^-}{p_i^+} \cdot \mathrm{E}\left(T_{i-1}^+\right).$$

Proof. When being in a state with $i \in \{1, \dots, n-1\}$ ones, the number of ones can increase, decrease or stay the same. This leads to the following equation:

$$\begin{aligned} & E(T_i^+) && = p_i^+ \cdot 1 + p_i^- \cdot \left(1 + E(T_{i-1}^+) + E(T_i^+)\right) + \left(1 - p_i^+ - p_i^-\right) \cdot \left(1 + E(T_i^+)\right) \\ \Leftrightarrow \quad & p_i^+ \cdot E(T_i^+) && = 1 + p_i^- \cdot E(T_{i-1}^+) \\ \Leftrightarrow \quad & E(T_i^+) && = \frac{1}{p_i^+} + \frac{p_i^-}{p_i^+} \cdot E(T_{i-1}^+). \end{aligned}$$

$\square$

Using this recursive equation to determine $\mathrm{E}\left(T_i^+\right)$, we can derive the following lemma by induction:

Lemma 4. *The expected number $\mathrm{E}\left(T_i^+\right)$ of steps to reach a state with $i+1$ ones for the first time, when starting in a state with $i \in \{1, \dots, n-1\}$ ones, is for all $j \in \{1, \dots, i\}$:*

$$\mathrm{E}\left(T_i^+\right) = \left(\sum_{k=0}^{j-1} \frac{\prod_{l=0}^{k-1} p_{i-l}^-}{\prod_{l=0}^{k} p_{i-l}^+}\right) + \frac{\prod_{l=0}^{j-1} p_{i-l}^-}{\prod_{l=0}^{j-1} p_{i-l}^+} \cdot \mathrm{E}\left(T_{i-j}^+\right).$$

Proof. The equation can be proven by induction over j. For $j = 1$ it is just Lemma 3.

Assuming that it is valid for j, we can prove it for $j+1$ in the following way:

$$\begin{aligned}
E(T_i^+) &= \left(\sum_{k=0}^{j-1} \frac{\prod_{l=0}^{k-1} p_{i-l}^-}{\prod_{l=0}^{k} p_{i-l}^+}\right) + \frac{\prod_{l=0}^{j-1} p_{i-l}^-}{\prod_{l=0}^{j-1} p_{i-l}^+} \cdot E(T_{i-j}^+) \\
&= \left(\sum_{k=0}^{j-1} \frac{\prod_{l=0}^{k-1} p_{i-l}^-}{\prod_{l=0}^{k} p_{i-l}^+}\right) + \frac{\prod_{l=0}^{j-1} p_{i-l}^-}{\prod_{l=0}^{j-1} p_{i-l}^+} \cdot \left(\frac{1}{p_{i-j}^+} + \frac{p_{i-j}^-}{p_{i-j}^+} \cdot E(T_{i-j-1}^+)\right) \\
&= \left(\sum_{k=0}^{j-1} \frac{\prod_{l=0}^{k-1} p_{i-l}^-}{\prod_{l=0}^{k} p_{i-l}^+}\right) + \frac{\prod_{l=0}^{j-1} p_{i-l}^-}{\prod_{l=0}^{j} p_{i-l}^+} + \frac{\prod_{l=0}^{j} p_{i-l}^-}{\prod_{l=0}^{j} p_{i-l}^+} \cdot E(T_{i-j-1}^+) \\
&= \left(\sum_{k=0}^{j} \frac{\prod_{l=0}^{k-1} p_{i-l}^-}{\prod_{l=0}^{k} p_{i-l}^+}\right) + \frac{\prod_{l=0}^{j} p_{i-l}^-}{\prod_{l=0}^{j} p_{i-l}^+} \cdot E(T_{i-(j+1)}^+).
\end{aligned}$$

□

Since $\mathrm{E}\left(T_0^+\right) = 1/p_0^+$, we get for the case $j = i$:

Corollary 5. *The expected number* $\mathrm{E}\left(T_i^+\right)$ *of steps to reach a state with* $i+1$ *ones, when starting in a state with* $i \in \{1, \ldots, n-1\}$ *ones, is:*

$$\mathrm{E}\left(T_i^+\right) = \left(\sum_{k=0}^{i-1} \frac{\prod_{l=0}^{k-1} p_{i-l}^-}{\prod_{l=0}^{k} p_{i-l}^+}\right) + \frac{\prod_{l=0}^{i-1} p_{i-l}^-}{\prod_{l=0}^{i-1} p_{i-l}^+} \cdot \frac{1}{p_0^+} = \sum_{k=0}^{i} \frac{\prod_{l=0}^{k-1} p_{i-l}^-}{\prod_{l=0}^{k} p_{i-l}^+} = \sum_{k=0}^{i} \frac{1}{p_k^+} \cdot \prod_{l=k+1}^{i} \frac{p_l^-}{p_l^+}.$$

Using these results we now show that there exists a function $\text{VALLEY}\colon \{0,1\}^n \to \mathbb{R}$, such that Algorithm 2 using an appropriate selection schedule with decreasing probability for accepting worsenings needs only polynomial expected time, while setting α constant implies exponential expected time, independent of the choice of α. We do this by showing that the running time with a special increasing selection schedule is polynomial with very high probability, so that all the remaining cases have only exponentially small probability and cannot influence the result by more than a constant.

Intuitively, the function VALLEY should have the following properties: With a probability that is bounded below by a positive constant we start with strings where it is necessary to accept worsenings. In the late steps of maximization the acceptance of worsenings increases the maximization time. We will show that the following function fulfills these intuitive concepts to a sufficient extent.

Definition 6. *The function* $\text{VALLEY} : \{0,1\}^n \to \mathbb{R}$ *is defined by (w. l. o. g.* n *is even):*

$$\text{VALLEY} := \begin{cases} n/2 - \|x\|_1 & \text{for } \|x\|_1 \le n/2, \\ 7n^2 \ln(n) - n/2 + \|x\|_1 & \text{for } \|x\|_1 > n/2, \end{cases}$$

where $\|x\|_1$ *denotes the number of ones in* x.

Theorem 7. *The expected number of steps until Algorithm 2 with constant* $\alpha(t) = \alpha$ *reaches the maximum of* VALLEY *for the first time is*

$$\Omega\left(\left(\sqrt{\frac{\alpha}{4}}\right)^n + \left(\frac{1}{\alpha} + 1\right)^n\right) = \Omega\left(1.179^n\right)$$

for all choices of $\alpha \in [1, \infty[$.

Proof. The idea of the proof is that for large α, i. e., small probability of accepting worsenings, the expected time to come from state $n/2-1$ to state $n/2$ is exponential, while for small α the expected time to come from state $n-1$ to state n is exponential.

When we take a look at the function VALLEY for all x with $||x||_1 < n/2$, we see, that it behaves like $-$ONEMAX with respect to Algorithm 2 with a static choice of α. So, if p_j^+ resp. p_j^- is the probability of increasing the number of ones resp. decreasing the number of ones by one, when the actual x contains exactly j ones, we have for all $j \in \{0, \ldots, n/2-1\}$

$$p_j^+ = \frac{n-j}{\alpha \cdot n} \text{ and } p_j^- = \frac{j}{n}.$$

Hence, using Corollary 5, we get for $E(T_i^+)$, the expected number of steps until we reach a bit string with $i+1$ ones, when starting with a bit string with $i < n/2$ ones:

$$\begin{aligned} E(T_i^+) &= \sum_{k=0}^{i} \frac{1}{p_k^+} \cdot \prod_{l=k+1}^{i} \frac{p_l^-}{p_l^+} = \sum_{k=0}^{i} \frac{\alpha \cdot n}{n-k} \cdot \prod_{l=k+1}^{i} \frac{l}{n} \cdot \frac{\alpha \cdot n}{n-l} \\ &= \sum_{k=0}^{i} \alpha^{i-k+1} \cdot \frac{n}{n-k} \cdot \frac{i! \cdot (n-i-1)!}{k! \cdot (n-k-1)!} = \sum_{k=0}^{i} \alpha^{i-k+1} \cdot \frac{\binom{n}{k}}{\binom{n-1}{i}}. \end{aligned} \tag{1}$$

So $E(T_{n/2-1}^+)$ can be lower bounded in the following way

$$E(T_{n/2-1}^+) = \sum_{k=0}^{n/2-1} \alpha^{n/2-k} \cdot \frac{\binom{n}{k}}{\binom{n-1}{n/2-1}} \geq \frac{\alpha^{n/2}}{\binom{n-1}{n/2-1}} \geq \frac{\alpha^{n/2}}{2^n}. \tag{2}$$

Hence, $E(T_{n/2-1}^+)$ is $\Omega\left(\left(\sqrt{\alpha/4}\right)^n\right)$. So for $\alpha \geq 4+\varepsilon$ (where $\varepsilon > 0$) this results in an exponential lower bound for $E(T_{n/2-1}^+)$, implying this bound for $E(T_i)$ for all $i \in \{0, \ldots, n/2-1\}$. Because this is at least a constant fraction of all bit strings, we have an exponential lower bound for the expected number of steps for any static choice of α with $\alpha \geq 4+\varepsilon$.

In the following we want to show an exponential lower bound for the expected number of steps $E(T_{n-1})$ for $\alpha < 4+\varepsilon$. When α is small, worsenings are accepted with large probability, so that we can expect $E(T_{n-1})$ to be large. To lower bound $E(T_{n-1})$ we use Lemma 4. Because we have for all $i \in \{n/2+2, \ldots, n-1\}$:

$$p_i^+ = \frac{n-i}{n} \text{ and } p_i^- = \frac{i}{n \cdot \alpha},$$

we can lower bound $E(T_{n-1}^+)$ by:

$$\begin{aligned} E(T_{n-1}^+) &\geq \sum_{k=0}^{n/2-3} \frac{\prod_{l=0}^{k-1} p_{n-1-l}^-}{\prod_{l=0}^{k} p_{n-1-l}^+} = \sum_{k=0}^{n/2-3} \frac{\prod_{l=n-k}^{n-1} p_l^-}{\prod_{l=n-k-1}^{n-1} p_l^+} = \sum_{k=0}^{n/2-3} \frac{\prod_{l=n-k}^{n-1} \frac{l}{n \cdot \alpha}}{\prod_{l=n-k-1}^{n-1} \frac{n-l}{n}} \\ &= \sum_{k=0}^{n/2-3} \frac{n}{\alpha^k} \cdot \frac{(n-1)!}{(n-k-1)! \cdot (k+1)!} = \sum_{k=0}^{n/2-3} \frac{\binom{n}{k+1}}{\alpha^k} = \alpha \cdot \sum_{k=1}^{n/2-2} \frac{\binom{n}{k}}{\alpha^k} \\ &\geq \alpha \cdot \left(\frac{\left(\frac{1}{\alpha}+1\right)^{n-4}}{2} - 1 \right) = \Omega\left(\left(\frac{1}{\alpha} + 1 \right)^{n-4} \right). \end{aligned}$$

Hence, for all $i \in \{0, \ldots, n/2-1\}$ the expected value of T_i is

$$E(T_i) = \Omega\left(\left(\sqrt{\frac{\alpha}{4}}\right)^n + \left(\frac{1}{\alpha}+1\right)^{n-4}\right),$$

which is exponential for all choices of $\alpha \in [1;\infty[$. As the fraction of bit strings with at most $n/2-1$ ones is bounded below by a positive constant, the expected running time is exponential for all α. Numerical analysis leads to the result that this is $\Omega(1.179^n)$. □

Intuitively, one can perform better on VALLEY, if the selection schedule works as follows: in the beginning, worsenings are accepted with probability almost one, so that the actual point x is almost making a random walk, until its number of ones increases to $n/2+1$. As the difference between the function values for $n/2+1$ and $n/2$ ones is so large, it is very unlikely that the number of ones of the actual x will fall below $n/2+1$, assuming $\alpha(t) > 1$ at this point of time. Hence, if the probability of accepting worsenings decreases after some carefully chosen number of steps, the maximum $(1, \ldots, 1)$ should be reached quickly:

Theorem 8. *With probability $1 - O(n^{-n})$ the number of steps until Algorithm 2 with the selection schedule*

$$\alpha(t) := 1 + \frac{t}{s(n)}$$

reaches the maximum of VALLEY *for the first time is $O(n \cdot s(n))$ for any polynomial s with $s(n) \geq 2en^4 \log n$. Furthermore, the expected number of steps until this happens is $O(n \cdot s(n))$, if we set $\alpha(t) := 1$ for $t > 2^n$.*

Proof. The basic idea of the proof is to split the run of Algorithm 2 into two phases of predefined length. We show that with very high probability a state with at least $n/2+1$ ones is reached within the first phase, and all succeeding states have at least $n/2+1$ ones, too. Furthermore, with very high probability the optimum is reached within the second phase. Finally, we upper bound the expected number of steps in the case, that any of these events do not happen.

The first phase has length $s(n)/n + 2en^3 \log n$. We want to upper bound the expected number of steps in the first phase Algorithm 2 takes to reach a state with at least $n/2+1$ ones. For that purpose we upper bound $\mathrm{E}\left(T_i^+\right)$ for all $i \in \{0, \ldots, n/2\}$. We do not care what happens during the first $s(n)/n$ steps. After that, we have $\alpha(t) \geq 1 + 1/n$. Pessimistically we assume that the current state at step $t = s(n)/n$ contains at most $n/2$ ones.

We use equation (1) of Theorem 7, which is valid for $i \in \{0, \ldots, n/2-1\}$.

$$\begin{aligned}
\mathrm{E}\left(T_i^+\right) &= \sum_{j=0}^{i} \alpha^{i-j+1} \cdot \frac{\binom{n}{j}}{\binom{n-1}{i}} = \sum_{j=0}^{i} \alpha^{j+1} \cdot \frac{\binom{n}{i-j}}{\binom{n-1}{i}} \\
&= \sum_{j=0}^{i} \alpha^{j+1} \cdot \frac{n!}{(i-j)! \cdot (n-i+j)!} \cdot \frac{i! \cdot (n-1-i)!}{(n-1)!} = \sum_{j=0}^{i} \alpha^{j+1} \cdot \frac{\binom{i}{j}}{\binom{n-i+j}{j}} \cdot \frac{n}{n-i}
\end{aligned}$$

As the last expression decreases with decreasing i, it follows that $\mathrm{E}\left(T_i^+\right) \leq \mathrm{E}\left(T_{i+1}^+\right)$ for all $i \in \{0, \ldots, n/2-1\}$. Since the length of the first phase is $s(n)/n + 2en^3 \log n$, we have

$\alpha(t) \leq 1 + 2/n$ during the first phase. Using this and setting $i = n/2 - 1$, we get

$$\mathrm{E}\left(T^+_{n/2-1}\right) \leq \sum_{j=0}^{n/2-1} \left(1 + \frac{2}{n}\right)^{j+1} \frac{\binom{n/2-1}{j}}{\binom{n/2+1+j}{j}} \cdot \frac{n}{n/2+1} \leq 2 \sum_{j=0}^{n/2-1} e = en.$$

Hence, by using Lemma 3 we can upper bound $\mathrm{E}\left(T^+_{n/2}\right)$ by

$$\mathrm{E}\left(T^+_{n/2}\right) = \frac{1}{(n/2)/n} + \frac{(n/2)/n}{(n/2)/n} \cdot \mathrm{E}\left(T^+_{n/2-1}\right) \leq 2 + en.$$

So, the expected number of steps until a bit string with more than $n/2$ ones is reached is bounded above by

$$(n/2) \cdot en + 2 + en \leq en^2.$$

We use the Markov inequality and see, that the probability of not reaching a state with more than $n/2$ ones within $2en^2$ steps is at most $1/2$. Our analysis is independent of the current bit string at the beginning of such a subphase of length $2en^2$. So, we can consider the $2en^3 \log n$ steps in the first phase as $n \log n$ independent subphases of length $2en^2$ each. Hence, the probability of not reaching a state with more than $n/2$ ones within the first phase is $O(n^{-n})$.

Assume that Algorithm 2 reaches a bit string with more than $n/2$ ones at some step t with $t \geq s(n)/n$. This yields $\alpha(t) \geq 1 + 1/n$. Let $p(n)$ be some polynomial. The probability to reach a bit string with at most $n/2$ ones within $p(n)$ steps is bounded above by

$$p(n) \cdot \frac{n/2+1}{n \cdot (1+1/n)^{7n^2 \ln n}} < \frac{p(n)}{e^{4n \ln n}} = O(n^{-n})$$

where the last equality follows since $p(n)$ is a polynomial. We conclude that after once reaching a bit string with more than $n/2$ ones, for polynomially bounded number of steps the number of ones is larger than $n/2$, too, with probability $1 - O(n^{-n})$. Hence, after the first phase the probability of not being in a state with more than $n/2$ ones is $O(n^{-n})$.

Now we consider the succeeding second phase, which ends with $t = n \cdot s(n)$, which is polynomially bounded. Therefore, we neglect the case that during the second phase a bit string with at most $n/2$ ones is reached. We saw above, that this case has probability $O(n^{-n})$.

We want to prove that with very high probability the optimum is reached within the second phase. In order to do so we upper bound the expected number of steps Algorithm 2 needs to reach the optimum. We do not care about the beginning of phase 2 and consider only steps with $t \geq (n-1)s(n)$. Then we have $\alpha(t) \geq n$. Due to the length of the second phase, we have $\alpha(t) \leq n + 1$, too. Using equation (2) of Theorem 7, we can upper bound $\mathrm{E}\left(T^+_{n/2-1}\right)$ in the following way.

$$\mathrm{E}\left(T^+_{n/2-1}\right) \leq \sum_{j=0}^{n/2-1} (n+1)^{n/2-j} \cdot \frac{\binom{n}{j}}{\binom{n-1}{n/2-1}} \leq \sum_{j=0}^{n/2-1} \binom{n}{j} n^{n-j} \leq (1+n)^n$$

Hence, we can upper bound $\mathrm{E}\left(T^+_{n/2}\right)$ by

$$\mathrm{E}\left(T^+_{n/2}\right) = \frac{1}{(n/2)/n} + \frac{(n/2)/n}{(n/2)/n} \cdot \mathrm{E}\left(T^+_{n/2-1}\right) \leq 2 + (1+n)^n$$

and $\mathrm{E}\left(T^+_{n/2+1}\right)$ by

$$\begin{aligned}\mathrm{E}\left(T^+_{n/2+1}\right) &\leq \frac{1}{(n/2-1)/n} + \frac{(n/2+1)/(n\cdot n^{7n^2\ln n})}{(n/2-1)/n}\cdot \mathrm{E}\left(T^+_{n/2}\right)\\ &\leq \frac{2n}{n-2} + \frac{n+2}{2n^{7n^2\ln(n)+1}}\cdot\frac{2n}{n-2}\left((1+n)^n+2\right)\leq 7.\end{aligned}$$

Using Lemma 4 for $j = i - n/2 - 1$, we get for all $i \in \{n/2+2,\dots,n-1\}$

$$\begin{aligned}\mathrm{E}\left(T^+_i\right) &= \left(\sum_{j=0}^{i-n/2-2}\frac{\prod_{k=0}^{j-1}p^-_{i-k}}{\prod_{k=0}^{j}p^+_{i-k}}\right)+\frac{\prod_{k=0}^{i-n/2-2}p^-_{i-k}}{\prod_{k=0}^{i-n/2-2}p^+_{i-k}}\cdot \mathrm{E}\left(T^+_{n/2+1}\right)\\ &\leq \left(\sum_{j=0}^{i-n/2-2}\frac{\prod_{k=i-j+1}^{i}p^-_{k}}{\prod_{k=i-j}^{i}p^+_{k}}\right)+\frac{\prod_{k=n/2+2}^{i}p^-_{k}}{\prod_{k=n/2+2}^{i}p^+_{k}}\cdot 7.\end{aligned}$$

As Valley behaves like OneMax for all states with at least $n/2+2$ ones with respect to Algorithm 2, we have $p^+_k = (n-k)/n$ and $p^-_k = k/(n\alpha/t)$. Hence, we get

$$\begin{aligned}\mathrm{E}\left(T^+_i\right) &\leq \left(\sum_{j=0}^{i-n/2-2}\frac{\prod_{k=i-j+1}^{i}k/(n\cdot n)}{\prod_{k=i-j}^{i}(n-k)/n}\right)+\frac{\prod_{k=n/2+2}^{i}k/(n\cdot n)}{\prod_{k=n/2+2}^{i}(n-k)/n}\cdot 7\\ &= \left(\sum_{j=0}^{i-n/2-2}\frac{n^{-2j}\cdot i!/(i-j)!}{n^{-j-1}\cdot(n-i+j)!/(n-i-1)!}\right)+\\ &\qquad \frac{n^{-2i+n+2}\cdot i!/(n/2+1)!}{n^{-i+n/2+1}\cdot(n/2-2)!/(n-i-1)!}\cdot 7\\ &= \left(\sum_{j=0}^{i-n/2-2}n^{1-j}\cdot\frac{\binom{i}{j}}{(n-i)\cdot\binom{n-i+j}{j}}\right)+\frac{(n/2-1)\cdot\binom{n}{n/2+1}\cdot n^{n/2+1}}{(n-i)\cdot\binom{n}{i}\cdot n^i}\cdot 7.\end{aligned}$$

To upper bound the second term, we derive the following for all $i \in \{0,\dots,n-2\}$.

$$\begin{aligned}&(n-i)\cdot\binom{n}{i}\cdot n^i \leq (n-(i+1))\cdot\binom{n}{i+1}\cdot n^{i+1}\\ \iff\quad & \frac{n-i}{n-i-1}\cdot\frac{n!}{i!\cdot(n-i)!}\cdot\frac{(i+1)!\cdot(n-i-1)!}{n!}\leq n\\ \iff\quad & \frac{i+1}{n-i-1}\leq n, \text{ which is valid for all } i\in\{0,\dots,n-2\}.\end{aligned}$$

Hence, we get the following upper bound for $\mathrm{E}\left(T^+_i\right)$, as i is at least $n/2+2$:

$$\mathrm{E}\left(T^+_i\right)\leq\left(\sum_{j=0}^{i-n/2-2}n^{1-j}\cdot\frac{\binom{i}{j}}{(n-i)\cdot\binom{n-i+j}{j}}\right)+7.$$

So, by upper bounding $\mathrm{E}\left(T^+_{n-1}\right)$, we get an upper bound for $\mathrm{E}\left(T^+_i\right)$ for all $i \in \{n/2+2,\dots,n-1\}$:

$$\begin{aligned}\mathrm{E}\left(T^+_{n-1}\right) &\leq n\cdot\left(\sum_{j=0}^{n/2-3}\left(\frac{1}{n}\right)^j\cdot\frac{\binom{n-1}{j}}{\binom{j+1}{j}}\right)+7\leq n\cdot\left(\sum_{j=0}^{n/2-3}\left(\frac{1}{n}\right)^j\binom{n}{j}\right)+7\\ &\leq n\cdot(1+1/n)^n+7\leq en+7\leq 2en\end{aligned}$$

Hence, for all $i \in \{n/2+1, \dots, n\}$ the value of $\mathrm{E}(T_i)$ can be upper bounded by en^2. Using the Markov inequality, this implies that after $2en^2$ steps the probability that the optimum is not reached is upper bounded by 1/2. Considering the $s(n) \geq 2en^4 \log n$ steps as at least $n^2 \log n$ independent subphases of length $2en^2$ each, implies that the optimum is reached with probability $1 - O(n^{-n})$. Altogether we proved that the optimum is reached within the first $n \cdot s(n)$ steps with probability $1 - O(n^{-n})$.

In order to derive the upper bound on the expected number of steps we consider the case that the optimum is not reached. This has probability $O(n^{-n})$. We use the additional assumption that $\alpha(t) = 1$ holds for $t > 2^n$. We do not care what else happens until $t > 2^n$ holds. Then we have $\alpha(t) = 1$. This implies that the algorithm performs a pure random walks, so the expected number of steps in this case is upper bounded by $O(2^n)$ (Garnier, Kallel, and Schoenauer 1999). This yields that the contribution in case of a failure to the expected number of steps is

$$O(2^n) \cdot O(n^{-n}) = O(1)$$

to the expected running time. Altogether, we see that the expected running time is upper bounded by $O(n \cdot s(n))$. □

4 DYNAMIC PARAMETER CONTROL IN MUTATION

In this section we present a variant of the (1+1) EA that uses a very simple dynamic variation scheme for the mutation probability $p(n)$. The key idea is to try all possible mutation probabilities. Since we do not want to have too many steps where no bit flips at all, we consider $1/n$ to be a reasonable lower bound: using $p(n) = 1/n$ implies that on average one bit is flipped in one mutation. As for the (1+1) EA we use 1/2 as an upper bound for the choice of $p(n)$. Furthermore, we do not want to try too many different mutation probabilities, since each try is a potential waste of time. Therefore, we double the mutation probability in each step, which yields a range of $\lfloor \log n \rfloor$ different mutation probabilities.

Algorithm 9.
1. Choose $x \in \{0,1\}^n$ uniformly at random.
2. $p(n) := 1/n$.
3. Create y by flipping each bit in x independently with probability $p(n)$.
4. If $f(y) \geq f(x)$, set $x := y$.
5. $p(n) := 2p(n)$. If $p(n) > 1/2$, set $p(n) := 1/n$.
6. Continue at line 3.

First of all, we demonstrate that the dynamic version has a much better worst case performance than the (1+1) EA with fixed mutation probability $p(n) = 1/n$. It is known (Droste, Jansen, and Wegener 1998a) that for some functions the (1+1) EA with $p(n) = 1/n$ needs $\Theta(n^n)$ steps for optimization.

Theorem 10. *For any function $f\colon \{0,1\}^n \to \mathbb{R}$ the expected number of steps Algorithm 9 needs to optimize f is upper bounded by $4^n \log n$.*

Proof. Algorithm 9 uses $\lfloor \log n \rfloor$ different values for the mutation probability $p(n)$, all from the interval $[1/n; 1/2]$. In particular, for each $d \in [1/n; 1/4]$ we have that some mutation probability $p(n) \in [d; 2d]$ is used every $\lfloor \log n \rfloor$-th step. Using $d = 1/4$ yields that in each $\lfloor \log n \rfloor$-th step we have $p(n) \geq 1/4$. In these steps, the probability to create a global maximum as child y in mutation is lower bounded by $(1/4)^n$. Thus, each $\lfloor \log n \rfloor$-th step with probability at least 4^{-n} a global maximum is reached. Therefore, the expected number of steps needed for optimization is upper bounded by $4^n \log n$. $\square$

Note, that depending on the value of n better upper bounds are possible. If n is a power of 2, $p(n) = 1/2$ is one of the values used and we have $2^n \log n$ as an upper bound. This is a general property of Algorithm 9: depending on the value of n different values for $p(n)$ are used which can yield different expected running times.

Of course, using the (1+1) EA with the static choice $p(n) = 1/2$ achieves an expected running time $O(2^n)$ for all functions. But, for each function with a unique global optimum the expected running time equals 2^n. For Algorithm 9 such dramatic running times on simple functions are usually not the case. We consider examples, namely the functions OneMax and LeadingOnes and the class of all linear functions.

Definition 11. *The function* OneMax: $\{0,1\}^n \to \mathbb{R}$ *is defined by* $\text{OneMax}(x) := \|x\|_1$ *for all* $x \in \{0,1\}^n$. *The function* LeadingOnes: $\{0,1\}^n \to \mathbb{R}$ *is defined by*

$$\text{LeadingOnes}(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$$

for all $x \in \{0,1\}^n$.

The expected running time of the (1+1) EA with $p(n) = 1/n$ is $\Theta(n \log n)$ for OneMax and $\Theta(n^2)$ for LeadingOnes (Droste, Jansen, and Wegener 1998a).

Theorem 12. *The expected running time of Algorithm 9 on the function* LeadingOnes *is* $\Theta(n^2 \log n)$. *Furthermore, there are two constants* $0 < c_1 < c_2$ *such that with probability* $1 - e^{-\Omega(n)}$ *Algorithm 9 optimizes the function* LeadingOnes *within* T *steps where* $c_1 n^2 \log n \leq T \leq c_2 n^2 \log n$ *holds.*

Proof. Assume that the current string x of Algorithm 9 contains exactly i leading ones, i. e., $\text{LeadingOnes}(x) = i$. Then, there is at least one mutation that flips the $(i+1)$-th bit in x and increases the function value by at least 1. This mutation has probability at least $(1/n)(1-1/n)^{n-1} > 1/(en)$ for $p(n) = 1/n$. This is the case each $\lfloor \log n \rfloor$-th step. In all other steps the number of leading ones cannot decrease. Thereby, ignoring all other steps can only increase the number of generations before the global maximum is reached. We have $en \log n$ as upper bound for the expected waiting time for one improvement. After at most n improvements the global maximum is reached. This leads to $O(n^2 \log n)$ as upper bound for the expected running time. The probability that after $2en$ steps with mutation probability $p(n) = 1/n$ the number of leading ones is not increased by at least one is upper bounded by 1/2. For optimization of LeadingOnes at most n such increasements can be necessary. We apply Chernoff bounds (Hagerup and Rüb 1989) and get that with probability $1 - e^{-\Omega(n)}$ all necessary increasements occur within $3en^2$ steps with mutation probability $p(n) = 1/n$. Therefore, with probability $1 - e^{-\Omega(n)}$ after $3en^2 \log n$ generations the unique global optimum is reached.

The lower bound can be proved in a similar way as for the (static) (1+1) EA with $p(n) = 1/n$ (Droste, Jansen, and Wegener 1998a). The main ideas that are additionally needed are that the varying mutation probabilities do not substantially enlarge the probability to enlarge the function value and that the number of enlargements in one phase can be controlled.

Assume that the current string x contains exactly i leading ones, i. e., $\textsc{LeadingOnes}(x) = i$ and that $i < n-1$ holds. We have $x_{i+1} = 0$ in this case. It is obvious that the $n-i-1$ bits $x_{i+2}, x_{i+3}, \ldots, x_n$ are all totally random, i. e., for all $y \in \{0,1\}^{n-i-1}$ we have $\text{Prob}\,(x_{i+1}x_{i+2}\cdots x_n = y) = 2^{-n+i+1}$. We consider a run of Algorithm 9 and start our considerations at the first point of time where $\textsc{LeadingOnes}(x) \geq n/2$ holds. We know that for each constant $\delta > 0$, we have that the probability that $\textsc{LeadingOnes}(x) > (1+\delta)n/2$ holds at this point of time is upper bounded by $e^{-\Omega(n)}$. The probability to increase the function value in one generation is upper bounded by

$$(1-p(n))^{\textsc{LeadingOnes}(x)} \cdot p(n) \leq (1-p(n))^{n/2} \cdot p(n).$$

We consider a subphase of length $\lfloor \log n \rfloor$, such that all $\lfloor \log n \rfloor$ different mutation probabilities are used within this phase. The probability for increasing the function value in one such subphase is upper bounded by

$$\sum_{i=0}^{\lfloor \log n \rfloor -1} \frac{2^i}{n} \cdot \left(1 - \frac{2^i}{n}\right)^{n/2} < \frac{1}{n} \sum_{i=0}^{\infty} 2^i \cdot e^{-2^{i-1}} \leq \frac{\beta}{n},$$

where β is a positive constant.

We say that a generation is successful, if the function value is increased. The probability to increase the function value by at least $k > 0$ in a successful generation equals 2^{-k+1}. Therefore, the probability to increase the function value by at least $2t$ in t successful generations is upper bounded by 2^{-t}. We conclude that with probability at least $1 - e^{-\Omega(n)}$ there have to be at least $((1-\delta)/4)\,n$ successful generations before the global optimum is reached.

We consider a slightly modified random process. In this new process after a successful generation the next $\lfloor \log n \rfloor$ generations are guaranteed not to be successful. By this modification it follows that in each subphase there is at most one successful generation. We note that this modified process may need longer to reach the global optimum. But the number of additional generations needed is upper bounded by $n \lfloor \log n \rfloor$, since after at most n successful generations, the global optimum is surely reached. Obviously, the probability to increase the function value in one subphase is bounded above by β/n for the modified process, too. We conclude that with probability $1 - e^{-\Omega(n)}$ within $((1-\delta)/(8\beta))n^2$ subphases there are at most $((1-\delta)/4)n$ successful generations.

Therefore, with probability $1 - e^{-\Omega(n)}$ Algorithm 9 does not reach the global optimum within

$$\frac{1-\delta}{8\beta} n^2 \lfloor \log n \rfloor - n \lfloor \log n \rfloor$$

generations. Since δ and β are constants with $0 < \delta < 1$ and $\beta > 0$ we see that $\Omega\left(n^2 \log n\right)$ generations are needed with probability $1 - e^{-\Omega(n)}$. □

Theorem 13. *The expected running time of Algorithm 9 on the function* OneMax *is upper bounded by $O(n \log^2 n)$. The expected running time of Algorithm 9 on an arbitrary linear function is bounded by $O(n^2 \log n)$.*

Sketch of Proof: For OneMax we partition $\{0,1\}^n$ into the sets

$$F_i := \{x \in \{0,1\}^n \mid \textsc{OneMax}(x) = i\}.$$

For a linear function f with $f(x) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n$ (we can assume without loss of generality $w_0 = 0$, $w_1 \geq w_2 \geq \cdots \geq w_n$) we use the partition

$$F_i^* := \{x \in \{0,1\}^n \mid w_1 + \cdots + w_i \leq f(x) < w_1 + \cdots + w_{i+1}\}.$$

Note that for all $i > j$ we have $\textsc{OneMax}(x_i) > \textsc{OneMax}(x_j)$ ($f(y_i) > f(y_j)$) for all $x_i \in F_i, x_j \in F_j$ ($y_i \in F_i^*, y_j \in F_j^*$). For $\textsc{OneMax}$ there are $n-i$ mutations of a single bit to leave F_i. For f, there is 1 mutation of a single bit to leave F_i^*. Therefore, in steps with $p(n) = 1/n$ we have at least probability $\binom{n-i}{1}(1/n)(1-1/n)^{n-1} \geq (n-i)/(en)$ to leave F_i and at least probability $(1/n)(1-1/n)^{n-1} \geq 1/(en)$ to leave F_i^*. This is the case each $\lfloor \log n \rfloor$-th step. Again, all other steps cannot do any harm, so by ignoring them we can only increase the number of steps needed for optimization. This leads to an upper bound on the expected running time of $\log n \left(\sum\limits_{i=1}^{n} en/i \right) = O(n \log^2 n)$ for $\textsc{OneMax}$ and $\log n \left(\sum\limits_{i=1}^{n} en \right) = O(n^2 \log n)$ for f. □

The exact asymptotic running time of Algorithm 9 on $\textsc{OneMax}$ and on arbitrary linear functions is still unknown. For linear functions one may conjecture an upper bound of $O(n \log^2 n)$. We see that Algorithm 9 is by far faster than the (1+1) EA with $p(n) = 1/n$ in the worst case and only by a factor $\log n$ slower in typical cases, where already the (1+1) EA with the static choice $p(n) = 1/n$ is efficient. Of course, these are not enough reasons to support Algorithm 9 as a "better" general optimization heuristic than the (1+1) EA with $p(n) = 1/n$ fixed. Now, we present an example where the dynamic variant by far outperforms the static choice $p(n) = 1/n$ and finds a global optimum with high probability in a polynomial number of generations.

We construct a function that serves as an example with the following properties. There is a kind of path to a local optimum, such that the path is easy to find and to follow with mutation probability $1/n$ and a local maximum is quickly found. Then, there is a kind of gap to all points with maximal function value, that can only be reached via a direct mutation. For such a direct mutation many bits (in the order of $\log n$) have to flip simultaneously. This is unlikely to happen with $p(n) = 1/n$. But raising the mutation probability to a value in the order of $(\log n)/n$ gives a good probability for this final step to a global optimum. Since Algorithm 9 uses both probabilities each $\lfloor \log n \rfloor$-th step, it has a good chance to quickly follow the path to the local maximum and jump over the gap to a global one.

Definition 14. *Let $n = 2^k$ be large enough, such that $n/\log n > 8$. First, we define a partition of $\{0,1\}^n$ into five sets, namely*

$$\begin{aligned}
L_1 &:= \{x \in \{0,1\}^n \mid n/4 < \|x\|_1 < 3n/4\}, \\
L_2 &:= \{x \in \{0,1\}^n \mid \|x\|_1 = n/4\}, \\
L_3 &:= \left\{x \in \{0,1\}^n \mid \exists i \in \{0,1,\ldots,(n/4)-1\} : x = 1^i 0^{n-i}\right\}, \\
L_4 &:= \left\{x \in \{0,1\}^n \mid (\|x\|_1 = \log n) \wedge \left(\sum_{i=1}^{2\log n} x_i = 0\right)\right\}, \textit{ and} \\
L_0 &:= \{0,1\}^n \setminus (L_1 \cup L_2 \cup L_3 \cup L_4),
\end{aligned}$$

where $1^i 0^{n-i}$ denotes the string with i consecutive ones followed by $n-i$ consecutive zeros.

The function PathToJump: $\{0,1\}^n \to \mathbb{R}$ *is defined by*

$$\textsc{PathToJump}(x) := \begin{cases} n - \|x\|_1 & \text{if } x \in L_1, \\ (3/4)n + \sum_{i=1}^{n/4} x_i & \text{if } x \in L_2, \\ 2n - i & \text{if } x \in L_3 \text{ and } x = 1^i 0^{n-i}, \\ 2n+1 & \text{if } x \in L_4, \\ \min\{\|x\|_1, n - \|x\|_1\} & \text{if } x \in L_0. \end{cases}$$

Theorem 15. *The probability that the* $(1+1)$ EA *with* $p(n) = 1/n$ *needs a superpolynomial number of steps to optimize* PathToJump *converges to* 1.

Proof. With probability exponentially close to 1 the initial string belongs to $L_1 \cup L_2 \cup L_3$. Thus, L_0 is never entered. All global maxima belong to L_4 and have Hamming distance at least $\log n$ to all points in $L_1 \cup L_2 \cup L_3$. The probability for a mutation of at least $\log n$ bits simultaneously is bounded above by

$$\binom{n}{\log n} \left(\frac{1}{n}\right)^{\log n} \leq \frac{1}{(\log n)!}.$$

Therefore, the probability that such a mutation occurs in $n^{O(1)}$ steps is upper bounded by $n^{O(1)}/((\log n)!)$ and converges to 0. □

We remark that Theorem 15 can be generalized to all mutation probabilities substantially different from $(\log n)/n$.

Theorem 16. *The expected number of steps until Algorithm 9 finds a global optimum of the function* PathToJump *is bounded above by* $O(n^2 \log n)$.

Proof. We define levels F_i of points with the same function value by

$$F_i := \{x \in \{0,1\}^n \mid \textsc{PathToJump}(x) = i\}.$$

Note, that there are less than $2n+2$ different levels F_i with $F_i \neq \emptyset$. Algorithm 9 can enter these levels only in order of increasing function values. For each level F_i we derive a lower bound for the probability to reach some $x' \in F_j$ with $j > i$ in one subphase, i. e., a lower bound on the probability

$$q_i := \max\left\{ \min\left\{ \sum_{x' \in \bigcup_{j>i} F_j} \left(\frac{2^k}{n}\right)^{H(x,x')} \left(1 - \frac{2^k}{n}\right)^{n-H(x,x')} \mid x \in F_i \right\} \mid 0 \leq k \leq \lfloor \log n \rfloor \right\},$$

where $H(x,x')$ denotes the Hamming distance between x and x'. Clearly, a lower bound $q_i' \leq q_i$ yields an upper bound of $1/q_i'$ on the expected number of subphases until Algorithm 9 leaves F_i and reaches another level. By summing up the upper bounds for all F_i, $i \leq 2n$, we get an upper bound on the expected running time.

We distinguish four different cases with respect to i.

Case 1: $i \in \{0, 1, \ldots, (3/4)n - 1\}$
We have $x \in L_0 \cup L_1$, so it is sufficient to mutate exactly one of at least $n/4$ different bits to increase the function value. This implies

$$q_i \geq \binom{n/4}{1} \frac{1}{n} \left(1 - \frac{1}{n}\right)^{n-1} = \Omega(1)$$

for this case. So we have $O(n)$ as upper bound for the expected number of subphases the algorithm spends in this part of the search space.

Case 2: $i \in \{(3/4)n, \ldots, n-1\}$
We have $x \in L_2$, it is $\|x\|_1 = n/4$. Among the bits $x_1, x_2, \ldots, x_{n/4}$ there are $i - (3/4)n$ bits with value 1. Among the other bits there are $n - i$ bits with value 1. In order to increase the function value it is sufficient to mutate exactly one of the $(n/4) - (i - (3/4)n) = n - i$ bits with value 0 in the first part of x and exactly one of the $n - i$ bits with value 1 in the second part of x simultaneously. Therefore, we have

$$q_i \geq (n-i)^2 \left(\frac{1}{n}\right)^2 \left(1 - \frac{1}{n}\right)^{n-2} = \Omega\left(\left(\frac{n-i}{n}\right)^2\right)$$

as lower bound and

$$\sum_{i=(3/4)n}^{n-1} \left(\frac{n}{n-i}\right)^2 = O\left(n^2\right)$$

as upper bound on the expected number of subphases Algorithm 9 spends in L_2.

Case 3: $i \in \{n, \ldots, 2n-1\}$
We have $x = 1^{2n-i}0^{i-n} \in L_3$. Obviously, it is sufficient to mutate exactly the most right bit with value 1 to increase the function value. This yields

$$q_i \geq \frac{1}{n}\left(1 - \frac{1}{n}\right)^{n-1} = \Omega\left(\frac{1}{n}\right)$$

as lower bound on the probability and $O\left(n^2\right)$ as upper bound on the expected number of subphases until Algorithm 9 leaves this part of the search space.

Case 4: $i = 2n$
In order to increase the function value it is necessary and sufficient that exactly $\log n$ bits, which all do not belong to the first $2\log n$ positions in x mutate simultaneously. This yields

$$q_i \geq \binom{n - 2\log n}{\log n} p(n)^{\log n} \left(1 - p(n)\right)^{n - \log n}$$

where we can choose $p(n) \in \left\{1/n, 2/n, \ldots, 2^{\lfloor \log n \rfloor}/n\right\}$ to maximize this lower bound. It is easy to see, that one should choose $p(n) = \Theta((\log n)/n)$ in order to maximize the bound. Therefore, we set $p(n) := (c \log n)/n$ for some positive constant c and discuss the value of c

later. This yields

$$\begin{aligned} q_i &\geq \binom{n-2\log n}{\log n}\left(\frac{c\log n}{n}\right)^{\log n}\left(1-\frac{c\log n}{n}\right)^{n-\log n} \\ &\geq \left(\frac{n-2\log n}{\log n}\right)^{\log n}\left(\frac{c\log n}{n}\right)^{\log n}\left(1-\frac{c\log n}{n}\right)^{n}\left(1-\frac{c\log n}{n}\right)^{-\log n} \\ &= \left(1-\frac{2\log n}{n}\right)^{\log n} c^{\log n}\cdot\Omega\left(\left(\frac{1}{n}\right)^{c/\ln 2}\right)\cdot\Omega(1) \\ &= \Omega\left(n^{(\log c)-c/\ln 2}\right) \end{aligned}$$

as lower bound on the probability and $n^{(c/\ln 2)-\log c}$ as upper bound on the number of subphases for this final mutation to a global optimum. Obviously, $(c/\ln 2) - \log c$ becomes minimal for $c = 1$. Unfortunately, it is not guaranteed that the value $(\log n)/n$ is used as mutation probability. Nevertheless, it is clear that for each d with $0 < d < n/(2\log n)$ every $\lfloor\log n\rfloor$-th generation a value from the interval $[(d\log n)/n; (2d\log n)/n]$ is used as mutation probability $p(n)$. We choose $d = \ln 2$ and get $O\left(n^{1-\log\ln 2}\right) = O\left(n^{1.53}\right)$ as upper bound on the expected number of subphases needed for the final step.

Altogether, we have $O\left(n^2\right)$ as an upper bound on the expected number of subphases before Algorithm 9 reaches the global optimum. As each subphase contains $\lfloor\log n\rfloor$ generations, we have $O\left(n^2\log n\right)$ as upper bound on the expected running time. □

We note, that the probability, that the optimum is not reached within $O(n^3\log n)$ steps is exponentially small.

One may speculate that this dynamic variant of the (1+1) EA is always by at most a factor $\log n$ slower than its static counterpart given that the fixed value $p(n)$ is used by Algorithm 9, i. e., we have $p(n) = 2^t/n$ for some $t \in \{1, \ldots, \lfloor(\log n) - 1\rfloor\}$. The reason for this speculation is clear: the fixed value of $p(n)$ the (static) (1+1) EA uses is tried by Algorithm 9 in each $\lfloor\log n\rfloor$-th step. But this speculation is wrong. Our proof idea is roughly speaking the following. In principle, Algorithm 9 can follow the same paths as the (1+1) EA with $p(n) = 1/n$ fixed. But if in some distance to the followed path there are so-called traps that once they are entered are difficult to leave, Algorithm 9 may be inferior. Due to the fact that it often uses mutation probabilities much larger than $1/n$, it has a much larger chance to reach traps that have a not too large distance to the path. In the following, we define an example function denoted by PathWithTrap and prove that the (1+1) EA with $p(n) = 1/n$ is with high probability by far superior to Algorithm 9. One important ingredient of the definition of PathWithTrap are long paths introduced by Horn, Goldberg, and Deb (1994).

Definition 17. *For $n \in \mathbb{N}$ and $k \in \mathbb{N}$ with $k > 1$ and $(n-1)/k \in \mathbb{N}$ we define the long k-path of dimension n P_k^n as a sequence of $|P_k^n|$ strings inductively. For $n = 1$ we set $P_k^1 := (0, 1)$. Let the long k-path of dimension $n - k$ $P_p^{n-k} = (v_1, \ldots, v_l)$ be well-defined. Then we define $S_0 := (0^k v_1, \ldots, 0^k, v_l)$, $S_1 := (1^k v_l, \ldots, 1^k v_1)$, $B_k^n := (0^{k-1}1v_l, 0^{k-2}1^2 v_l, \ldots, 01^{k-1}v_l)$. We obtain P_k^n as concatenation of S_0, B_k^n, S_1.*

Long k-paths have some structural properties that make them a helpful tool. A proof for the following lemma can be found in (Rudolph 1997).

Lemma 18. *Let $n, k \in \mathbb{N}$ be given such that the long k-path of dimension n is well-defined. All $|P_k^n| = (k+1)2^{(n-1)/k} - k + 1$ points in P_k^n are different. For all $i \in \{1, 2, \ldots, k-1\}$ we have, that if $x \in P_k^n$ has at least i successors on the path, then the i-th successor has Hamming distance i to x and all other successors of x have Hamming distances different from i.*

Definition 19. *For $k \in \mathbb{N}$ with $k > 20$ we define the function* $\textsc{PathWithTrap}\colon \{0,1\}^n \to \mathbb{R}$ *as follows. Let $n := 2^k$, $j := 3k^2 + 1$. Let p_i denote the i-th point of the long k-path of dimension j. We define a partition of $\{0,1\}^n$ into seven sets $P_0, \ldots, P_6$.*

$$
\begin{array}{lcl}
P_1 & := & \{x \in \{0,1\}^n \mid 7n/16 < \|x\|_1 \leq 9n/16\} \\
P_2 & := & \{x \in \{0,1\}^n \mid \|x\|_1 = 7n/16\} \\
P_3 & := & \left\{x \in \{0,1\}^n \mid \left(\sqrt{n} < \|x\|_1 < 7n/16\right) \wedge \left(\sum\limits_{i=j+1}^{j+\sqrt{n}} x_i = \sqrt{n}\right)\right\} \\
P_4 & := & \left\{x \in \{0,1\}^n \mid \exists i \in \left\{1, 2, \ldots, \sqrt{n}\right\} : x = 0^j 1^i 0^{n-i-j}\right\} \\
P_5 & := & \left\{x \in \{0,1\}^n \mid \left(x_1 x_2 \cdots x_j \in P_k^j\right) \wedge \left(\sum\limits_{i=j+1}^{n} x_i = 0\right)\right\} \\
P_6 & := & \left\{x \in \{0,1\}^n \mid \left(x_1 x_2 \cdots x_j \in P_k^j\right) \wedge \left(\sum\limits_{i=j+1}^{j+k} x_i = 0\right) \wedge \left(\sum\limits_{i=j+1}^{n} x_i = k\right)\right\} \\
P_0 & := & \{0,1\}^n \setminus (P_1 \cup P_2 \cup P_3 \cup P_4 \cup P_5 \cup P_6)
\end{array}
$$

Given this partition we define

$$
\textsc{PathWithTrap}(x) := \begin{cases}
n - \|x\|_1 & \text{if } x \in P_1, \\
n - \|x\|_1 + \sum\limits_{i=j+1}^{j+\sqrt{n}} x_i & \text{if } x \in P_2, \\
2n - \|x\|_1 & \text{if } x \in P_3, \\
4n - i & \text{if } (x \in P_4) \wedge \left(x = 0^j 1^i 0^{n-i-j}\right), \\
4n + 2i & \text{if } (x \in P_5) \wedge \left(x_1 \cdots x_j = p_i \in P_k^j\right), \\
4n + 2\left|P_k^j\right| - 1 & \text{if } x \in P_6, \\
\min\{\|x\|_1, n - \|x\|_1\}/3 & \text{if } x \in P_0,
\end{cases}
$$

for all $x \in \{0,1\}^n$.

Obviously, there is a unique string x_{opt} with maximal function value under $\textsc{PathWithTrap}$: This string is equal to the very last point of P_k^j on the first j bits and is all zero on the other bits. Moreover, for all $x_0 \in P_0$, $x_1 \in P_1$, $x_2 \in P_2$, $x_3 \in P_3$, $x_4 \in P_4$, $x_5 \in P_5 \setminus \{x_{\text{opt}}\}$, $x_6 \in P_6$ and $x_7 = x_{\text{opt}}$ we have

$$0 \leq i < j \leq 7 \Rightarrow \textsc{PathWithTrap}(x_i) < \textsc{PathWithTrap}(x_j).$$

The main idea behind the definition of the function $\textsc{PathWithTrap}$ is the following. There is a more or less easy to follow path leading to the global optimum x_{opt}. The length of the path is $\Theta(n^3 \log n)$, so that both algorithms follow the path for a quite long time. In some sense parallel to this path there is an area of points, P_6, that all have second best function value. The Hamming distance between these points and the path is about $\log n$. Therefore, it is very unlikely that this area is reached using a mutation probability of $1/n$. On the other hand, with varying mutation probabilities "jumps" of length $\log n$ do occur and this area can be reached. Then, only a direct jump to x_{opt} is accepted. But, regardless of the mutation probability, the probability for such a mutation is very small. In this sense we call P_6 a trap. Therefore, it is at least intuitively clear, that the (1+1) EA is more likely to be successful on PathWithTrap than Algorithm 9.

Theorem 20. *The* (1+1) *EA with mutation probability* $p(n) = 1/n$ *finds the global optimum of* PathWithTrap *with probability* $1 - e^{-\Omega(\log n \log \log n)}$ *within* $O(n^4 \log^2 n \log \log n)$ *steps.*

Sketch of Proof: With probability $1 - e^{-\Omega(n)}$ the initial string x belongs to P_1. Then, no string in P_0 can ever be reached. For all $x \in \{0,1\}^n \setminus (P_0 \cup P_6)$ and all $y \in P_6$ we have that the Hamming distance between x and y is lower bounded by $\log n$. The probability for a mutation of at least $\log n$ bits simultaneously is upper bounded by

$$\binom{n}{\log n} \left(\frac{1}{n}\right)^{\log n} \leq \frac{1}{(\log n)!} = e^{-\Omega(\log n \log \log n)}.$$

Therefore, with probability $1 - e^{-\log n \log \log n}$ P_6 is not reached within $n^{O(1)}$ steps. Under the assumption that P_6 is not reached one can in a way similar to the proof of Theorem 16 consider levels of equal fitness values and prove that with high probability the (1+1) EA with $p(n) = 1/n$ reaches the global optimum fairly quickly. □

Theorem 21. *Algorithm 9 does not find the global optimum of* PathWithTrap *within* $n^{O(1)}$ *steps with probability* $1 - e^{-\Omega(\log^2 n)}$.

Sketch of Proof: The proof of the lower bound for Algorithm 9 is much more involved than the proof of the upper bound for the (1+1) EA.

Again, with probability $1 - e^{-\Omega(n)}$ the initial bit string belongs to P_1 and P_0 will never be entered. For all $x \in P_1 \cup P_2 \cup P_3$ we have that all strings $y \in P_5$ have Hamming distance at least $\sqrt{n}/2$. Therefore, for all mutation probabilities the probability to reach P_5 from somewhere in $P_1 \cup P_2 \cup P_3$ (thereby "skipping" P_4) within $n^{O(1)}$ steps is upper bounded by $e^{-\Omega(\sqrt{n} \log n)}$. We conclude that some string in P_4 is reached before the global optimum is reached with high probability.

It is not too hard to see that with probability $1 - e^{-\Omega(\log^2 n)}$ within $n^{O(1)}$ steps no mutation of at least $(\log^2 n)/n$ bits simultaneously occurs. We divide P_5 into two halves according to increasing function values. One can prove that with probability $1 - e^{-\Omega(\log^2 n)}$ the first point $y \in P_5$ that is reached via a mutation from some point $x \in P_4$ belongs to the first half. Therefore, the length of the rest of the long k-path the algorithm faces is still $\Theta(n^3 \log n)$. We conclude that with probability $1 - e^{-\Omega(\log^2 n)}$ Algorithm 9 spends $\Omega(n^3)$ steps on the path. In each of these steps where the current mutation probability equals $(\log n)/n$ with probability at least

$$\left(\frac{n}{2\log n}\right)^{\log n} \cdot \left(\frac{\log n}{n}\right)^{\log n} \cdot \left(1 - \frac{\log n}{n}\right)^{n - \log n} \geq \frac{e^{-\log n}}{n} > n^{-2.45}$$

some point in P_6, the trap, is reached. Therefore, with probability $1 - e^{-\Omega(\sqrt{n})}$ the trap is entered within the $\Omega(n^3/\log n)$ steps that we have with this mutation probability on the path with high probability. So, altogether we have that with probability $1 - e^{-\Omega(\log^2 n)}$ Algorithm 9 enters the trap. Once this happens, i. e., some $x \in P_6$ becomes the current string of Algorithm 9, a mutation of exactly $\log n$ specific bits is needed to reach the global optimum. The probability that this happens in one step is upper bounded by

$$\max\left\{\left(\frac{2^i}{n}\right)^{\log n}\left(1-\frac{2^i}{n}\right)^{n-\log n} \mid i \in \{0,1,\ldots,\lfloor\log n\rfloor - 1\}\right\}$$
$$\leq \left(\frac{\log n}{n}\right)^{\log n}\left(1-\frac{\log n}{n}\right)^{n-\log n} = e^{-\Omega(\log^2 n)}.$$

This yields on the one hand $\Omega(e^{\log^2 n})$ as lower bound on the expected running time. And on the other hand we have that with probability $1 - e^{-\Omega(\log^2 n)}$ Algorithm 9 does not find the global optimum x_{opt} of PathWithTrap within $n^{O(1)}$ steps. □

5 CONCLUSIONS

We have studied two variants of the (1+1) EA with dynamic parameter control. The first variant uses a probabilistic selection mechanism that accepts worsenings with a probability that depends on a parameter α. We have proved for a simple example function that a dynamic parameter control can tremendously decrease the expected running time compared to optimal static choices. Our example proves an exponential gap between the Metropolis algorithm and simulated annealing in a simple and understandable way.

Second, we have considered the (1+1) EA with dynamically changing mutation probabilities. We have seen that this leads to an enhanced robustness while only slowing the algorithm down by the factor $\log n$ for typical functions. For one example we have seen that the dynamic variant outperforms the static one for the most recommended static choice $p(n) = 1/n$. It remains open, whether in practical situations the enhanced robustness of the dynamic variant turns out to be an advantage.

Acknowledgments

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

References

T. Bäck (1993). Optimal mutation rates in genetic search. In S. Forrest (Ed.), *Proc. of the 5th Int. Conf. on Genetic Algorithms (ICGA '93)*, 2–8. Morgan Kaufmann.

T. Bäck (1998). An overview of parameter control methods by self-adaptation in evolutionary algorithms. *Fundamenta Informaticae 35*, 51–66.

H.-G. Beyer (1996). Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation 3*(3), 311–347.

S. Droste, T. Jansen, and I. Wegener (1998a). On the analysis of the $(1+1)$ evolutionary algorithm. Technical Report CI-21/98, Univ. Dortmund, Collaborative Research Center 531.

S. Droste, T. Jansen, and I. Wegener (1998b). A rigorous complexity analysis of the $(1+1)$ evolutionary algorithm for separable functions with Boolean inputs. *Evolutionary Computation 6*(2), 185–196.

J. Garnier, L. Kallel, and M. Schoenauer (1999). Rigorous hitting times for binary mutations. *Evolutionary Computation 7*(2), 173–203.

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

J. Horn, D. E. Goldberg, and K. Deb (1994). Long path problems. In Y. Davidor, H.-P. Schwefel, and R. Männer (Eds.), *Proceedings of the 3rd Parallel Problem Solving From Nature* (*PPSN III*), Volume 866 of *Lecture Notes in Computer Science*, Berlin, 149–158. Springer.

M. Jerrum and A. Sinclair (1997). The Markov chain Monte Carlo method: an approach to approximate counting and integration. In D. S. Hochbaum (Ed.), *Approximation Algorithms for NP-hard Problems*, 482–520. PWS Publishers.

M. Jerrum and G. B. Sorkin (1998). The Metropolis algorithm for graph bisection. *Discrete Applied Mathematics 82*, 155–175.

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

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller (1953). Equation of state calculation by fast computing machines. *Journal of Chemical Physics 21*, 1087–1092.

H. Mühlenbein (1992). How genetic algorithms really work. Mutation and hillclimbing. In R. Männer and R. Manderick (Eds.), *Proc. of the 2nd Parallel Problem Solving from Nature (PPSN II)*, 15–25. North-Holland.

Y. Rabani, Y. Rabinovich, and A. Sinclair (1998). A computational view of population genetics. *Random Structures and Algorithms 12*(4), 313–334.

G. Rudolph (1997). *Convergence Properties of Evolutionary Algorithms*. Verlag Dr. Kovač.

G. Rudolph (1999). Self-adaptation and global convergence: A counter-example. In *Proc. of the Congress on Evolutionary Computation (CEC '99)*, 646–651. IEEE Press.

H.-P. Schwefel (1995). *Evolution and Optimum Seeking*. Wiley.

A. Sinclair (1993). *Algorithms for Random Generation and Counting: A Markov Chain Approach*. Boston: Birkhäuser.

G. B. Sorkin (1991). Efficient simulated annealing on fractal energy landscapes. *Algorithmica 6*, 367–418.