Provably Optimal Self-adjusting Step Sizes for Multi-valued Decision Variables

Benjamin Doerr[1], Carola Doerr[2(✉)], and Timo Kötzing[3]

[1] École Polytechnique, Palaiseau, France
[2] CNRS and Sorbonne Universités, UPMC Univ Paris 06, LIP6, Paris, France
carola.doerr@lip6.fr
[3] Hasso-Plattner-Institut, Potsdam, Germany

Abstract. We regard the problem of maximizing a OneMax-like function defined over an alphabet of size r. In previous work [GECCO 2016] we have investigated how three different mutation operators influence the performance of Randomized Local Search (RLS) and the (1+1) Evolutionary Algorithm. This work revealed that among these natural mutation operators none is superior to the other two for any choice of r. We have also given in [GECCO 2016] some indication that the best achievable run time for large r is $\Theta(n \log r(\log n + \log r))$, regardless of how the mutation operator is chosen, as long as it is a *static choice* (i.e., the distribution used for variation of the current individual does not change over time).

Here in this work we show that we can achieve a better performance if we allow for *adaptive* mutation operators. More precisely, we analyze the performance of RLS using a *self-adjusting mutation strength*. In this algorithm the size of the steps taken in each iteration depends on the success of previous iterations. That is, the mutation strength is increased after a successful iteration and it is decreased otherwise. We show that this idea yields an expected optimization time of $\Theta(n(\log n + \log r))$, which is optimal among all comparison-based search heuristics. This is the first time that self-adjusting parameter choices are shown to outperform static choices on a discrete multi-valued optimization problem.

Keywords: Run time analysis · Adaptive parameter choices · Mutation · Theory

1 Introduction

We combine in this work two ideas that came up quite recently in the theory of randomized search heuristics for the optimization of discrete problems: the study of multi-valued functions $f : \{0, 1, \ldots, r-1\}^n \to \mathbb{R}$ and an adaptive choice of the parameters. For the multi-valued generalization of OneMax-type functions we present a variant of Randomized Local Search (RLS) that chooses its step sizes in a self-adjusting manner. We prove that this algorithm is optimal among all comparison-based black-box optimizers. Even more, its expected optimization

J. Handl et al. (Eds.): PPSN XIV 2016, LNCS 9921, pp. 782–791, 2016.
DOI: 10.1007/978-3-319-45823-6_73

time is strictly smaller than that of any comparison-based search heuristic using static parameter choices. After the work presented in [5] this is only the second time that a self-adjusting parameter setting is proven to outperform any static choice for a discrete optimization problem, and it is the first time that this is shown for a problem over multiple decision variables.

Some background information and references for the concepts used in this work follow.

1.1 Optimization of Multi-valued OneMax Functions

Most research in discrete evolutionary computation theory regards problems that are defined over the n-dimensional Hamming cube $\{0,1\}^n$, while many experimental results exist also for other discrete search spaces (see, for example, [20] and the references therein for early examples). Only few theoretical works exist that study the extension of evolutionary algorithms and other randomized search heuristics to more general domains Ω, cf. [7] for a discussion. In [7] we considered the minimization of r-valued OneMax-type functions f_z assigning to each string $x \in \{0,1,\dots,r-1\}^n$ the sum $\sum_{i=1}^{n} d(x_i, z_i)$ of the component-wise distances to a fixed unknown string $z \in \{0,1,\dots,r-1\}^n$ (cf. Sect. 2 for detailed definitions).

We have analyzed in [7] three different ways to extend RLS and the $(1+1)$ Evolutionary Algorithm (EA) to black-box optimizers for r-valued functions. All three versions maintain the property that for RLS in each iteration the entry of exactly one position $i \in \{1,\dots,n\}$ is changed, while for the (1+1) EA an independent coin flip with success probability $1/n$ decides whether or not the entry of the i-th position is subject to change. The three variants thus differ in how entries selected for modification are updated. The uniform step operator replaces an entry by different one chosen uniform at random, while the ± 1 step operator adds or subtracts 1 from the current entry. For large r the operator with the best performance on the r-valued generalizations of OneMax is the Harmonic one which adds or subtracts to the current entry a number $j \leq r$ that is chosen with probability proportional to $1/j$. Its expected optimization time on r-valued OneMax is $\Theta(n \log r(\log n + \log r))$.

A natural question to ask is whether a better performance with respect to r can be achieved. However, from [4] we know that no *static* distribution of step sizes can achieve a better run time than $\Omega((\log r)^2)$ for $n = 1$ (see [7] for a discussion).

1.2 RLS with Self-adjusting Step Sizes

In this work we show that a better dependence on r can be achieved if we allow the step operator to change over time. More precisely, we regard the algorithm $\mathrm{RLS}_{a,b}$ which works as follows. A current search point $x \in \{0,1,\dots,r-1\}^n$ is maintained, along with a real-valued velocity vector $v \in [1, r/4]^n$ denoting the *step size* in each dimension. In each iteration, one dimension $i \leq n$ is chosen uniformly at random for variation; with probability $1/2$ the value x_i is increased by $\lfloor v_i \rfloor$, otherwise decreased by $\lfloor v_i \rfloor$, all other dimension remain as in x. If this

new search point has better fitness, the old search point is discarded and the step size v_i is increased to av_i (for some constant $a > 1$). If the new search point has worse fitness, it is discarded and the velocity v_i is decreased to bv_i (for a positive constant $b < 1$). See Algorithm 1 in Sect. 2.2 for details.

We show that, for suitable constants a and b, the expected optimization time of $\text{RLS}_{a,b}$ on the set of r-valued OneMax functions is $O(n(\log n + \log r))$, thus gaining a factor of at least $\log r$ over any RLS variant using static step sizes. This bound is provably optimal among all comparison-based algorithms. That is, no black-box algorithm can achieve a better performance on r-valued OneMax functions unless it explicitly exploits absolute fitness-values.

1.3 Self-adjusting Parameter Choices

One easily observes that in continuous optimization static parameter choices are not very meaningful. This is why for such problems several examples exist where adaptive parameter choices are well understood also from a theoretical perspective (for example, the works [2,14,15] analyze the convergence rates of different evolution strategies). As has been noted in [7], however, such results are difficult to compare to performance guarantees in discrete optimization let alone being transferable to such problems. This is mostly due to the fact that in discrete optimization we do not study the speed of convergence but the time needed to hit an optimal solution. But even if one studies continuous optimization with an a-priori fixed target precision (see [16] and the references therein), then typically the norms used to evaluate a solution differ from the typically regarded 1-norm used in discrete optimization.

For the discrete domain, several empirical works exist that suggest an advantage of adaptive parameter updates (cf. [12], [13, Chap. 8], and [17] for surveys). However, the first work formally showing an asymptotic gain over static parameter selection is the self-adjusting choice of the population size of the $(1+(\lambda,\lambda))$ GA proposed and analyzed in [5]. In that work the advantage is shown for the classic OneMax functions $f_z : \{0,1\}^n \to \mathbb{R}, x \mapsto |\{1 \leq i \leq n \mid x_i = z_i\}|$. Our result is hence the first of its type for a multi-valued search problem in the discrete domain.

Also when we include in our consideration other adaptive parameter choices[1] only few situations exist for which an advantage over static parameter choices could be proven. All these works study the optimization of pseudo-Boolean functions $f : \{0,1\}^n \to \mathbb{R}$. To be more precise, the only theoretical investigations of adaptive parameter choices in discrete optimization that we are aware of

[1] Following the terminology introduced in [5, Sect. 3.1] we distinguish between functionally-dependent and self-adjusting parameter choices. While *functionally-dependent* parameter choices depend only on the current state of the algorithm, they may explicitly use absolute fitness values. Fitness-dependent mutation rates are a typical example for such *functionally-dependent* parameter choices. *Self-adjusting* parameter choices, in contrast, do not depend on absolute fitness information but rather on the success of previous iterations. This is the case of the parameter updates of the $\text{RLS}_{a,b}$ considered in this work.

analyze advantages of a fitness-dependent mutation rate for the (1+1) EA optimizing LeadingOnes [3] and for RLS optimizing OneMax [8], a self-adjusting choice of the number of parallel evaluations in a parallel EA [19] as well as a fitness-dependent [6] and the above-mentioned self-adjusting [5] choice of the population size for the $(1 + (\lambda, \lambda))$ GA.

We believe that self-adjusting parameter choices provide a possibility for significant improvement of many search heuristics, and theoretical analyses can offer guidance for how to design such self-adjustment mechanisms. Our work shows that our mathematical toolbox, in particular drift analysis, is well-suited to analyze such systems.

2 Preliminaries

For any positive integer n we set $[n] := \{1, 2, \dots, n\}$ and $[0..n] := \{0\} \cup [n]$. We regard in this work r-valued functions over strings of length n, i.e., functions $f : [0..r-1]^n \to \mathbb{R}$. The value of r may or may not depend on n, and it may or may not be smaller or larger than n.

We briefly define below the problem setting and the self-adjusting version of RLS that we aim at analyzing.

2.1 Multi-valued OneMax Problems

As in [7] we regard two classes of r-valued OneMax functions. These classes are the collection of functions $f_z : [0..r-1]^n \to \mathbb{R}; x \mapsto \sum_{i=1}^{n} d(x_i, z_i)$, $z \in [0..r-1]^n$. They differ in the metric d used to evaluate the *distance* of x_i to z_i. The first metric, which we call the *interval-metric* d_{int}, is the usual metric on the integers, i.e.,

$$d_{\text{int}}(a, b) := |b - a|.$$

Note that in the interval-metric the fitness landscapes of the r-valued OneMax functions are not isomorphic to each other. This can be easily seen, for example, by comparing $f_{(0,\dots,0)}$ with $f_{(r/2,\dots,r/2)}$ which has a much more symmetric fitness landscape. Note that, for the boundary handling we employ in this paper (see Sect. 2.2), our results are unaffected by the exact choice of r-valued OneMax function. This is why we also consider a second metric, which we call the *ring-metric* d_{ring}. This metric connects the two endpoints of the interval $[0..r-1]$ such that it forms a ring, i.e.,

$$d_{\text{ring}}(a, b) := \min\{|b - a|, |b - a + r|, |b - a - r|\}.$$

Unlike the name OneMax suggests, we regard in this work the *minimization* of the r-valued OneMax functions. It is easily seen that, regardless of the metric in place, the unique global optimum of f_z is thus the string z. We call z the *target vector* of f_z.

2.2 RLS with Self-adjusting Mutation Strength

We investigate the following natural generalization of RLS to a multi-valued algorithm $\mathrm{RLS}_{a,b}$ with a self-adjusting mutation strength whose update rules are parametrized by the constants $1 < a \leq 2$ and $1/2 < b < 1$. The algorithm is summarized in Algorithm 1.

Algorithm 1. $\mathrm{RLS}_{a,b}$ with self-adjusting step sizes maximizing a function $f : [0..r-1]^n \to \mathbb{R}$

```
Initialization: Let v ∈ [1, ⌊r/4⌋]^n uniformly at random;
Sample x ∈ [0..r − 1]^n uniformly at random and query f(x);
Optimization: for t = 1, 2, 3, ... do
    Choose i ∈ [n] uniformly at random;
    for j = 1, ..., n do
        if j = i then with probability 1/2 let y_j ← x_j − ⌊v_j⌋ and let
        y_j ← x_j + ⌊v_j⌋ otherwise
        else y_j ← x_j
    Query f(y);
    if f(y) < f(x) then v_i ← min{a v_i, ⌊r/4⌋} else v_i ← max{1, b v_i}
    if f(y) ≤ f(x) then x ← y
```

$\mathrm{RLS}_{a,b}$ maintains a search point $x \in [0..r-1]^n$ as well as a real-valued *velocity vector* $v \in [1, \lfloor r/4 \rfloor]^n$; we use real values for the velocity to circumvent rounding problems. Both these strings are initialized uniformly at random, but it is not difficult to verify that all results shown in this paper apply to any arbitrary initialization of x and v. In one iteration of the algorithm a position $i \in [n]$ is chosen uniformly at random. The entry x_i is replaced by $x_i - \lfloor v_i \rfloor$ with probability $1/2$ and by $x_i + \lfloor v_i \rfloor$ otherwise (see below for how to deal with overstepping the endpoints of the interval $[0, r-1]$). The entries in positions $j \neq i$ are not subject to mutation. The resulting string y replaces x if its fitness is at least as good as the one of x, i.e., if $f(y) \leq f(x)$ holds (recall that we regard the minimization of f). If the offspring y is strictly better than its parent x, i.e., if $f(y) < f(x)$, we increase the velocity v_i in the i-th component by multiplying it with the constant a and we decrease v_i to bv_i otherwise. The algorithm proceeds this way until we decide to stop it. Since we regard in this work the time needed until $\mathrm{RLS}_{a,b}$ evaluates for the first time an optimal solution (this random variable is called the *run time* of Algorithm 1), we do not specify any stopping criterion here.

We will now discuss some technical details.

It may happen that $x_i - \lfloor v_i \rfloor < 0$ or $x_i + \lfloor v_i \rfloor > r - 1$. If we are working with the interval-metric then we assume that the algorithm does not change its current position, that is, the offspring is discarded and the velocity is not adjusted (decreasing the velocity in this case would lead to the same results). In the ring-metric we identify all values *modulo* r, i.e., we identify values $p < 0$ with $p + r$ and values $p > r - 1$ with $p - r$. Note that in the ring-metric it can

happen that we decrease the fitness regardless of whether we add or subtract from x_i the value $\lfloor v_i \rfloor$. This in particular applies when x_i is close to $r/2$.

Furthermore, we emphasize that the velocity vector is an element in the real interval $[1, \lfloor r/4 \rfloor]$, that is, it does not necessarily take integer values. This technicality avoids that rounding inaccuracies accumulate over several velocity adaptations. The velocity is capped at 1 (to avoid situations in which we do not move at all) and at $\lfloor r/4 \rfloor$ (to avoid too large jumps).

To further lighten the notation, we say that the algorithm *"moves in the right direction"* or *"towards the target value"* if the distance to the target is actually decreased by $\lfloor v_i \rfloor$. Analogously, we speak otherwise of a step *"away from the target"* or *"in the wrong direction"*.

2.3 Drift Analysis

The idea of drift analysis is to map the optimization process to a series of real-valued random variables that measure, in a suitable way, the expected progress that the algorithm achieves in one iteration. The hope is to show that this expected progress systematically depends on the current state of the algorithm, for example, in an additive or a multiplicative way. Drift theorems then help to convert the expected progress made in one iteration to bounds on the time needed to hit a certain goal such as identifying an optimal search point; cf. [10,18] for a more detailed discussion of drift theory.

In the context of $\text{RLS}_{a,b}$ the state of the algorithm can be described by the pair (x, v) consisting of the current search point $x \in [0..r-1]^n$ and the current velocity vector $v \in [1, \lfloor r/4 \rfloor]^n$. We will design in Sect. 3 a potential function g that maps these states to real numbers in a way that the expected progress of one iteration of $\text{RLS}_{a,b}$ depends on the current potential $g(x, v)$ in a multiplicative way. That is, for y and v' denoting the resulting search point and velocity vector after one iteration of $\text{RLS}_{a,b}$, we will show that $E(g(x, v) - g(y, v')) \geq \delta g(x, v)$ for some positive constant δ. The following drift theorem will then allow us to derive bounds on the expected run time of $\text{RLS}_{a,b}$ on any r-valued OneMax function. This *multiplicative drift theorem* had first been introduced to the theory of randomized search heuristics in [10]. A more direct proof of this results, that also gives large deviation bounds, can be found in [9]. The variables $X^{(t)}$ in the statement correspond to the state $g(x, v)$ of the algorithm after t iterations.

Theorem 1 (from [10]). *Let $X^{(0)}, X^{(1)}, \ldots$ be a random process taking values in $S := \{0\} \cup [s_{\min}, \infty) \subseteq \mathbb{R}$. Assume that $X^{(0)} = s_0$ with probability one. Assume that there is a $\delta > 0$ such that for all $t \geq 0$ and all $s \in S$ with $\Pr[X^{(t)} = s] > 0$ we have $E[X^{(t+1)} | X^{(t)} = s] \leq (1 - \delta)s$. Then $T := \min\{t \geq 0 \mid X^{(t)} = 0\}$ satisfies $E[T] \leq \frac{\ln(s_0/s_{\min})+1}{\delta}$.*

3 Main Result

In this section we sketch the proof of the following statement (the full proof does not fit the available space).

Theorem 2. *For constants a, b satisfying $1 < a \leq 2$, $1/2 < b \leq 0.9$, $2ab-b-a > 0$, $a+b > 2$, and $a^2 b > 1$ (one can choose, for example, $a = 1.7$ and $b = 0.9$) the expected run time of $RLS_{a,b}$ (Algorithm 1) on any generalized r-valued* OneMax *function is $\Theta(n(\log n + \log r))$ and this is optimal among all comparison-based algorithms.*

The lower bound as well as the statement that no comparison-based algorithm can have an expected run time of smaller order easily follows from a coupon collector argument and the information-theoretic lower bound. In a bit more detail, we note that in the initial solution there are, with high probability, $\Theta(n)$ positions i in which the value x_i does not agree with that of the target string. The algorithm has to touch each of these positions at least once, which by the well-known coupon collector theorem (cf. [1, Sect. 1] for an introduction to this problem) requires $\Theta(n \log n)$ iterations on average and with high probability. The $\Omega(n \log r)$ follows from the observation that there are r^n possible target strings in total. Since $\text{RLS}_{a,b}$ exploits only the information whether or not the offspring has a fitness value that is at least as good as that of its parent (in the decision of whether or not to replace the parent) and whether or not its fitness is strictly better (in the decision how to update the velocity), it is a *comparison-based algorithm* that uses only $\log_2(3)$ bits of information per iteration. As such it therefore needs $\Omega(\log(r^n)) = \Omega(n \log r)$ iterations in expectation to optimize any unknown r-valued OneMax function. See [11] for how to turn the latter information-theoretic consideration into a formal proof.

To prove the upper bound we use *drift analysis*; multiplicative drift analysis to be more precise. To this end, as explained in Sect. 2.3, we need to find a mapping of the state (x, v) of the algorithm to a real value. This *potential function* should measure some sort of distance to the target state. We briefly discuss this potential function below. Proving that it yields the required multiplicative drift is the purpose of Lemma 3.

To simplify the notation below, for a given search point x and the target bit string z and the chosen metric d, we let $d_i = d(x_i, z_i)$ (for all $i \leq n$) be the distance vector of x to z. Thus, the goal is to reach a state in which the distance vector is $(0, \ldots, 0)$. We now want to define a potential function in dependence on (d, v) (where of course d is dependent on x) such that it is 0 when d is $(0, \ldots, 0)$ and strictly positive for any $x \neq (0, \ldots, 0)$. Furthermore, we easily see that there are two important ways to make progress, either by advancing in terms of fitness or by adjusting the velocity to a value that is more suitable to make progress in future iterations. This has to be reflected in the potential function. Our ultimate goal being the minimization of fitness, it is not difficult to see that some preference should be given to a progress in fitness. This can be achieved by multiplying the term accounting for the appropriateness of the velocity with some constant $c < 1$. We measure the appropriateness of the velocity as the maximum of the ratios $d_i/(2v_i)$ and $2v_i/d_i$, reflecting the fact that a velocity of $d_i/2$ is very well-suited for progress; smaller values give less progress, while larger values lead to a badly adjusted velocity in the next iteration (and very large values make progress in fitness impossible).

One problem in getting good drift is that velocities v_i just below $2d_i$ allow for jumping over the target while increasing the (already too large) velocity. We get around this problem by observing that it is equally likely that the large velocity is reduced because of a jump in the wrong direction, and then, while still larger than d_i, will still give a good improvement when overstepping the goal. We reflect this in the potential function by giving a penalty term of pd_i (for some suitable constant p) on any state (d, v) having a too large velocity.

To sum up this discussion we use as potential function the following map $g : [0..r-1]^n \times [1, \lfloor r/4 \rfloor]^n \to \mathbb{R}, (x, v) \mapsto \sum_{i=1}^n g_i(d_i, v_i)$ where $g_i(d_i, v_i) := 0$ for $d_i = 0$ and for $d_i \geq 1$

$$g_i(d_i, v_i) := d_i + \begin{cases} cd_i \max\{2v_i/d_i, d_i/(2v_i)\}, & \text{if } v_i \leq 2bd_i; \\ cd_i \max\{2v_i/d_i, d_i/(2v_i)\} + pd_i, & \text{otherwise} \end{cases} \tag{1}$$

and c, p are (small) constants specified below.

Summarizing all the conditions needed below, we require that the constants a, b, c, p satisfy $1 < a \leq 2$, $1/2 < b \leq 0.9$, $2ab - b - a > 0$, $a + b > 2$, $a^2 b > 1$, $8abc + 2p + 4c/b \leq 1/16$, $p > 8c\left(\frac{a+b}{2} - 1\right)$, and $p > 4(a-1)c > 0$.

We can thus choose, for example, $a = 1.7$, $b = 0.9$, $p = 0.01$, and $c = 0.001$.

The following lemma, together with the observation that the initial potential is of order at most nr^2 plugged into the multiplicative drift theorem (Theorem 1) proves the desired overall expected run time of $O(n \log(nr))$.

Lemma 3. *Let $d \neq (0, \ldots, 0)$ and $v \in [1, \lfloor r/4 \rfloor]^n$. Let (d', v') be the state of Algorithm 1 started in (d, v) after one iteration (i.e., after a possible update of x and v). The expected difference in potential satisfies*

$$E\left(g(d, v) - g(d', v') \mid d, v\right) \geq \frac{\delta}{n} g(d, v)$$

for some positive constant δ.

4 Conclusions

While in [7] we analyzed static mutation operators for optimizing multi-valued functions $f : [0..r-1]^n \to \mathbb{R}$, in this paper we gave an operator based on self-adjusting step sizes. We proved that, in the case of RLS, this leads to a provably optimal run time for r-valued OneMax functions.

Already for the analysis of RLS we gave an intricate drift-argument, with many different cases to consider and penalty terms for resolving situations which would otherwise allow for search points with negative drift. Extending our results to the case of the (1+1) EA might thus be a very challenging task, pushing the limits of drift theory.

Note that we chose a specific step size adaptation scheme which guarantees optimal run time. It would also be interesting to investigate other adaptation schemes. For example, the step size, in each iteration, could be drawn from a

distribution (just as in one of the operators presented in [7]), and the parameters of this distribution are adapted.

Another issue with step sizes is that infeasible areas of the search space might be reached (in our setting this can happen if we use the interval metric). The issue of boundary handling is a known problem, and our boundary handling technique is by no means the only way for dealing with it. We believe that our choice is natural and leads to a "fair" treatment of all parts of the search space, and it leads to an optimal run time for our setting. It might be interesting to see whether there are other settings where a different boundary handling is more natural, or gives better run time.

Acknowledgments. This research benefited from the support of the "FMJH Program Gaspard Monge in optimization and operation research", and from the support to this program from EDF (Électricité de France).

References

1. Auger, A., Doerr, B.: Theory of Randomized Search Heuristics. World Scientific, Singapore (2011)
2. Auger, A., Hansen, N.: Linear convergence on positively homogeneous functions of a comparison based step-size adaptive randomized search: the (1+1) ES with generalized one-fifth success rule. CoRR, abs/1310.8397 (2013). http://arxiv.org/abs/1310.8397
3. 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)
4. Dietzfelbinger, M., Rowe, J.E., Wegener, I., Woelfel, P.: Tight bounds for blind search on the integers and the reals. Comb. Probab. Comput. **19**, 711–728 (2010)
5. Doerr, B., Doerr, C.: Optimal parameter choices through self-adjustment: applying the 1/5-th rule in discrete settings. In: Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO 2015), pp. 1335–1342. ACM (2015)
6. Doerr, B., Doerr, C., Ebel, F.: From black-box complexity to designing newgenetic algorithms. Theor. Comput. Sci. **567**, 87–104 (2015)
7. Doerr, B., Doerr, C., Kötzing, T.: The right mutation strength for multi-valued decision variables. In: Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO 2016). ACM (2016, to appear). http://arxiv.org/abs/1604.03277
8. Doerr, B., Doerr, C., Yang, J.: Optimal parameter choices via precise black-box analysis. In: Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO 2016). ACM (2016, to appear)
9. Doerr, B., Goldberg, L.A.: Adaptive drift analysis. Algorithmica **65**, 224–250 (2013)
10. Doerr, B., Johannsen, D., Winzen, C.: Multiplicative drift analysis. Algorithmica **64**, 673–697 (2012)
11. Droste, S., Jansen, T., Wegener, I.: Upper and lower bounds for randomized search heuristics in black-box optimization. Theor. Comput. Syst. **39**, 525–544 (2006)
12. Eiben, A.E., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary. IEEE Trans. Evol. Comput. **3**, 124–141 (1999)

13. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Heidelberg (2003)
14. Hansen, N., Gawelczyk, A., Ostermeier, A.: Sizing the population with respect to the local progress in (1,λ)-evolution strategies - a theoretical analysis. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1995), pp. 80–85. IEEE (1995)
15. Jägersküpper, J.: Rigorous runtime analysis of the (1+1) ES: 1/5-rule and ellipsoidal fitness landscapes. In: Wright, A.H., Vose, M.D., De Jong, K.A., Schmitt, L.M. (eds.) FOGA 2005. LNCS, vol. 3469, pp. 260–281. Springer, Heidelberg (2005)
16. Jägersküpper, J.: Oblivious randomized direct search for real-parameter optimization. In: Halperin, D., Mehlhorn, K. (eds.) ESA 2008. LNCS, vol. 5193, pp. 553–564. Springer, Heidelberg (2008)
17. Karafotias, G., Hoogendoorn, M., Eiben, A.: Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans. Evol. Comput. **19**, 167–187 (2015)
18. Kötzing, T.: Concentration of first hitting times under additive drift. Algorithmica **75**, 490–506 (2016)
19. Lässig, J., Sudholt, D.: Adaptive population models for offspring populations and parallel evolutionary algorithms. In: Proceedings of the ACM Workshop on Foundations of Genetic Algorithms (FOGA 2011), pp. 181–192. ACM (2011)
20. Rudolph, G.: An evolutionary algorithm for integer programming. In: Davidor, Y., Schwefel, H.-P., Mönner, R. (eds.) (PPSN 1994). LNCS, pp. 139–148. Springer, Heidelberg (1994)