k-Bit Mutation with Self-Adjusting k Outperforms Standard Bit Mutation

Benjamin Doerr[1], Carola Doerr[2(✉)], and Jing Yang[1]

[1] École Polytechnique, Palaiseau, France
[2] CNRS and Sorbonne Universités, UPMC University Paris 06, LIP6, Paris, France
doerr@lip6.fr

Abstract. When using the classic standard bit mutation operator, parent and offspring differ in a random number of bits, distributed according to a binomial law. This has the advantage that all Hamming distances occur with some positive probability, hence this operator can be used, in principle, for all fitness landscapes. The downside of this "one-size-fits-all" approach, naturally, is a performance loss caused by the fact that often not the ideal number of bits is flipped. Still, the fear of getting stuck in local optima has made standard bit mutation become the preferred mutation operator.

In this work we show that a self-adjusting choice of the number of bits to be flipped can both avoid the performance loss of standard bit mutation and avoid the risk of getting stuck in local optima. We propose a simple mechanism to adaptively learn the currently optimal mutation strength from previous iterations. This aims both at exploiting that generally different problems may need different mutation strengths and that for a fixed problem different strengths may become optimal in different stages of the optimization process.

We experimentally show that our simple hill climber with this adaptive mutation strength outperforms both the randomized local search heuristic and the (1+1) evolutionary algorithm on the LeadingOnes function and on the minimum spanning tree problem. We show via mathematical means that our algorithm is able to detect precisely (apart from lower order terms) the complicated optimal fitness-dependent mutation strength recently discovered for the OneMax function. With its self-adjusting mutation strength it thus attains the same runtime (apart from $o(n)$ lower-order terms) and the same (asymptotic) 13 % fitness-distance improvement over RLS that was recently obtained by manually computing the optimal fitness-dependent mutation strength.

1 Introduction

When using a bit-string representation in evolutionary computation, that is, when the search space is $\Omega = \{0,1\}^n$, then *standard bit mutation* is the by far most-employed mutation operator. It creates a new individual (*offspring*) from an existing one (*parent*) by flipping each bit of the parent independently with some probability p, often with $p = 1/n$. By this, the Hamming distance of parent

J. Handl et al. (Eds.): PPSN XIV 2016, LNCS 9921, pp. 824–834, 2016.
DOI: 10.1007/978-3-319-45823-6_77

and offspring, that is, the number of positions in which the strings differ, follows a binomial distribution with parameters n and p. If $p = 1/n$, then the expected distance is one (following the idea that mutation should be a minimalistic change of the individual), but all distances in $[0..n] := \{0, 1, \ldots, n\}$ occur with positive probability. Consequently, a hill climber using this mutation operator (e.g., the $(1 + 1)$ evolutionary algorithm (EA)) cannot get permanently stuck in a local optimum, no matter what the fitness landscape of the underlying optimization problem looks like.

The downside of this "one-size-fits-all" approach, naturally, is that it does not exploit particular properties of the landscape. It is well-known (though not in all cases explicitly proven) that for simple fitness landscapes like those of OneMax, linear functions, LeadingOnes, and royal-road functions, flipping only single bits gives smaller optimization times than using standard bit mutation (often, the improvement is by a factor of $e \approx 2.718$). For the minimum spanning tree (MST) problem, a mutation operator randomly choosing between flipping a single random bit or two random bits gives again better results than standard bit mutation, whereas flipping always only one bit or always exactly two bits in most cases lets the algorithm get stuck in a local optimum.

Our Results: The examples above show that using a problem-specific optimal mutation strength can lead to a fair speed-up over standard bit mutation, however, with the risk of making the algorithm fail badly when choosing a wrong mutation strength. For this reason, we design a simple hill climber that autonomously tries to choose the optimal mutation rate by analyzing the past performance of the different mutation strengths. This aims both at exploiting that different problems ask for different mutation strengths and at exploiting that for a fixed problem the optimal mutation strength may change during the optimization process; a problem even less understood than the right problem-specific static mutation strength.

We experimentally analyze our new algorithm on the LeadingOnes and the MST problem. We observe that, for suitable parameter settings, it clearly outperforms the $(1 + 1)$ EA. Interestingly, it even beats the randomized local search (RLS) heuristic (flipping always one bit) for the LeadingOnes problem and the variant of RLS flipping one or two bits for the MST problem. This shows that for these problems a better performance can be obtained from a mutation strength that changes over time, and that our algorithm is able to find such superior fitness-dependent mutation strengths.

The heart of our work is making this effect mathematically precise for OneMax. For this function, an optimal fitness-dependent mutation strength was recently found in [4]. This optimal mutation strength is quite particular. It uses, for all but a lower order fraction of the runtime, the mutation strength one (that is, flips a random bit). In a short initial segment of the optimization process, flipping a larger number of bits is superior. The optimal number of bits is decreasing with increasing fitness, but is always an odd number. Despite differing from RLS only in a short period, the simple hill climber using this fitness-dependent mutation strength with a fixed budget of iterations computes solutions that have

an expected fitness distance that is 13 % smaller than those computed by RLS, making it the current best unbiased mutation-based optimizer for OneMax (cee [8] for a discussion on the fixed-budget performance measure). However, due to its complicated nature, it is not clear how a non-expert should find such fitness-dependent mutation strengths.

For our new algorithm with self-adjusting mutation strength, we show that it essentially is able to find this optimal mutation schedule on the fly. More precisely, with high probability our algorithm always (apart from a lower-order fraction of the iterations) uses a mutation strength which gives an expected progress equal to the best possible progress (again, apart from lower order terms). Consequently, our algorithm has the same optimization time (apart from an $o(n)$ additive lower order term) and the same asymptotic 13 % superiority in the fixed budget perspective as the algorithm with the hand-crafted mutation strength schedule from [4].

These first results indicate that a self-adjusting mutation strength both works well for problems with different optimal mutation strengths (in an even better way than the "one-size-fits-all" approach of standard bit mutation) and, beyond this, can also find good fitness-dependent mutation schedules. We defer the details to the following sections, where we propose our new algorithm (Sect. 2), give some experimental evidence for its superiority (Sect. 3), conduct a rigorous runtime analysis for OneMax (including the proof that the optimal mutation strength essentially is always employed) in Sect. 4, and discuss how to choose parameters and take other design choices (Sect. 5).

Discussion of Previous Works on Adaptive Mutation Operators: Given the importance of mutation, not surprisingly, there is a plethora of works on adaptive uses of mutation. With very few exceptions, these works are experimental in nature. They mostly indicate that an adaptive change of how mutation is performed can be beneficial. However, it seems hard to derive generally accepted design rules from these works. For reasons of space, we cannot avoid referring the reader to some of the central works [1,5,9,12] and the extensive follow-up work.

On the theoretical side, a first dynamic setting of the mutation rate was proposed and analyzed in [7]. They propose to use the $(1+1)$ EA with a mutation rate that, depending on the iteration counter, takes a value in $\{2^k/n \mid k = 0, 1, 2, \ldots, \lceil \log_2(n) \rceil - 2\}$. They construct an example function where the $(1+1)$ EA with this dynamic mutation rate greatly outperforms the $(1+1)$ EA with any fixed mutation rate. However, they also show that their EA has an asymptotically larger runtime on most classic test functions. In [2], a fitness-dependent choice of the mutation rate was proposed that improves the runtime of the $(1+1)$ EA on the LeadingOnes function from approximately $0.86n^2$ for the fixed mutation probability $1/n$ to approximately $0.68n^2$. For population-based EAs a rank-based mutation rate has been investigated in [11].

All of the works discussed above use standard bit mutation, that is, flip each bit independently with a certain, adaptively chosen, probability p. Our main point in this work is that flipping a fixed number of r bits, where the mutation

Algorithm 1. RLS with fitness-dependent mutation strength. When maximizing functions $f : \{0,1\}^n \to D$, the algorithm takes as parameter a mutation strength function $r : D \to [1..n]$ describing how many bits to flip given a certain fitness of the current search point. The operator $\text{flip}(x, r)$ generates from x a new search point by flipping exactly r random bit positions.

```
1 Initialization: Choose x ∈ {0,1}^n uniformly at random;
2 Optimization: for t = 1, 2, 3, ... do
3     y ← flip(x, r(f(x)));
4     if f(y) ≥ f(x) then x ← y;
```

strength r is chosen in a self-adjusting manner, is more profitable because it greatly reduces the use of r-bit flips with a sub-optimal mutation strength r. We are not aware of any work on this type of self-adjusting mutation. An optimal fitness-dependent choice of r for OneMax was determined in [4] recently.

2 Randomized Local Search with Fitness-Dependent and Self-adjusting Mutation Strength

In [4], a variant of the classic *randomized local search* (RLS) heuristic with fitness-dependent mutation strength was proposed (Algorithm 1). Whereas the classic version of RLS creates a new search point always by flipping a single random bit, *RLS with fitness-dependent mutation strength* flips a number of bits ("mutation strength") functionally depending on the current fitness.

While it is clear that choosing the best mutation strength for each fitness level can improve the performance, it is not so clear how to find a good mutation strength function. The example of OneMax studied in [4] indicates that a substantial understanding of the underlying optimization problem is necessary to profit from varying the mutation strength depending on the fitness.

To overcome this difficulty, in this work we propose to choose the mutation strength in each iteration based on the experience in the optimization process so far. We enforce gaining a certain experience by designating each iteration with probability δ as a *learning iteration*. In a learning iteration we flip a random number of bits (chosen uniformly at random from a domain $[1..r_{\max}]$) and store (in an efficient manner) the progress made in these iterations. In all regular iterations, we use the experience made in these learning iterations to determine the most promising mutation strength and create the offspring with this mutation strength.

More precisely, let us denote by x_t the search point after the t-th iteration, that is, after the mutation and selection step of iteration t. Denote by x_0 the random initial search point. If t is a learning iteration, denote by r_t the random mutation strength r used in this iteration. Otherwise set $r_t = 0$.

The main idea of our algorithm is to learn the efficiency of the mutation strengths, that is, the expected progress made when flipping r bits, for all

$r \in [1..r_{\max}]$. We do so via a time-discounted average of the progresses observed in the learning iterations: We define an estimate for the future progress, called *velocity* in the absence of a better name, after the t-th iteration by

$$v_t[r] := \frac{\sum_{s=1}^{t} \mathbf{1}_{r_s=r}(1-\varepsilon)^{t-s}(f(x_s) - f(x_{s-1}))}{\sum_{s=1}^{t} \mathbf{1}_{r_s=r}(1-\varepsilon)^{t-s}}. \tag{1}$$

In this expression, the parameter ε, called *forgetting rate*, determines the decrease of the importance of older information. Since $(1-\varepsilon)^{1/\varepsilon} = (1/e) + o(1)$ for all $\varepsilon = o(1)$, the reciprocal $1/\varepsilon$ of the forgetting rate is (apart from constant factors) the information half-life.

We first observe that we can compute the velocities iteratively and thus, unlike equation (1) might suggest, do not need to store the full history of the learning iterations. To this aim, we need to store one additional value for each r, namely the sum of the $(1-\varepsilon)^{t-s}$ terms used in the weighted average, that is,

$$w_t[r] := \sum_{s=1}^{t} \mathbf{1}_{r_s=r}(1-\varepsilon)^{t-s}.$$

Then the following recursive description of the velocities and weight sums is easily seen: If in iteration $t+1$ we have not done a learning step with mutation strength r, that is, $r_{t+1} \neq r$, then $v_{t+1}[r] = v_t[r]$ and $w_{t+1}[r] = (1-\varepsilon)w_t[r]$. If $r_{t+1} = r$, then

$$\begin{aligned} v_{t+1}[r] &= \frac{(1-\varepsilon)w_t[r]v_t[r] + f(x_{t+1}) - f(x_t)}{(1-\varepsilon)w_t[r] + 1}, \\ w_{t+1}[r] &= (1-\varepsilon)w_t[r] + 1. \end{aligned}$$

For exploiting the experience gained in the learning iterations, we adopt a greedy strategy and always choose the mutation strength with highest velocity (breaking ties randomly, but giving preference to the previous-best mutation strength). While we generally postpone a discussion on parameter settings and other design choices to Sect. 5, let us remark already here that our greedy choice of the mutation strength might be detrimental for fitness landscapes in which the optimal mutation strength changes very frequently. There a velocity-weighted random choice might be more fruitful.

From this discussion, we derive the algorithm *RLS with self-adjusting mutation strength* (Algorithm 2).

3 Experimental Results

In this section we describe some experimental results for our algorithm. These are by no means intended to account for a thorough scientific investigation, both for reasons of space and because we feel that the mathematical investigation in the subsequent section is more insightful, also with respect to *why* the proposed ideas

Algorithm 2. RLS with self-adjusting mutation strength. The parameters of the algorithm are the maximum mutation strength $r_{\max}$, the learning rate δ, and the forgetting rate ε.

Initialization:
  Choose $x \in \{0,1\}^n$ uniformly at random;
  for $r = 1$ **to** $r_{\max}$ **do** $v[r] := 0$ and $w[r] := 0$;
  $r^* \leftarrow 1$;
Optimization: for $t = 1, 2, 3, \ldots$ **do**
  $z \leftarrow \text{random}([0,1])$;
  if $z \leq \delta$ **then** % learning iteration
    $r \leftarrow \text{random}(\{1, \ldots, r_{\max}\})$;
    $y \leftarrow \text{flip}(x, r)$;
    $v[r] \leftarrow \frac{(1-\varepsilon)w[r]v[r]+\max\{0, f(y)-f(x)\}}{(1-\varepsilon)w[r]+1}$;
    $w[r] \leftarrow (1-\varepsilon)w[r] + 1$;
    if $f(y) \geq f(x)$ **then** $x \leftarrow y$;
    for $r' \in \{1, \ldots, r_{\max}\} \setminus \{r\}$ **do** $w[r'] \leftarrow (1-\varepsilon)w[r']$;
  else
    $r^+ \leftarrow \text{random}(\text{argmax}_r(v[r]))$;
    if $v[r^+] > v[r^*]$ **then** $r^* \leftarrow r^+$;
    $y \leftarrow \text{flip}(x, r^*)$;
    if $f(y) \geq f(x)$ **then** $x \leftarrow y$;
    for $r \in \{1, \ldots, r_{\max}\}$ **do** $w[r] \leftarrow (1-\varepsilon)w[r]$;

work well. Nevertheless, the experimental results indicate that our new algorithm gives good results also for problems other than OneMax, they give some hints on how to choose the parameters $r_{\max}$, δ and ε (more on this in Sect. 5), and they taught us that finding suitable parameters was not very difficult—we were immediately faster than the $(1+1)$ EA and with at most a few trials were able to beat RLS. All experiments were repeated 100 times; all numbers given below are the averages of these 100 runs.

LeadingOnes Function: The LeadingOnes function is defined by $\text{Lo}(x) := \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\}$, that is, it counts, starting from the left end, how many consecutive ones the bit-string x contains. The expected optimization time (number of iterations until the optimum is found) for RLS is $0.5n^2 \pm O(n)$, that of the $(1+1)$ EA with mutation rate $p = 1/n$ is $0.5n^2(1-1/n)((1-1/n)^{-n}-1) = 0.5(e-1)n^2 \pm O(n) \approx 0.8591n^2$. When taking the asymptotically optimal mutation rate of approximately $1.59/n$, the optimization time drops to approximately $0.7720n^2$. When taking a (best-possible) fitness-dependent mutation rate of $p_i = 1/(i+1)$ at fitness i, then the optimization time drops to $(e/4)n^2 \pm O(n) \approx 0.6796n^2$ [2].

Experimentally, for $n = 10,000$ and taking the parameters $r_{\max} = 5$, $\delta = 0.1$ and $\varepsilon = 1/(5,000,000)$, we observed an average optimization time of 45.0 million

iterations, that is, $0.450n^2$, which clearly beats RLS and all $(1+1)$ EA results described above. The relative standard deviation is low, 4.36 % to be precise.[1]

Minimum Spanning Trees: Given a connected undirected graph $G = (V, E)$ with edge weights $w : E \to \mathbb{R}_{>0}$, the *minimum spanning tree problem* asks for finding a tree in G that connects all vertices and that has minimal total weight. This problem can be solved via evolutionary methods by taking a bit-string representation (each bit describes whether some edge is part of the tree or not) and taking as fitness function (to be minimized) the sum of the weights of the edges in the string representation plus a punishment term for each connected component (except the first one). For this representation of the problem, both RLS (flipping one or two bits with equal probability) and the $(1+1)$ EA find an optimal solution in any input in expected time $O(|E|^2 \log(|E|w_{\max})$, where $w_{\max}$ is the maximum weight of an edge (see [10]).

We ran the following experiments. We took as graph G the complete graph on 50 vertices (hence $|E| = 1225$) with edge weights chosen independently at random in $[0, 1]$, thus having a unique minimum spanning tree. On this instance, RLS in the variant that flips either one or two bits (random choice between these two alternatives) took $5.08 \cdot 10^6 \pm 37.75$ % iterations. Our algorithm with $r_{\max} = 5$, $\delta = 0.1$, and $\varepsilon = 1/(20,000)$ took $2.70 \cdot 10^6 \pm 36.34$ % iterations. Analyzing these runs in more detail, we observe that the preferred mutation strength r after a short initial phase takes the maximum value 5, then decreases to one, and finally goes back to two, which is then used for the large remainder of the optimization process. For reasons of computation time, we could not evaluate the $(1+1)$ EA on graphs on 50 vertices. For graphs on 20 vertices, the $(1+1)$ EA was roughly 2.7 times slower than RLS.

4 Mathematical Runtime Analysis on OneMax

In this section, we analyze via mathematical means how our algorithm optimizes OneMax. This is an asymptotic analysis in terms of the problem size n. We refer to the previous well-established runtime analysis literature for more details on the motivations of mathematical runtime analysis and on the meaning of asymptotic results, cf. [6].

The main result of this section is a proof that our algorithm with reasonable parameter settings very precisely detects the optimal mutation strength. It thus, apart from the learning iterations, has the same performance as the recently proposed randomized local search algorithm with fitness-dependent mutation strength [4]. The main technical challenge in this analysis are the dependencies between the progress of the algorithm and the learning system trying to estimate the velocities. We overcome these, among others, via a domination argument developed in [3, Lemma 1.20].

Throughout this section, we assume that $r_{\max}$ is a constant independent of n. For simplicity, we only regard the parameters $\varepsilon = n^{-0.99}$ and $\delta = n^{-0.01}$ but

[1] We report in the following the mean and relative standard deviations of our experiments by expressing, for example, the previous numbers as $45.0 \cdot 10^6 \pm 4.36$ %.

remark that broader ranges of these parameters would work as well. In addition to the notation introduced in Sect. 2, we write r_t^* for the number of bits flipped in a non-learning iteration. We also define the fitness distance $d(x) = n - f(x)$ for all $x \in \{0,1\}^n$.

For reasons of space, all proofs had to be removed from this extended abstract. They will be made available in a full journal version.

The following lemma states that, apart from an initial segment of the optimization process, the values of $w_t[r]$ can essentially assumed to be constant over time.

Lemma 1. *Let $r \in [1..r_{\max}]$, $t \geq H := (1/\varepsilon)\ln(n)$ and $w^* := \delta/r_{\max}\varepsilon$. Then with probability $1 - \exp(-n^{\Omega(1)})$, $|w_t[r] - w^*| \leq w^* O(n^{-0.002})$.*

The following two lemmas show how well our learning mechanism is able to detect the currently most profitable mutation strength. We denote in the following the progress from flipping r bits when in distance d from the optimum by $X_d^r := \max\{d(x) - d(\text{flip}(x,r)), 0\}$, where x is any search point with $d(x) = d$.

Lemma 2. *Let $r \in [1..r_{\max}]$, $t \geq 1$ and $H = (1/\varepsilon)2\ln(n)$. Then with probability at least $1 - \exp(-n^{\Omega(1)})$, we have $v_{t+H}[r] \leq (1 + O(n^{-0.002}))\max\{E[X^r_{d(x_{t+H})}], (\varepsilon/\delta)n^{0.01}\}$.*

Lemma 3. *Let $r \in [1..r_{\max}]$, $t \geq 1$ and $H = (1/\varepsilon)\ln(n)$. Assume that $E[X^r_{d(x_{t+H})}] = \Omega(n^{0.01}\varepsilon/\delta)$. Then with probability at least $1 - \exp(-n^{\Omega(1)})$, we have $v_{t+H}[r] \geq (1 - O(n^{-0.002}))E[X^r_{d(x_{t+H})}]$.*

The fact that our algorithm very precisely detects the optimal mutation strength implies that its fitness progress in each iteration is very close to the maximum possible (Theorem 1) and that it has a performance very close to the algorithm developed in [4] (Theorem 2).

Theorem 1. *Let T be the optimization time of our algorithm with parameters $\delta = n^{-0.01}$ and $\varepsilon = n^{-0.99}$ on* OneMax. *Let $T' = \min\{T, 2n\ln(n)\}$. Then with probability at least $1 - O(n^{0.19})$, for each non-learning iteration $t \in [2\ln(n)/\varepsilon, T']$, we have $E[X^{r_t^*}_{d(x_{t-1})}] \geq (1 - O(n^{-0.002}))\max\{E[X^r_{d(x_{t-1})}] \mid r \in [1..r_{\max}]\}$.*

Theorem 2. *Let $T_{r_{\max}}$ be the minimal expected runtime on the* OneMax *problem among all randomized local search algorithms with fitness-dependent mutation strength flipping at most $r_{\max}$ bits. Then the expected runtime T of our algorithm A is at most $T_{r_{\max}} + o(n)$. Consequently, by taking $r_{\max}$ large enough, our algorithm has the same expected runtime (apart from $o(n)$ terms) as the algorithm using the optimal fitness-dependent mutation strength of [4]. Also, let x_A be the current solution of our algorithm and x_{RLS} be the current solution of randomized local search after a fixed budget of $B \geq 0.2675n$ iterations. Then the expected Hamming distances to the optimum x^* satisfy $E[H(x_A, x^*)] \leq (1 + o(1))0.872E[H(x_{RLS}, x^*)]$.*

5 Parameter Choice and Design Alternatives

In this work we have proposed and analyzed a first hill climber that adaptively—based on previous fitness improvements—decides how many bits to flip in the mutation step. The required design choices were influenced by the positive experimental results and by our desire to prove with mathematical means that this algorithm tracks well the optimal mutation strength recently exhibited for OneMax. We now discuss two design variants that might prove useful for other optimization problems and give some general hints on how to choose the parameters of the algorithm.

A first observation is that our algorithm does not necessarily converge to an optimal solution. If there are local optima that can only be left by flipping more than $r_{\max}$ bits, then our algorithm has a positive probability of being stuck in such an optimum indefinitely. A simple way to overcome this is to use the mechanism proposed in this work only to redistribute the probability mass on r-bit-flips with $r \in [1..r_{\max}]$ and keep the probability distribution from standard bit mutation for the rest. In other words, in the main optimization loop in a non-learning iteration with probability $\Pr[\mathcal{B}(n,1/n) > r_{\max}]$, the algorithm determines r according to the binomial distribution $\mathcal{B}(n,1/n)$ conditional on being greater than $r_{\max}$; and it determines r from the learned velocities otherwise. This obviously ensures that the algorithm converges.

A second observation is that the highly greedy choice of r as the maximizer of the learned velocities might be too greedy for less well-behaved optimization problems in which the ideal mutation strength changes very frequently. In such situations it might be preferable to use the learned velocities only to give a mild preference to seemingly more profitable strengths. For example, in line 15 of Algorithm 2 one could choose r^+ with probability proportional to $v[r]$ and then flip r^+ bits.

A final modification that we want to propose is to use the progress experienced in any iteration (and not only the learning iterations) to update the velocities. Our update rule is designed in a way that different frequencies of the r values impose no problems. Hence in a sense, we are currently wasting the information available from the non-learning iterations. Our main motivation for doing so is that the mathematical analysis would have been more difficult, in particular, Lemma 1 would not be true with updates in each iteration. In experiments, the version with updates in each iteration usually, but not consistently, performed better.

A word on the parameters: it seems advisable to choose $r_{\max}$ small, because the learning effort is proportional to $r_{\max}$ and because in the vast majority of the iterations a small r was optimal. In our experiments, we always obtained good results with $r_{\max} = 5$, but we admit that in this first study we have not conducted an exhaustive series of experiments (also not for the other parameters). For the learning rate δ, we obtained good results with $\delta = 0.1$. It is immediately clear that $\delta(1-1/r_{\max})$ is the rate of iterations using a non-optimal mutation strength (unless two strengths are equally good), which gives some motivation to keep δ small. Of course, often a non-optimal mutation strength still has a reasonable

chance of giving progress, so these iterations often are not completely wasted (that is, only spent on learning and not on optimization). The parameter hardest to set is ε. A large value implies that we quickly forget the outcomes of previous iterations. This may allow a quick adaption to a changed environment, but also carries the risk that a rare exceptional success with a non-ideal r-value has a too large influence. In our experiments, the latter aspect was seemingly more dominant and we obtained the best results with relatively small ε-values like 0.01 or even the reciprocal of 0.1 times the expected total number of iterations.

6 Conclusion

We proposed and analyzed a simple hill climber using k-bit flips with a self-adjusting choice of the mutation strength k. This use of k-bit flips instead of the usually preferred standard bit mutation with its random mutation strength allowed to much better exploit the most effective mutation strength. At the same time, the self-adjusting choice allowed to find the optimal mutation strength automatically and on-the-fly. By this, also the risk of getting stuck in local optima, the known draw-back of k-bit flips, was overcome. We are confident that replacing standard bit mutation by k-bit flips with a self-adjusting choice of k will lead to performance gains for many optimization problem beyond the ones regarded in this work.

References

1. Bäck, T.: An overview of parameter control methods by self-adaption in evolutionary algorithms. Fundam. Inform. **35**(1–4), 51–66 (1998)
2. Böttcher, S., Doerr, B., Neumann, F.: Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN XI. LNCS, vol. 6238, pp. 1–10. Springer, Heidelberg (2010)
3. Doerr, B.: Analyzing randomized search heuristics: tools from probability theory. In: Auger, A., Doerr, B. (eds.) Theory of Randomized Search Heuristics, pp. 1–20. World Scientific Publishing, Singapore (2011)
4. Doerr, B., Doerr, C., Yang, J.: Optimal parameter choices via precise black-box analysis. In: GECCO 2016. ACM (2016, to appear)
5. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Trans. Evol. Comput. **3**, 124–141 (1999)
6. Jansen, T.: Analyzing Evolutionary Algorithms–The Computer Science Perspective. Natural Computing Series. Springer, Heidelberg (2013)
7. Jansen, T., Wegener, I.: On the analysis of a dynamic evolutionary algorithm. J. Discrete Algorithms **4**, 181–199 (2006)
8. Jansen, T., Zarges, C.: Performance analysis of randomised search heuristics operating with a fixed budget. Theor. Comput. Sci. **545**, 39–58 (2014)
9. Karafotias, G., Hoogendoorn, M., Eiben, A.: Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans. Evol. Comput. **19**, 167–187 (2015)

10. Neumann, F., Wegener, I.: Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. Theor. Comput. Sci. **378**, 32–40 (2007)
11. Oliveto, P.S., Lehre, P.K., Neumann, F.: Theoretical analysis of rank-based mutation - combining exploration and exploitation. In: CEC 2009, pp. 1455–1462. IEEE (2009)
12. Thierens, D.: Adaptive mutation rate control schemes in genetic algorithms. In: CEC 2002, pp. 980–985. IEEE (2002)