Towards a Theory-Guided Benchmarking Suite for Discrete Black-Box Optimization Heuristics: Profiling $(1+\lambda)$ EA Variants on OneMax and LeadingOnes

Carola Doerr[1], Furong Ye[2], Sander van Rijn[2], Hao Wang[2], and Thomas Bäck[2]

[1]*Sorbonne Université, CNRS, LIP6, Paris, France*
[2]*LIACS, Leiden University, Leiden, The Netherlands*

July 17, 2018

Abstract

Theoretical and empirical research on evolutionary computation methods complement each other by providing two fundamentally different approaches towards a better understanding of black-box optimization heuristics. In discrete optimization, both streams developed rather independently of each other, but we observe today an increasing interest in reconciling these two sub-branches. In continuous optimization, the COCO (COmparing Continuous Optimisers) benchmarking suite has established itself as an important platform that theoreticians and practitioners use to exchange research ideas and questions. No widely accepted equivalent exists in the research domain of discrete black-box optimization.

Marking an important step towards filling this gap, we adjust the COCO software to pseudo-Boolean optimization problems, and obtain from this a benchmarking environment that allows a fine-grained empirical analysis of discrete black-box heuristics. In this documentation we demonstrate how this test bed can be used to profile the performance of evolutionary algorithms. More concretely, we study the optimization behavior of several $(1+\lambda)$ EA variants on the two benchmark problems OneMax and LeadingOnes. This comparison motivates a refined analysis for the optimization time of the $(1+\lambda)$ EA on LeadingOnes.

1 Introduction

After a long phase of around 20 years in which mathematical and empirical research on evolutionary algorithms (EAs) seemed to have advanced more or less independently of each other, we observe today a steadily growing interest to reconcile these two different methodologies. This interest is spurred by the awareness that certain advantages of one approach cannot be compensated for by the other: While theoretical work has the advantage of universal bounds that are proven with mathematical rigor, empirical research contributes concrete numbers (as opposed to the predominant asymptotic bounds derived by theoretical approaches) and can provide results for complex problems or algorithms that do not admit a stringent mathematical analysis. This way, both research methods offer a different view on the working principles of evolutionary computation (EC) methods. Bringing these two streams closer together has the potential to create a much more complete picture than what any of the two approaches can deliver individually.

In continuous optimization, a strong promoter for joint projects and discussion is the COCO (COmparing Continuous Optimisers) platform, which offers a well-designed and widely accepted benchmarking environment for derivative-free continuous black-box optimizers. The affiliated BBOB workshops have established themselves as a prime platform for theoreticians and practitioners to exchange novel ideas and research questions. For discrete black-box optimization, unfortunately, no initiative of equivalent reputation exists.

With this work we wish to make an important step towards establishing a well-designed benchmarking environment for pseudo-Boolean black-box optimization. To this end, we build on the freely available COCO software [13] (cf. [12] for a documentation of this framework) and adapt it to discrete benchmark problems. We summarize in this work how this framework can be used to generate novel and—as we shall see—sometimes quite surprising findings that, in turn, inspire new research directions.

To demonstrate the functioning of the benchmarking platform, we investigate in this work the performance of different $(1+\lambda)$ EA variants. These variants differ in their parameter settings, i.e., in the offspring population size λ and the mutation rates p used by the variation operator. Apart from $(1+\lambda)$ EA variants with static parameter settings, we also investigate the performance profiles of four different algorithms with self-adjusting parameter choices.

To provide a comparison that is meaningful for theoreticians and practitioners alike, we respond to a recent critique of the way EAs are typically evaluated in the theory of EC literature [6] and regard in our comparison $(1+\lambda)$ EA variants that do not create offspring that are identical to their parents. For our static and noiseless benchmark problems, such offspring would not provide any new information to the algorithm, and since a plus selection scheme is in place, they do also not bring any other benefit to the optimization process. In our $(1+\lambda)$ EA$_{>0}$ variants, we therefore ensure that every offspring differs from the parent in at least one bit.

1.1 The Role of Benchmarking in EC

Benchmarking aims at supporting practitioners in choosing appropriate algorithms for a given optimization problem through a systematic empirical comparison of different algorithmic techniques. Typical research questions that can be addressed by a benchmarking approach concern

- the *scalability* of performance in terms of problem dimension or decision variables,
- the optimization behavior across different *benchmark functions*,
- the dependence of performance on selected *problem and instance features*, and
- the *sensitivity* of a heuristic with respect to small changes of its parameters.

For theoreticians, benchmarking can be an essential tool for the enhancement of mathematically-derived ideas into techniques that can be beneficial for a broad range of problems. It can also be used to identify reasonable parameter settings for which performance bounds can be proven. Benchmarking is also an important tool to communicate mathematical research results, as it can be more appealing to look at performance plots than to compare mathematical formulae. Finally, benchmarking can also serve as a source for interesting research questions. We present such an example in Section 4.3, where we prove refined runtime bounds for the $(1+\lambda)$ EA on the LeadingOnes problem. This result is clearly motivated by the observation that the $(1+1)$ EA$_{>0}$ and the $(1+50)$ EA$_{>0}$ show a very similar performance in our empirical comparisons.

1.2 Performance Criteria

Most theoretical works use *total expected optimization time*, i.e., the average number of function evaluations needed to identify an optimal solution, as the only performance criterion. This measure is very coarse, as it reduces the whole optimization process to a single number. It has therefore been suggested in [19] to complement optimization time with a more fine-grained performance measure. Jansen and Zarges [19] suggested to adapt a *fixed-budget perspective,* and to measure the expected quality of a solution that can be obtained within a given computational budget. A complementary performance measure, which is more commonly used for algorithm comparison, are *fixed-target results,* which measure the expected number of function evaluations needed to identify solutions of a certain target quality. This perspective extends optimization time to arbitrary target values. Our benchmarking tool, just like COCO, provides both these performance measures.

1.3 Comment on the Benchmark Problems

At the current state, our benchmarking environment is set up for the comparison of black-box optimizers on a few selected benchmark problems, such as linear functions, functions with a plateau (so-called jump-functions), or the non-separable LeadingOnes functions. This selection is clearly biased towards functions that are analyzable by mathematical means. We will extend this selection significantly in the near future.

We have chosen to present in this work a comparison of performance profiles for the two functions OneMax and LeadingOnes. For these two problems a number of theoretical results exist, and one may have the feeling that these problems are quite well understood already. Our empirical comparison shows nevertheless some surprising effects, which have inspired us to prove a more precise runtime bound for the performance of the $(1+\lambda)$ EA on LeadingOnes. The empirical performance profiles also raise a number of other interesting questions that may be useful to address by a mathematical approach.

We recall that theoreticians regard simple benchmark functions like OneMax and LeadingOnes in the hope that, among other reasons,

- they give insights into how the studied algorithms perform on the easier parts of a difficult optimization problem,
- in order to understand some basic working principles of the algorithms, which can then be used for the analysis of more complex problems, more complex

algorithms, and for the development of new algorithmic ideas,

- the theoretical investigations (which even for seemingly simple algorithms and problems can be surprisingly complex) triggers the development of new analytical tools that can be used for more complex tasks, and
- very precise mathematical statements can be obtained, which allow to determine, for example, how the chosen parameter values influence the performance of an algorithm, or how these parameters can be controlled in an optimal way.

We furthermore note that even for OneMax and LeadingOnes, despite being studied since the very early days of theory of evolutionary computation, several unsolved problems exist, many of which are of seemingly simple nature such as the optimal dynamic mutation rate of the $(1+1)$ EA for OneMax. We are confident that our benchmarking environment will be a key enabler to strengthen existing results, to guide researchers towards interesting research questions, and, ultimately, towards high-performing optimization techniques.

2 Summary of Algorithms

Two fundamental building blocks of evolutionary algorithms are global variation operators and populations. *Global variation operators* are sampling strategies that are characterized by the property that every possible solution candidate has a positive probability of being sampled within a short time window, regardless of the current state of the algorithm. Standard bit mutation is an example of a global mutation operator. From a given input string $x \in \{0,1\}^n$, standard bit mutation creates an offspring by flipping each bit in x with some positive probability $0 < p < 1$, with independent decisions for each bit. For any x, y the probability to sample y from x is thus $p^{H(x,y)}(1-p)^{n-H(x,y)}$, where $H(x,y)$ is the number of bits in which x and y differ (*Hamming distance*). This probability is positive even for search points that are very far apart. The motivation to use global sampling strategies is to overcome local optima by eventually performing a sufficiently large jump.

Storing information about the optimization process, maintaining a diverse set of reasonably good solutions, and gathering a more complete picture about the structure of the problem at hand are among the most important reasons to employ *population-based EAs.* The first two objectives are served by the *parent population*, which is the subset of previously evaluated search points that are kept in the memory of the algorithm. The parent population is updated after each generation. New solution candidates are sampled from it through the use of variation operators. These points form the *offspring population* of the generation. Non-trivial offspring population sizes address the desire to gather more information about the fitness landscape before making any decision about which of the points from the parent and offspring population to keep in the memory for the next iteration.

It is very well understood that both the size of the parent population as well as the size of the offspring population can have a significant impact on the performance. Finding suitable parameter values for these two quantities remains to be a challenging problem in practical applications of EAs. From an analytical point of view, populations increase the complexity of the optimization process considerably, as they introduce a lot of dependencies that need to be taken care of in the mathematical analysis. It is therefore not surprising that

Algorithm 1: flip$_\ell$ chooses ℓ different positions and flips the entries in these positions.

```
Input: $x = (x_1 \dots x_n) \in \{0,1\}^n$, $\ell \in \mathbb{N}$;
$y \leftarrow x$;
Select $\ell$ pairwise different positions $i_1, \dots, i_\ell \in [n]$ u.a.r.;
for $j = 1, ..., \ell$ do $y_{i_j} \leftarrow 1 - x_{i_j}$;
```

only few theoretical works on population-based EAs exist, cf. [15] and references mentioned therein. Most existing theoretical works regard algorithms with non-trivial *offspring population* sizes, while the impact of the *parent population* size has received much less interest.

Since one of the goals of this work is to showcase how benchmarking can serve as a platform for theoreticians and practitioners to discuss new research directions, we present below empirical and theoretical results for the $(1+\lambda)$ EA, the arguably simplest EA that combines a global sampling technique with a non-trivial offspring population size. The $(1+\lambda)$ EA and its various variants analyzed in Sections 3 and 4 are formally defined below. Existing theoretical results for the selected benchmark problems will be presented in the respective sections.

Notation. Throughout this work, the problem dimension is denoted by n. As common in evolutionary computation, we assume that the algorithms know the dimension of the problem that they are optimizing. We further assume that we aim to *maximize* the objective function. For every positive integer r, we abbreviate by $[r]$ the set $\{1, 2, \dots, r\}$ and we set $[0..r] := \{0\} \cup [r]$. By ln we denote the natural logarithm to base $e := \exp(1)$.

2.1 Motivation for the Modifications

As noted in [6] there exists an important discrepancy between the algorithms classically regarded in the theory of evolutionary computation literature and their common implementations in practice. For mutation-based algorithms like $(\mu+\lambda)$ and (μ, λ) EAs, this discrepancy concerns the way new solution candidates are sampled from previously evaluated ones, and how the function evaluations are counted. Both algorithms use the above-described standard bit mutation as only variation operator. An often recommended value for the mutation rate p is $1/n$, which corresponds to flipping exactly one bit on average, an often desirable behavior when the search converges.

When implementing standard bit mutation, it would be rather inefficient to decide for each $i \in [n]$ whether or not the i-th bit of x should be flipped. Luckily, this is not needed, as we can simply observe that standard bit mutation can be equally expressed as drawing a random number ℓ from the binomial distribution $\mathrm{Bin}(n, p)$ with n trials and success probability p and then flipping ℓ bits that are sampled from $[n]$ uniformly at random (u.a.r.) and without replacement. This latter operation is formalized by the flip$_\ell$ operator in Algorithm 1. We refer to ℓ as the *mutation strength* or the *step size*, while we call p the *mutation rate*.

Analyzing standard bit mutation, we easily observe that the probability to not flip any bit at all equals $(1-p)^n$, which for $p = 1/n$ converges to $1/\exp(1)$. That is, in about 36.8% of calls to this operator, a copy of the input is returned.

For the $(1+\lambda)$ EA there is no benefit of evaluating such a copy (unless facing a dynamic or noisy optimization setting), since it applies *plus selection*, where both the parent as well as the offspring can be selected to "survive" for the next generation. It is therefore advisable to change the probability distribution from which the mutation strength ℓ is sampled. A straightforward (and commonly used) idea is to simply re-sample ℓ from $\text{Bin}(n,p)$ until a non-zero value is returned. This approach corresponds to distributing the probability mass $(1-p)^n$ of sampling a zero proportionally to all step sizes $\ell > 0$. This gives the conditional binomial distribution $\text{Bin}_{>0}(n,p)$, which assigns to each $\ell \in \mathbb{N}$ a probability of $\binom{n}{\ell}p^\ell(1-p)^{n-\ell}/(1-(1-p)^n)$. All our empirical results use this conditional sampling strategy. The results can therefore differ significantly from figures previously published in the theory of EA literature.[1]

2.2 The Basic $(1+\lambda)$ EA$_{>0}$

The $(1+\lambda)$ EA samples λ offspring in every iteration, from which only the best one survives (ties broken uniformly at random). Each offspring is created by standard bit mutation. Following our discussion above, we make use of the re-sampling strategy described above, and obtain the $(1+\lambda)$ EA$_{>0}$, which we summarize in Algorithm 2.

Algorithm 2: The $(1+\lambda)$ EA$_{>0}$ with mutation rate $p \in (0,1)$ for the maximization of $f : \{0,1\}^n \to \mathbb{R}$

```
Initialization: Sample $x \in \{0,1\}^n$ u.a.r.;
Optimization: for $t = 1, 2, 3, \ldots$ do
    for $i = 1, \ldots, \lambda$ do
        Sample $\ell^{(i)}$ from $\text{Bin}_{>0}(n,p)$;
        $y^{(i)} \leftarrow \text{flip}_{\ell^{(i)}}(x)$;
    Sample $x$ from $\arg\max\{f(x), f(y^{(1)}), \ldots, f(y^{(\lambda)})\}$ u.a.r.;
```

2.3 Adaptive $(1+\lambda)$ EA$_{>0}$ Variants

The $(1+\lambda)$ EA$_{>0}$ has two parameters: the offspring population size λ and the mutation rate p. Common implementations of the $(1+\lambda)$ EA$_{>0}$ use the same population size λ and the same mutation rate p throughout the whole optimization process (*static parameter choice*), while the use of *dynamic parameter values* is much less established. A few works exist, nevertheless, that propose to *control* the parameters of the $(1+\lambda)$ EA online [7]. We focus in our empirical comparison on algorithms that have a mathematical support. These are summarized in the following two subsections.

2.3.1 Adaptive Mutation Rates

One of the few works that experiments with a *non-static mutation rate* for the $(1+\lambda)$ EA was presented at GECCO 2017 [2]. The there-suggested al-

[1] We note that the suggestion to adapt the performance evaluation has been made several times in the literature, e.g., [9, 18, 6], but has not been picked up systematically.

gorithm stores a parameter r that is adjusted online. In each iteration, the $(1+\lambda)$ $\text{EA}_{r/2,2r}$ creates $\lambda/2$ offspring by standard bit mutation with mutation rate $r/(2n)$, and it creates $\lambda/2$ offspring with mutation rate $2r/n$. The value of r is updated after each iteration. With probability $1/2$ it is set to the value that the best offspring individual of the last iteration has been created with (ties broken at random), and it is replaced by either $r/2$ or $2r$ otherwise (unbiased random decision). Finally, the value r is capped at 2 if smaller, and at $n/4$, if it exceeds this value. In our experiments, we use $r=2$ as initial value.

In [2] it is shown analytically that the $(1+\lambda)$ $\text{EA}_{r/2,2r}$ yields an asymptotically optimal runtime on OneMax. This performance is strictly better than what any static mutation rate can achieve, cf. Section 3.1. How well the adaptive scheme works for other benchmark problems is left as an open question in [2].

2.3.2 Adaptive Population Sizes

Apart from the mutation rate, one can also consider to *adjust the offspring population size* λ. This is a much more prominent problem, because λ is an explicit parameter, while the mutation rate is often not specified (and thus by default assumed to be $1/n$).

In the theory of EC literature, the following three success-based update rules have been studied. In [20] the offspring population size λ is initialized as one. After each iteration, we count the number s of offspring that are at least as good as the parent. When $s=0$, we double the population size, and we replace it by $\lfloor \lambda/s \rfloor$ otherwise. For brevity, we call this algorithm the $(1+\{2\lambda,\lambda/s\})$ EA, and its resampling variant the $(1+\{2\lambda,\lambda/s\})$ $\text{EA}_{>0}$.

Two similar schemes were studied in [11] where λ is doubled if no strictly better search point has been identified and either set to one or to $\max\{1,\lfloor \lambda/2 \rfloor\}$ otherwise. We regard here the resampling variants of these algorithms, which we call the $(1+\{2\lambda,1\})$ $\text{EA}_{>0}$ and the $(1+\{2\lambda,\lambda/2\})$ $\text{EA}_{>0}$, respectively.

3 OneMax

We start our empirical investigation with the class of OneMax functions, the generalization of the function Om that assigns to each bit string x the number $|\{i \in [n] \mid x_i = 1\}|$ of ones in it. For this generalization, Om is composed with all possible XOR operations on the hypercube. More precisely, for any bit string $z \in \{0,1\}^n$ we define the function $\text{Om}_z : \{0,1\}^n \to [0..n], x \mapsto |\{i \in [n] \mid x_i = z_i\}|$, the number of bits in which x and z agree. The OneMax problem is the collection of all functions Om_z, $z \in \{0,1\}^n$.

To identify interesting parameter ranges for the offspring population size λ and the mutation rate p, we first summarize known theoretical results in Section 3.1. Results of the performance profiling will be discussed in Section 3.2.

3.1 Theoretical Bounds

OneMax is often referred to as the *drosophila of EC*. It is therefore not surprising that among all benchmark functions, OneMax is the problem for which most runtime results are available. Due to the space limit, we can only summarize a few selected results.

Concerning the (1+λ) EA, the first question that one might ask is whether or not it can be beneficial to generate more than one offspring per iteration. When using the number of function evaluations (and not the number of generations) as performance measure, intuitively, it should always be better to create the offspring sequentially, to profit from intermediate fitness gains. This intuition has been formally proven in [20], where it is shown that for all $\lambda, k \in \mathbb{N}$ the expected optimization time (i.e., the number of function evaluations until an optimal solution is queried for the first time) of the $(1 + k\lambda)$ EA cannot be better than that of the $(1 + \lambda)$ EA. This result implies that $\lambda = 1$ is an optimal choice. Note, however, that the number of *generations* needed to find an optimal solution can significantly decrease with increasing λ, so that it can be beneficial even for OneMax to run the $(1 + \lambda)$ EA with $\lambda > 1$ when a parallel function evaluations are possible.

For $\lambda = 1$ the runtime of the $(1 + 1)$ EA with **static mutation rate** $p > 0$ is quite well understood, cf. [3] for a detailed discussion. For general λ and static mutation rate $p = c/n$ (where here and henceforth $c > 0$ is assumed to be constant), the expected optimization time is $(1 \pm o(1))\big(\frac{n\lambda \ln\ln\lambda}{2\ln\lambda} + \frac{e^c}{c} n \ln n\big)$ [4]. An interesting observation made in [2] reveals that the parametrization $p = c/n$ is suboptimal: with $p = \ln(\lambda)/(2n)$ the $(1 + \lambda)$ EA needs only an expected number of $O(n\lambda/\log(\lambda) + \sqrt{\lambda} n \log n)$ function evaluations to optimize OneMax [2] (the proof requires $\lambda \geq 45$ and $\lambda = n^{O(1)}$). We call this algorithm the $(1 + \lambda)$ EA$_{p=\ln(\lambda)/(2n)}$.

When using **non-static mutation rates**, the best expected optimization time that a $(1 + \lambda)$ EA can achieve on OneMax is bounded from below by $\Omega(n\lambda/\log(\lambda) + n \log n)$ [8]. This bound is attained by a $(1 + \lambda)$ EA variant with fitness-dependent mutation rates [8]. Interestingly, it is also achieved by the self-adjusting $(1 + \lambda)$ EA$_{r/2,2r}$ described in Section 2.3.1 (the proof requires again $\lambda \geq 45$ and $\lambda = n^{O(1)}$).

Concerning the algorithms using **non-static offspring population sizes** (cf. Section 2.3.2), we do not have an explicit theoretical analysis for the $(1 + \{2\lambda, \lambda/s\})$ EA, but it is known that the expected optimization time of both the $(1 + \{2\lambda, 1\})$ EA and the $(1 + \{2\lambda, \lambda/2\})$ EA is $O(n \log n)$ [11].

Disclaimer. It is important to note that all the bounds reported above (and those mentioned in Section 4.1) hold, a priori, only for the classical $(1 + \lambda)$ EA variants, not the resampling versions regarded here in this work. For most bounds, and in particular the ones with static parameter choices, it is, however, not difficult to prove that the modifications do not change the asymptotic order of the expected optimization times. What does change, however, is the leading constant. As a rule of thumb, runtime bounds for the $(1 + \lambda)$ EA decrease by a multiplicative factor of about $1 - (1 - p)^n$ when the mutation strengths are sampled from the conditional binomial distribution $\text{Bin}_{>0}(n, p)$.

Note also that in our summary we collect only statements about the total expected *optimization time.* Fixed-target results for non-optimal target values are not yet very common in theoretical works on EAs, and neither are fixed-budget results. Here again the $(1 + 1)$ EA and the $(1 + \lambda)$ EA with static parameters form an exception as for these two algorithms a few theoretical fixed-budget results exist, cf. [10, 1, 5, 16] and references therein.

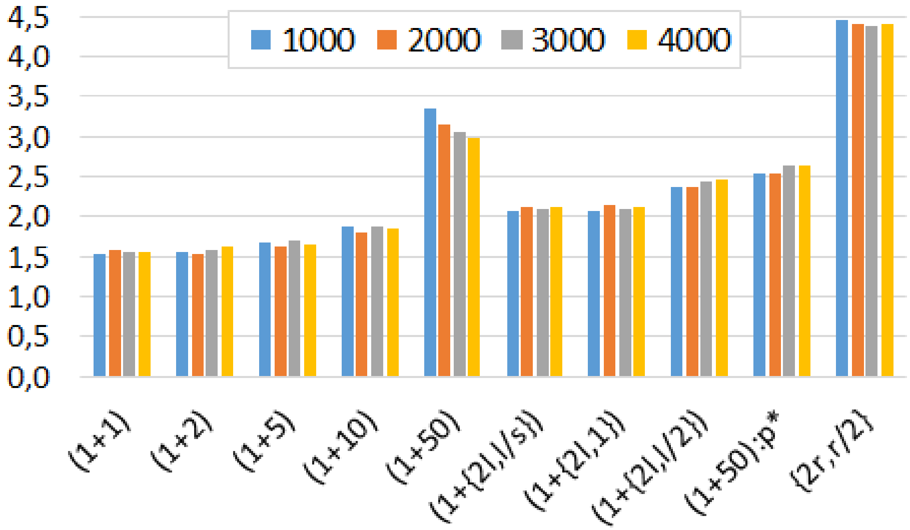


Figure 1: Average optimization times for 100 independent runs of the $(1 + \lambda)$ EA$_{>0}$ variants, normalized by $n \ln n$. (Initial) population size for the adaptive variants is 50. $p^* = \ln(\lambda)/(2n)$

3.2 Empirical Evaluation

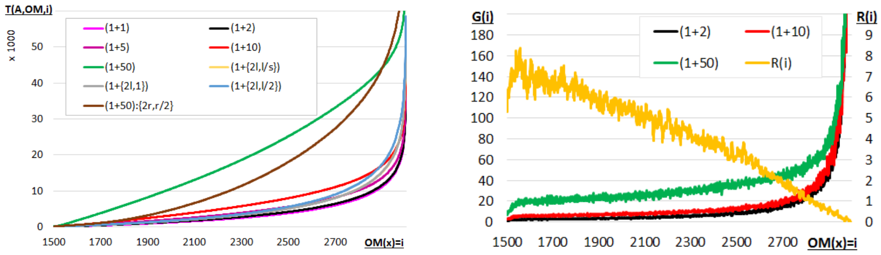
We now come to the results of our empirical investigation. All figures presented in this work are averages over 100 independent runs. This might look like a small number, but we recall that we track the whole optimization process, to present the fixed-target results below. As a rule of thumb, storing the data for the 100 runs of one algorithm on one dimension requires around 5-10 MB of storage. Despite the seemingly small number of repetitions, the numbers and results presented below are quite consistent. We also recall that in the COCO framework, an often recommended choice is to have around 15 independent runs, cf. [14, Section 3.1].

We also note that for this work we have not yet adjusted the post-processing part of the COCO environment, so that the plots in this submission have been created by Microsoft Excel. We are working on making an automated post-processing available as soon as possible.

We have seen above that, for reasonable parameter settings, the average optimization times of the $(1 + \lambda)$ EA variants are all of order $n \log n$. We therefore normalize the empirical averages in Figure 1 by this factor.

We observe that the normalized averages are quite stable across the dimensions, with an exception of the $(1+50)$ EA, whose relative performance improves with increasing problem dimension. We also see that the $(1+50)$ EA$_{>0}$ variant with $p^* = \ln(\lambda)/(2n)$ achieves a better optimization time than the $(1+50)$ EA$_{>0}$ with $p = 1/n$. The variants with adaptive offspring population size perform significantly worse than the $(1+1)$ EA$_{>0}$, which is not surprising given the performance hierarchy of the $(1+\lambda)$ EAs mentioned in the beginning of Section 3.1.

For the tested problem dimensions, the worst-performing algorithm in our comparison is the $(1 + \lambda)$ EA$_{r/2,2r}$. This may come as a surprise since this



(a) Average fixed-target runtimes $T(A, \text{O}\textsc{m}, i)$. The difference of the $(1 + \{2\lambda, \lambda/s\})$ EA$_{>0}$ to the $(1 + \{2\lambda, 1\})$ EA$_{>0}$ is less than 2% for $\text{O}\textsc{m}(x)$-values $\geq 1,800$, and the curves are therefore indistinguishable in this range.

(b) Gradients $G(A, \text{O}\textsc{m}, i)$ and relative disadvantage $R(i)$ of the (1+50) EA$_{>0}$ over the (1+2) EA$_{>0}$

Figure 2: Fixed target data for the $3,000$-dimensional OneMax problem (averages over 100 runs)

algorithm is the one with the best theoretical support. The advantage of this algorithm seems to require much larger problem dimensions, different values of λ, and/or different settings of the hyper-parameters that determined its update mechanism.

A general question raised by the data in Figure 1 concerns the sensitivity of the adaptive $(1 + \lambda)$ EA variants with respect to the initialization of their parameters and with respect to their hyper-parameters, which are the update strengths, but also the initial parameter values of λ and r, respectively.

It has been discussed in Section 3.1 that for OneMax the performance of the $(1+\lambda)$ EA can only be worse than that of the $(1+1)$ EA. In fact, the data in Figure 1 demonstrates a quite significant discrepancy between the performance of the $(1 + 1)$ EA$_{>0}$ and the $(1 + \lambda)$ EA$_{>0}$ variants with $\lambda \geq 10$. The expected optimization time of the $(1 + 50)$ EA$_{>0}$, for example, is about twice as large as that of the $(1 + 1)$ EA$_{>0}$. Intuitively, this can be explained as follows. At the beginning of the OneMax optimization process the probability that a random offspring created by the $(1 + \lambda)$ EA improves upon its parent is constant. In this phase the $(1 + \lambda)$ EA variants with small λ have an advantage as they can (almost) instantly make use of this progress, while the $(1 + \lambda)$ EAs with large λ first need to wait for all λ offspring to be evaluated. Since the expected fitness gain of the best of these λ offspring is not much larger than that of a random individual, large offspring population sizes are detrimental in this first phase of the optimization process. We note, however, that the relative disadvantage of large λ is much smaller towards the end of the optimization process. When, say, the parent individual x satisfies $\text{O}\textsc{m}(x) = n - \Theta(1)$, the probability that a random offspring has a better function value is of order $\Theta(1/n)$ only. We therefore have to create $\Theta(n)$ offspring, in expectation, before we see any progress. The relative disadvantage of creating several offspring in one generation is therefore almost negligible in the later parts of the optimization process (provided that $\lambda = O(n)$). This informal explanation is confirmed by the plots in Figure 2, which display for $n = 3,000$

1. in Figure 2a: the empirical average fixed-target times $T(A, \textsc{Om}, i)$; i.e., the average number of function evaluations that the $(1+\lambda)$ EA$_{>0}$ variant A needs to identify a solution of Om-value at least i. We cap this plot at $60,000$ evaluations.

2. in Figure 2b: the gradients $G(A,i) := T(A, \textsc{Om}, i) - T(A, \textsc{Om}, i-1)$ (three lowermost curves), and the relative difference $R_i := (\text{avg}_{j=i,i+1,\ldots,i+5}(G((1+50)\text{EA}_{>0}, j) - \text{avg}_{j=i,i+1,\ldots,i+5} G((1+2)\text{EA}_{>0}, j)/G((1+2)\text{EA}_{>0}, j))$ of the rolling average of the gradients of the $(1+50)$ and the $(1+2)$ EA$_{>0}$.

Note that the gradient $G(A,i)$ measures the average time needed by algorithm A to make a progress of one when starting in a point x of OneMax-value $\textsc{Om}(x) = i$. Small gradients are therefore desirable. We see that, for example, the (1+50) EA$_{>0}$ (green curve) needs, on average, about 20 fitness evaluations to generate a strictly better search point when starting in a solution of Om-value around 1,700. The $(1+2)$ EA$_{>0}$, in contrast, needs only about 2.6 function evaluations, on average. The relative disadvantage of the $(1+50)$ EA$_{>0}$ over the $(1+2)$ EA$_{>0}$ decreases with increasing function values from around 7 to zero, cf. the uppermost (yellow) curve in Figure 2b. We use the rolling average of 5 consecutive values here to obtain a smoother curve for $R(i)$.

Another important insight from Figure 2a is that the dominance of the $(1+1)$ EA$_{>0}$ over all $(1+\lambda)$ EA$_{>0}$ variants does not only apply to the total expected optimization time, but also for to all intermediate target values. This can be shown with mathematical rigor by adjusting the proofs in [20, Section 3] to suboptimal target values.

4 LeadingOnes

In this section we consider the performance of the $(1+\lambda)$ EA on LeadingOnes, which will shed a different light on the usefulness of large population sizes. Known theoretical bounds are summarized in Section 4.1. In Section 4.2 we discuss selected findings of our empirical investigation. These results inspire us to present a refined mathematical analysis for the expected optimization time of the $(1+\lambda)$ EA on LeadingOnes in Section 4.3.

Before presenting our results, we recall that LeadingOnes is the generalization of the function Lo, which counts the number of initial ones in the string, i.e., $\textsc{Lo}(x) = \max\{i \in [0..n] \mid \forall j \leq i : x_j = 1\}$. The generalization is by composing with an XOR-shift and a permutation of the positions. This way, we obtain for every $z \in \{0,1\}^n$ and for every permutation (one-to-one map) σ of the set $[n]$ the function $\textsc{Lo}_{z,\sigma} : \{0,1\}^n \to \mathbb{N}$, which assigns to each x the function value $\max\{i \in [0..n] \mid \forall j \in [i] : x_{\sigma(j)} = z_{\sigma(j)}\}$. The LeadingOnes problem is the collection of all these functions.

4.1 Theoretical Bounds

Also for LeadingOnes it has been proven that the optimal value of the offspring population size λ in the $(1+\lambda)$ EA is one when using function evaluations and not generations as performance indicator [20]. In contrast to OneMax, however, we will observe, by empirical and mathematical means, that the disadvantage of non-trivial population sizes is much less pronounced for this problem.

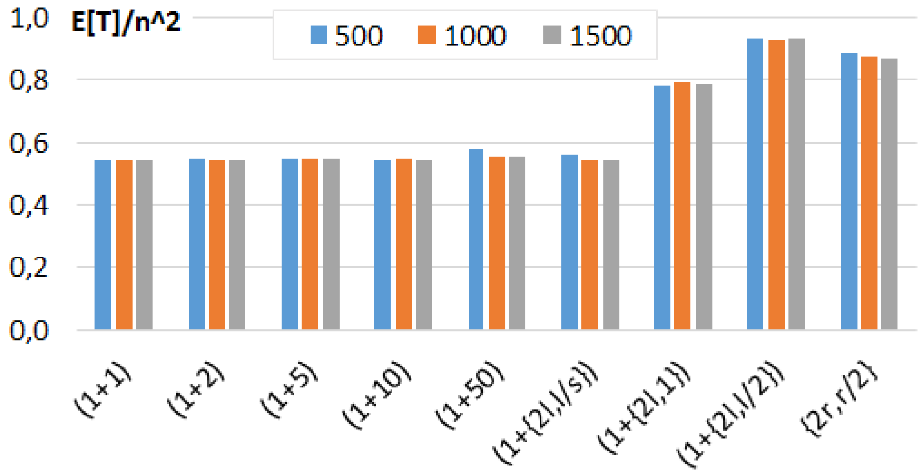


Figure 3: Average optimization times for 100 independent runs, normalized by n^2. (Initial) population size for the adaptive variants is 50.

The $(1+1)$ EA with fixed mutation probability p has an expected optimization time of $\frac{1}{2p^2}((1-p)^{-n+1}-(1-p))+1$ on LeadingOnes [17], which is minimized for $p \approx 1.59/n$. This choice gives an expected runtime of about $0.77n^2$. A fitness-dependent mutation rate can decrease this runtime further to around $0.68n^2$ [17]. For the $(1+1)$ EA$_{>0}$, it has been observed in [18] that its expected optimization time decreases with decreasing p. More precisely, it equals $\frac{1-(1-p)^n}{2p^2}((1-p)^{-n+1}-(1-p))+1$, which converges to $n^2/2+1$ for $p \to 0$.

For $\lambda = n^{O(1)}$, the expected optimization time of the $(1+\lambda)$ EA with mutation rate $p = 1/n$ is $O(n^2+n\lambda)$ [20]. With this mutation rate, the adaptive $(1+\{2\lambda, \lfloor\lambda/2\rfloor\})$ EA and the $(1+\{2\lambda,1\})$ EA achieve an expected optimization time of $O(n^2)$ [11]. Any $(1+\lambda)$ EA variant with fixed offspring population size λ but possibly adaptive mutation rate p needs at least $\Omega(\frac{\lambda n}{\ln(\lambda/n)}+n^2)$ function evaluations, on average, to optimize LeadingOnes [8].

As mentioned, we will revisit and refine the bound for the $(1+\lambda)$ EA in Section 4.3 but we first present the empirical results that have motivated this analysis.

4.2 Empirical Results

Similarly to the data plotted in Figure 1, we show in Figure 3 the normalized average optimization times of the different algorithms; the normalization factor is n^2.

For all $(1+\lambda)$ EA variants, a fairly stable performance across the three tested dimensions $n = 500$, $n = 1,000$, and $n = 1,500$ can be observed. We also see that the $(1+\{2\lambda, \lambda/s\})$ EA$_{>0}$ performs very well; it is just slightly worse than the $(1+2)$ EA$_{>0}$. The $(1+\{2\lambda, \lambda/2\})$ EA$_{>0}$, the $(1+\{2\lambda,1\})$ EA$_{>0}$, and the $(1+\lambda)$ EA$_{r/2,2r}$, in contrast, perform significantly worse than any of the tested $(1+\lambda)$ EA$_{>0}$. For the former two algorithms, preliminary test show that the disadvantage can be decreased by adjusting the update rule to the one used by

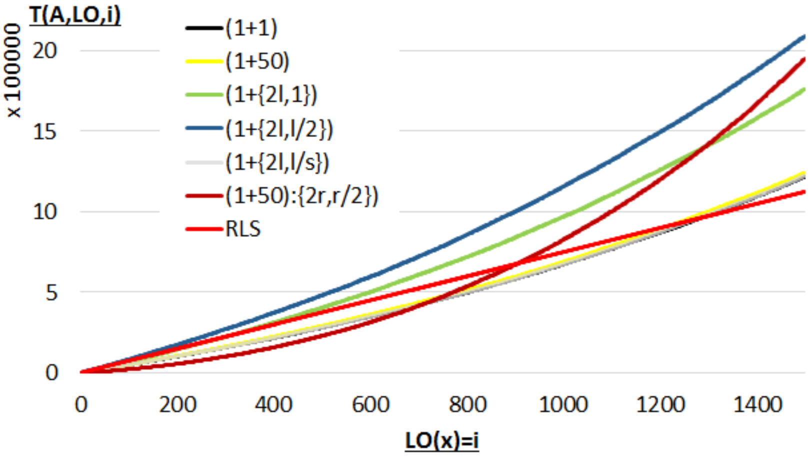


Figure 4: Fixed-target runtimes of the $(1+\lambda)$ EA$_{>0}$ variants on the $1,500$-dimensional LeadingOnes problem

the $(1+\{2\lambda,\lambda/s\})$ EA$_{>0}$, i.e., to base the update decision upon the presence of offspring that are at least as good as the parent. With this update rule the performances become very similar to that of the $(1+\{2\lambda,\lambda/s\})$ EA$_{>0}$. The $(1+\lambda)$ EA$_{r/2,2r}$ will be discussed in more detail below.

Another interesting observation is that the value of λ does not seem to have a significant impact on the expected performance. This is in sharp contrast to the situation for OneMax, cf. our discussion in the second half of Section 3.2. Building on our discussion there, we can explain this phenomenon as follows. Unlike for OneMax, the situation for LeadingOnes is that the expected fitness gain of a random offspring created by a $(1+\lambda)$ EA variant with static mutation rate $p = c/n$ is very small throughout the whole optimization process. More precisely, it decreases only mildly from around $2c/n$ when $\text{Lo}(x) = 0$ to $2c(1-c/n)^{n-1}/n \approx 2c/(e^c n)$ for $\text{Lo}(x) = n-1$. Thus, intuitively, the whole optimization process of LeadingOnes is very similar to the last steps of the OneMax optimization.

Figure 4 presents the average fixed-target runtimes for selected $(1+\lambda)$ EA$_{>0}$ variants on the $1,500$-dimensional LeadingOnes problem. We add to this figure the fixed target runtime of Randomized Local Search (RLS), the greedy (1+1)-type hill climber that always flips one random bit per iteration. RLS has a constant expected fitness gain of $2/n$ on LeadingOnes and thus a total expected optimization time of $n^2/2+1$.

We observe that the performance of the $(1+1)$ EA$_{>0}$, the $(1+50)$ EA$_{>0}$, and the $(1+\{2\lambda,\lambda/s\})$ EA$_{>0}$ are indeed very similar throughout the optimization process, the curves are almost indistinguishable. The same applies to the $(1+\lambda)$ EA$_{>0}$ with $\lambda = 2, 5, 10$, these algorithms are therefore not shown in Figure 4. We also see that the fixed-target performance of these algorithms is better than that of RLS for all Lo-values up to around $1,250 \approx 0.42n$ (exact empirical values are $1,227$ for the $(1+50)$ EA$_{>0}$ and $1,283$ for the $(1+1)$ EA$_{>0}$, but we recall that such numbers should be taken with care as they represent an average of 100 runs only. It should not be very difficult to compute the cutting point precisely, by mathematical means).

Since the only difference between the $(1+1)$ EA$_{>0}$ and RLS is the distribution from which new offspring are sampled, we see that the $(1+\lambda)$ EA variants profit from iterations in which more than one bit are flipped in the beginning of the optimization process, while they suffer from this same effect in the later parts. This situation is even more pronounced in the $(1+\lambda)$ EA$_{r/2,2r}$, which outperforms all other tested algorithms for target values up to 709. Its performance then suffers from creating at least half of its offspring with a too large mutation rate. Recall that even if the algorithm has correctly identified the optimal mutation rate $p(\textsc{Lo}(x))$, it still creates half of its offspring with mutation rate $2p(\textsc{Lo}(x))$. This results in a mediocre overall performance. This observation certainly raises the question of how to adjust the structure of the $(1+\lambda)$ EA$_{r/2,2r}$ to benefit from its good initial performance. From the viewpoint of hyper-heuristics, or algorithm selection, an adaptive selection between the $(1+\lambda)$ EA$_{r/2,2r}$ and the $(1+\lambda)$ EA$_{>0}$ would be desirable.

If we had looked only at the total optimization times, we would have classified the $(1+\lambda)$ EA$_{r/2,2r}$ as being inefficient. The fixed-target results, however, nicely demonstrate that despite the poor overall performance, there is something to be learned from this algorithm. This emphasizes the need for a fine-grained benchmarking environment.

4.3 Precise Bounds for LeadingOnes

We have observed that, for LeadingOnes, the runtimes of the $(1+\lambda)$ EA$_{>0}$ variants with static parameter choices are very close to that of the $(1+1)$ EA$_{>0}$. In Theorem 1 we show that for every constant λ, the expected optimization time of the $(1+\lambda)$ EA$_{>0}$ converges from above against that of the $(1+1)$ EA$_{>0}$ (and the same holds for the (1+λ) EA and (1+1) EA, respectively). Theorem 1 can be proven by adjusting the proofs in [17] to the (1+λ) EA. An important ingredient in this analysis is the observation that the probability of making progress in one *generation* with parent individual x equals $1-(1-p(1-p)^{\textsc{Lo}(x)})^{\lambda}$, i.e., 1 minus the probability that none of the λ offspring is better. Recall that in order to create a better offspring, none of the first $\textsc{Lo}(x)$ bits should flip, while the $(\textsc{Lo}(x)+1)$-st bit *does* need to be flipped. The results for the $(1+\lambda)$ EA$_{>0}$ can be obtained from that for the $(1+\lambda)$ EA by taking into account that the resampling strategy increases the expected fitness gain by a multiplicative factor of $1/(1-(1-p)^n)$.

Theorem 1. *For all $n, \lambda \in \mathbb{N}$ the expected optimization time of the $(1+\lambda)$ EA with static mutation rate $0<p<1$ on the n-dimensional* LeadingOnes *function is at most*

$$1+\frac{\lambda}{2}\sum_{j=0}^{n-1}\frac{1}{1-(1-p(1-p)^j)^{\lambda}} \tag{1}$$

and the expected optimization time of the $(1+\lambda)$ EA$_{>0}$ is at most

$$1+\frac{(1-(1-p)^n)\lambda}{2}\sum_{j=0}^{n-1}\frac{1}{1-(1-p(1-p)^j)^{\lambda}}. \tag{2}$$

To judge the precision of the bound stated in Theorem 1, we first note that for $\lambda = 1$ expression (1) is tight by the result presented in [17] (note though that the additive +1 term is suppressed there as they regard the number of iterations, not function evaluations). Similarly, for the $(1+1)$ EA$_{>0}$ expression (2) is tight by the bound proven in [18].

Apart from this case with trivial offspring population size $\lambda = 1$, it might be tedious to compute the expected optimization time of the $(1+\lambda)$ EA on LeadingOnes exactly, since in our proof for Theorem 1 we would have to take into account that in one generation more than one search point that improves upon the current best search point can be generated. Since the $(1+\lambda)$ EA chooses the best one of these, the distribution of this offspring would have to be computed. Note, however, that this effect can only have a very mild impact on the bounds stated above, as is occurs relatively rarely and does, in general, not result in a much larger fitness gain. Put differently, the bounds in Theorem 1 are close to tight for reasonable (i.e., not too large) values of λ.

As the expressions in Theorem 1 are not easy to interpret, we provide in Table 1 a numerical evaluation of the upper bound (2) for different values of λ and n. We add to this table (second row) the empirically observed averages, which show a good match to the theoretical bound.

	500	1,000	1,500	10,000	100,000	500,000
(1+1)	54.317%	54.313%	54.311%	54.309%	54.308%	54.308%
emp.	54.0%	54.1%	54.2%	-	-	-
(1+2)	54.349%	54.328%	54.322%	54.310%	54.308%	54.308%
emp.	54.8%	54.4%	54.2%	-	-	-
(1+5)	54.444%	54.376%	54.353%	54.315%	54.309%	54.308%
emp.	54.5%	54.8%	54.6%	-	-	-
(1+50)	55.883%	55.091%	54.829%	54.386%	54.316%	54.310%
emp.	57.6%	55.3%	55.2%	-	-	-

Table 1: Theoretical upper bounds from Theorem 1

5 Conclusions

Building on the COCO software we have developed a benchmark environment that can be used to create fine-grained performance profiles for pseudo-Boolean benchmark problems. Results for OneMax and LeadingOnes have been presented in this documentation. These results inspired a refined analysis of the expected optimization time of the $(1+\lambda)$ EA on LeadingOnes.

Our mid-term objective is to develop the benchmarking environment into a platform that practitioners and theoreticians use to exchange ideas and research questions on pseudo-Boolean black-box optimization. A key design question will be the selection of problems that shall be included in the benchmark suite. Our initial focus has been on functions for which some theoretical understanding of typical optimization processes is available, such as linear functions, LeadingOnes, Jump, etc. Going forward, we intend to include optimization problems that shed light on how the performance is influenced by certain problem features, such as the modality, the separability, the degree of constraints, etc.

Acknowledgments

This work was supported by a public grant as part of the Investissement d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH, in a joint call with Gaspard Monge Program for optimization, operations research and their interactions with data sciences. Parts of our work have been inspired by discussions of COST Action CA15140 Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice.

References

[1] Benjamin Doerr, Carola Doerr, and Jing Yang. Optimal parameter choices via precise black-box analysis. In *GECCO'16*, pages 1123–1130. ACM, 2016.

[2] Benjamin Doerr, Christian Gießen, Carsten Witt, and Jing Yang. The $(1+\lambda)$ evolutionary algorithm with self-adjusting mutation rate. In *GECCO'17*, pages 1351–1358. ACM, 2017.

[3] Carsten Witt. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability & Computing*, 22:294–318, 2013.

[4] Christian Gießen and Carsten Witt. The interplay of population size and mutation probability in the $(1+\lambda)$ EA on OneMax. *Algorithmica*, 78(2):587–609, 2017.

[5] Dogan Corus, Jun He, Thomas Jansen, Pietro S. Oliveto, Dirk Sudholt, and Christine Zarges. On easiest functions for mutation operators in bio-inspired optimisation. *Algorithmica*, 78:714–740, 2017.

[6] Eduardo Carvalho Pinto and Carola Doerr. Discussion of a more practice-aware runtime analysis for evolutionary algorithms. In *EA'17*, pages 298–305, 2017.

[7] G. Karafotias, M. Hoogendoorn, and A.E. Eiben. Parameter control in evolutionary algorithms: Trends and challenges. *IEEE Transactions on Evolutionary Computation*, 19:167–187, 2015.

[8] Golnaz Badkobeh, Per Kristian Lehre, and Dirk Sudholt. Unbiased black-box complexity of parallel search. In *PPSN'14*, volume 8672 of *LNCS*, pages 892–901. Springer, 2014.

[9] Jano I. van Hemert and Thomas Bäck. Measuring the searched space to guide efficiency: The principle and evidence on constraint satisfaction. In *PPSN'02*, volume 2439 of *LNCS*, pages 23–32. Springer, 2002.

[10] Johannes Lengler and Nicholas Spooner. Fixed budget performance of the (1+1) EA on linear functions. In *FOGA'15*, pages 52–61. ACM, 2015.

[11] Jörg Lässig and Dirk Sudholt. Adaptive population models for offspring populations and parallel evolutionary algorithms. In *FOGA'11*, pages 181–192. ACM, 2011.

[12] N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *ArXiv e-prints*, arXiv:1603.08785, 2016.

[13] N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. COCO github page. `https://github.com/numbbo/coco`, [n. d.].

[14] Nikolaus Hansen, Anne Auger, Dimo Brockhoff, Dejan Tusar, and Tea Tusar. COCO: performance assessment. *CoRR*, abs/1605.03560, 2016.

[15] Per Kristian Lehre and Pietro Simone Oliveto. Runtime analysis of population-based evolutionary algorithms: introductory tutorial at GECCO 2017. In *GECCO'17 Companion Material*, pages 414–434. ACM, 2017.

[16] Samadhi Nallaperuma, Frank Neumann, and Dirk Sudholt. Expected fitness gains of randomized search heuristics for the traveling salesperson problem. *Evolutionary Computation*, 25, 2017.

[17] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *PPSN'10*, volume 6238 of *LNCS*, pages 1–10. Springer, 2010.

[18] Thomas Jansen and Christine Zarges. Analysis of evolutionary algorithms: from computational complexity analysis to algorithm engineering. In *FOGA'11*, pages 1–14. ACM, 2011.

[19] Thomas Jansen and Christine Zarges. Performance analysis of randomised search heuristics operating with a fixed budget. *TCS*, 545:39–58, 2014.

[20] Thomas Jansen, Kenneth A. De Jong, and Ingo Wegener. On the choice of the offspring population size in evolutionary algorithms. *Evolutionary Computation*, 13:413–440, 2005.