The (1 + (λ, λ)) global SEMO algorithm

Benjamin Doerr, Omar El Hadri, Adrien Pinard

▶ **To cite this version:**

Benjamin Doerr, Omar El Hadri, Adrien Pinard. The (1 + (λ, λ)) global SEMO algorithm. GECCO '22: Genetic and Evolutionary Computation Conference, Jul 2022, Boston Massachusetts, United States. pp.520-528, ⟨10.1145/3512290.3528868⟩. ⟨hal-03806399v2⟩

HAL Id: hal-03806399

https://hal.science/hal-03806399v2

Submitted on 30 Mar 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

HAL Authorization

The $(1 + (\lambda, \lambda))$ Global SEMO Algorithm

Benjamin Doerr
Laboratoire d'Informatique (LIX), CNRS, École Polytechnique, Institut Polytechnique de Paris
Palaiseau, France

Omar El Hadri
École Polytechnique, Institut Polytechnique de Paris
Palaiseau, France

Adrien Pinard
École Polytechnique, Institut Polytechnique de Paris
Palaiseau, France

ABSTRACT

The $(1 + (\lambda, \lambda))$ genetic algorithm is a recently proposed single-objective evolutionary algorithm with several interesting properties. We show that its main working principle, mutation with a high rate and crossover as repair mechanism, can be transported also to multi-objective evolutionary computation. We define the $(1 + (\lambda, \lambda))$ global SEMO algorithm, a variant of the classic global SEMO algorithm, and prove that it optimizes the OneMinMax benchmark asymptotically faster than the global SEMO. Following the single-objective example, we design a one-fifth rule inspired dynamic parameter setting (to the best of our knowledge for the first time in discrete multi-objective optimization) and prove that it further improves the runtime to $O(n^2)$, whereas the best runtime guarantee for the global SEMO is only $O(n^2 \log n)$.

CCS CONCEPTS

• **Theory of computation → Theory of randomized search heuristics**.

KEYWORDS

Runtime analysis, multi-objective optimization, mutation operator, one-fifth success rule, dynamic parameter setting.

ACM Reference Format:
Benjamin Doerr, Omar El Hadri, and Adrien Pinard. 2022. The $(1 + (\lambda, \lambda))$ Global SEMO Algorithm. In *Genetic and Evolutionary Computation Conference (GECCO '22), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3512290.3528868

1 INTRODUCTION

The theory of evolutionary algorithms (EAs) for a long time has accompanied our attempts to understand the working principles of evolutionary computation [9, 21, 33, 43]. In the recent years, this field has not only explained existing approaches, but also proposed new operators and algorithms.

The theory of *multi-objective* EAs, due to the higher complexity of these algorithms, is still lagging behind its single-objective counterpart. There are several runtime analyses for various multi-objective EAs which explain their working principles. Also, some new ideas specific to multi-objective evolutionary algorithms (MOEAs) have been developed recently. However, many recent developments in single-objective EA theory have not been exploited in multi-objective evolutionary computation (see Section 2 for more details).

In this work, we try to profit in multi-objective evolutionary computation from the ideas underlying the $(1 + (\lambda, \lambda))$ GA. The $(1 + (\lambda, \lambda))$ GA, proposed first in [18], tries to combine a larger radius of exploration with traditional greedy-style exploitation of already detected profitable solutions. To this end, the $(1 + (\lambda, \lambda))$ GA uses mutation with a higher-than-usual mutation rate together with a biased crossover with the parent, which can repair the unwanted effects of the aggressive mutation operations.

We defer the detailed discussion of this algorithm to Section 4 and note here only that several results indicate that this basic idea was successful. In the early works on the OneMax benchmark, the $(1 + (\lambda, \lambda))$ GA was shown to have a better runtime than the $(1 + 1)$ EA for a decent range of parameters. With the optimal parameter setting, this runtime becomes $\Theta\big(n\big(\frac{\log(n) \log\log\log(n)}{\log\log(n)}\big)^{1/2}\big)$. While this is not a radical improvement over the $\Theta(n \log n)$ runtime of many classic EAs, it is still noteworthy in particular when recalling that no black-box algorithm can optimize this benchmark faster than in time $\Theta(n/\log n)$ [25]. With a suitable fitness-dependent parameter setting or a self-adjusting parameter choice following a 1/5-rule, the runtime of the $(1 + (\lambda, \lambda))$ GA can further be lowered to $O(n)$ [16]. Similar results have been obtained for random satisfiability instances in a planted solution model [13]. Together with a heavy-tailed choice of the parameters, the $(1 + (\lambda, \lambda))$ GA can also obtain a linear runtime [1]. Stronger runtime improvements over many classic algorithms have been obtained on the Jump benchmark, see [3] and the references therein. Since here the right choice of the parameters is less understood and since a multi-objective analogue of the Jump-benchmark has been proposed [23] only very recently, in this first work on multi-objective versions of the $(1 + (\lambda, \lambda))$ GA we shall concentrated on the more established OneMinMax problem, which is a multi-objective analogue of the classic OneMax benchmark.

Our results: We develop a multi-objective version of the $(1 + (\lambda, \lambda))$ GA by interpreting the main loop of the $(1 + (\lambda, \lambda))$ GA as a complex mutation operator and then equipping the classic *global simple evolutionary multi-objective optimizer (GSEMO)* with this mutation operator. For this algorithm, which we call $(1 + (\lambda, \lambda))$ GSEMO, we conduct a mathematical runtime analysis on the OneMinMax benchmark. We show that a reasonable range of static parameter settings give a better runtime than the $O(n^2 \log n)$ guarantee known for the classic GSEMO. With an optimal parameter choice we obtain a runtime of $O(n^2 \sqrt{\log n})$. With a suitable state-dependent parameter choice comparable to the one of [18], the runtime drops further to $O(n^2)$. Since such a state-dependent parameter choice requires a deep understanding of the algorithm and the problem, we then design a self-adjusting parameter

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
GECCO '22, July 9–13, 2022, Boston, MA, USA. **Author generated version.**
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/3512290.3528868

setting inspired by the classic one-fifth success rule. Some adjustments are necessary to work in this multi-objective setting, but then we obtain a simple dynamic parameter setting that gives the same $O(n^2)$ runtime as with the complicated state-dependent parameter choice. To the best of our knowledge, this is the first time that such a dynamic parameter choice is developed for a MOEA for a discrete search space. From a broader perspective, this work shows that the main idea of the $(1 + (\lambda, \lambda))$ GA, so far only used with $(1 + 1)$-type algorithms, can also be used in more complex algorithmic frameworks.

This work is organized as follows. In the following section, we describe the previous works most relevant to ours. In Section 3, we briefly recall the basic notations of MOEAs and state the OneMinMax problem. We develop the $(1 + (\lambda, \lambda))$ GSEMO in Section 4 and conduct our mathematical runtime analysis for static parameters in Section 5. In Section 6, we propose state-dependent parameters and prove the $O(n^2)$ runtime guarantee for these. We develop and analyze the self-adjusting parameter choice inspired by the one-fifth rule in Section 7. A short experimental evaluation in Section 8 shows that already the $(1 + (\lambda, \lambda))$ GSEMO with static parameters easily outperforms the classic GSEMO and this already for small problem sizes. We summarize our findings and discuss future research ideas in Section 9.

2 PREVIOUS WORK

Soon after the first runtime analyses of single-objective EAs have appeared, see, e.g., [24, 27, 49] for three early and influential works, also multi-objective EAs were studied under this theory perspective. These works, e.g., [28, 37, 38], followed the example of the theoretical works on single-objective EAs and proved estimates on the runtime of simple MOEAs such as the SEMO and GSEMO on benchmark problems that were defined as multi-objective analogues of classic benchmarks like OneMax or LeadingOnes.

In the recent past, the theory of MOEA has more concentrated on research topics which are specific to multi-objective optimization, e.g., parent selection schemes that speed up the exploration of the Pareto front [45], approximations of the Pareto front [11, 19], or specific MOEAs such as the MOEA/D or the NSGA-II [10, 22, 30, 31, 40, 54, 55]. While it is natural that the theory of MOEAs has regarded these questions, at the same time this carries the risk that trends and insights from the general EA theory are not exploited in multi-objective evolutionary computation. In fact, this effect is already very visible. Topics such as precise runtime analyses (see, e.g., [4] and the references therein), fixed-budget analysis [35], and optimal parameter settings (see, e.g., [53] for an early such result in single-objective EA theory) have not been considered yet in multi-objective EA theory. More critically, also the recently developed new algorithmic building blocks, for example, a large number of successful ways to dynamically set parameters [17], memetic algorithms with proven performance guarantees (see [44] and the references therein), and hyperheuristic approaches with proven guarantees (see [41] and the references therein) have all not been considered for MOEAs. In fact, to the best our our knowledge, the only example of a work transporting recent algorithmic ideas developed in single-objective EA theory to the multi-objective world is [23], where the fast mutation of [20] and the stagnation detection of [46] are used in a MOEA.

For this reason, we shall study in this work how another algorithmic idea recently proposed in EA theory can be used in multi-objective evolutionary computation, namely the $(1 + (\lambda, \lambda))$ GA. We defer an account of the previous work on this algorithm to Section 4, where this algorithm will be detailed.

3 PRELIMINARIES

In this section, we give a brief introduction to multi-objective optimization and to the notation we use.

For $a, b \in \mathbb{R}$, we write $[a..b] = \{z \in \mathbb{Z} \mid a \leq z \leq b\}$ for the integers in the interval $[a, b]$. We denote the binomial distribution with parameters n and p by $\mathrm{Bin}(n, p)$ and we write $X \sim \mathrm{Bin}(n, p)$ to denote that X is a sample from this distribution, that is, that $\Pr[X = i] = \binom{n}{i} p^i (1-p)^{n-i}$ for all $i \in [0..n]$.

For the ease of presentation, in the remainder we shall concentrate ourselves on two objectives which have to be maximized. A bi-objective function on the search space Ω is a pair $f = (f_1, f_2)$, where each $f_i : \Omega \to \mathbb{R}$. We write $f(x) = (f_1(x), f_2(x))$ for all $x \in \Omega$. We shall always assume that we have a bit-string representation, that is, $\Omega = \{0,1\}^n$ for some $n \in \mathbb{N}$. The challenge in multi-objective optimization is that often there is no solution x that maximizes both f_1 and f_2.

We say x *weakly dominates* y, denoted by $x \succeq y$, if and only if $f_1(x) \geq f_1(y)$ and $f_2(x) \geq f_2(y)$. We say x *strictly dominates* y, denoted by $x \succ y$, if and only if $f_1(x) \geq f_1(y)$ and $f_2(x) \geq f_2(y)$ and at least one of the inequalities is strict. We say that a solution is Pareto-optimal if it is not strictly dominated by any other solution. The set of objective values of all Pareto optima is called the Pareto front of f. In this work, as in many previous works on the (G)SEMO family of algorithms, our aim is to compute the full Pareto front, that is, to compute a set P of Pareto optima such that $f(P) = \{f(x) \mid x \in P\}$ is the Pareto front.

In this work, we will mainly be interested in the OneMinMax benchmark function, which is a bi-objective analogue of the classic OneMax benchmark. It is defined by

$$\textsc{OneMinMax} : \{0,1\}^n \to \mathbb{R}^2;$$
$$x \mapsto \left(\sum_{i=1}^{n} x_i, \sum_{i=1}^{n} (1 - x_i) \right),$$

that is, the first objective counts the number of ones in x and the second objective counts the number of zeros. We immediately see that any $x \in \{0,1\}^n$ is Pareto optimal. Hence the Pareto front of this problem is $\{(i, n-i) \mid i \in [0..n]\}$.

4 FROM THE $(1 + (\lambda, \lambda))$ GA TO THE $(1 + (\lambda, \lambda))$ GSEMO

In this section, we design a MOEA building on the main ideas of the $(1 + (\lambda, \lambda))$ GA. We start with a brief description of the $(1 + (\lambda, \lambda))$ GA and the known results on this algorithm, then move on to the GSEMO, and finally discuss how to merge the two.

4.1 The $(1 + (\lambda, \lambda))$ GA

The $(1 + (\lambda, \lambda))$ GA is a still simple genetic algorithm for the maximization of pseudo-Boolean functions, that is, functions f :

$\{0,1\}^n \to \mathbb{R}$ defined on the set of bit-strings of length n. The $(1 + (\lambda, \lambda))$ GA works with a parent population of size one. Its parameters are the offspring population size $\lambda \in \mathbb{N}$, the mutation rate $p \in [0, 1]$, usually parameterized as $p = k/n$ for some number $k \in [0, n]$, and the crossover bias $c \in [0, 1]$. In a first mutation stage, from the parent individual λ offspring are created. Each offspring is distributed as if obtained from bit-wise mutation with mutation rate p, however, to remedy the risk that offspring are better or worse just because they have a different distance from the parent, all offspring are created in the same distance. Consequently, first a number ℓ is sampled according to a binomial distribution with parameters n and p and then λ offspring are generated each by flipping a random set of exactly ℓ bits in the parent. In the intermediate selection stage, an offspring with maximal fitness is selected (breaking ties randomly).

Due to the usually high mutation rate used in the $(1 + (\lambda, \lambda))$ GA, this mutation winner will typically be much worse than the parent. Being the best among the offspring, we can still hope that besides all destruction through the aggressive mutation, it has also gained some advantages. The crossover stage now aims at preserving these advantages and repairing the unwanted destruction. To this aim, in the crossover phase λ offspring are created from a biased crossover between mutation winner and original parent. This biased crossover takes bits from the mutation winner with probability c, otherwise from the parent. In the final selection stage, the best crossover offspring is taken as new parent, except if it is worse than the original parent, in which case the original parent is kept. The pseudocode of this algorithm is given as Algorithm 1.

Algorithm 1: The $(1 + (\lambda, \lambda))$ GA with offspring population size λ, mutation rate $p = k/n$, and crossover bias c for the maximization of a function $f : \{0,1\}^n \to \mathbb{R}$.

```
Sample x ∈ {0,1}^n uniformly at random;
for t = 1, 2, 3, ... do
    Sample ℓ from a binomial distribution B(n, k/n);
    Generate x_1, x_2, ..., x_λ ∈ {0,1}^n each by flipping ℓ
      random bits of x;
    Select x^+ ∈ {x_1, x_2, ..., x_λ} such that x^+ maximizes f;
    Generate x_1^+, x_2^+, ..., x_λ^+ ∈ {0,1}^n via cross_c(x, x^+);
    Select y ∈ {x_1^+, x_2^+, ..., x_λ^+} such that y maximizes f;
    if f(y) ≥ f(x) then
        x ← y;
```

It is not completely understood how to optimally set the parameters for this algorithm, but the original work [18] proposed to take as mutation rate $p = \lambda/n$ and as crossover bias $c = 1/\lambda$. This setting is natural in the sense that an individual created by mutation and subsequent crossover with the parent has the same distribution as if generate with bit-wise mutation with mutation rate $1/n$, which is the common way of performing mutation. We shall call this the *standard parameter setting* for the $(1 + (\lambda, \lambda))$ GA.

In a first runtime analysis of the $(1 + (\lambda, \lambda))$ GA, it was shown that the $(1 + (\lambda, \lambda))$ GA with parameters $\lambda \geq 2$ as well as $p = k/n$ and $c = 1/k$ for some $k \geq 2$ optimizes the OneMax benchmark in time $O((\frac{1}{k} + \frac{1}{\lambda})n \log n + (k+\lambda)n)$. This bound is minimized for $\lambda = k$, that is, the standard parameter setting, and further for $\lambda = k = \Theta(\sqrt{\log n})$. In this case, the runtime guarantee becomes $O(n\sqrt{\log n})$, which is asymptotically faster than the $\Theta(n \log n)$ runtime of many evolutionary algorithms [5, 8, 24, 34, 39, 42, 48, 51, 52]. The truly optimal parameters, determined in [16], minimally deviate from these, but they also belong to the standard setting and they only improve the runtime by a $\Theta\big(\big(\frac{\log\log\log n}{\log\log n}\big)^{1/2}\big)$ factor, so we skip the details.

In [18], also a fitness-dependent parameter choice was proposed. If, in the standard setting, λ is chosen as $\sqrt{\frac{n}{n-f(x)}}$, where $f(x)$ is the fitness of the current solution, then the runtime reduces to $O(n)$.

Since both the best static and this fitness-dependent parameter setting are non-trivial to find, in [16] a self-adjusting parameter choice was proposed. If, again in the standard setting, λ is controlled via a variant of the 1/5 success rule, then the actual value of λ follows closely the fitness-dependent choice from [18], resulting also in an $O(n)$ runtime. These first results (apart from the very precise optimal static parameter setting of [16]) are the basis of our work.

For this reason, we now describe in less detail the remaining previous works on the $(1 + (\lambda, \lambda))$ GA. Results similar to the ones just described were shown in [13] for the optimization of certain random SAT instances. On OneMax again, a heavy-tailed choice of λ also gives a linear runtime [1]. On the LeadingOnes benchmark, the $(1 + (\lambda, \lambda))$ GA does not have a runtime advantage over classic EAs, however, with all parameter choices in the standard setting it also obtains the $O(n^2)$ runtime of these algorithms [6]. On the multimodal jump benchmark, a perfect understanding of the $(1 + (\lambda, \lambda))$ GA has not yet been obtained, but the existing results [2, 3, 7, 26] show that with parameters outside the standard regime or a heavy-tailed parameter choice, runtimes of roughly $n^{m/2}$ can be obtained on jump functions with jump size m, which is significantly faster than the $\Theta(n^m)$ time of, e.g., the $(1 + 1)$ EA [24].

4.2 The GSEMO

The *global simple evolutionary multi-objective optimizer (GSEMO)* [28], different from the SEMO algorithm [37] only in that is uses global mutations, is an algorithm that computes the full Pareto front of a multi-objective optimization problem. It uses a population of variable size which always consists of solutions that are incomparable, that is, no solution dominates another one. The algorithm starts with a population consisting of a single random individual. In its main loop, from a parent randomly chosen from the population an offspring is created via bit-wise mutation with mutation rate $1/n$. Any individual dominated by the offspring is removed from the population, then the offspring is added to the population if it is not dominated by a member of the population. The number of iterations this algorithm takes to find the full Pareto front of a multi-objective problem is called the *runtime* of the GSEMO.

The runtime of the GSEMO on the OneMinMax benchmark, the natural bi-objective version of the OneMax problem, is $O(n^2 \log n)$. This bound was shown for the SEMO only [29], but it is easy to see that the proof extends to the GSEMO. A lower bound of $\Omega(n^2 \log n)$

also exists only for the SEMO; this proof, however, does not immediately extend to the GSEMO.

4.3 Designing the $(1 + (\lambda, \lambda))$ GSEMO

A closer look at the $(1 + (\lambda, \lambda))$ GA reveals that we can interpret this algorithm as a variant of the $(1 + 1)$ EA that uses a complex mutation operator. This mutation operator creates λ offspring from a given parent, selects a best of these ("mutation winner"), creates λ offspring via a biased crossover between parent and mutation winner, and returns a best of these.

With this view, the natural way to merge the GSEMO and the $(1 + (\lambda, \lambda))$ GA is to use the GSEMO with this complex mutation operator instead of bit-wise mutation. Since our complex mutation operator relies on a fitness function, there is one more design choice to take, namely what to use as fitness in a multi-objective setting. Again, there is a natural choice, and this is to use all objectives the multi-objective problem is composed of. Hence in the mutation phase, we once generate λ offspring, then for each objective we select a mutation winner, and then, separately for each of them, we conduct a crossover phase. For a d-objective problem, this yields $d\lambda$ crossover offspring. For all of these, we check if it is profitable to add them to the population, more precisely, sequentially for each of them we remove the individuals it dominates and add it to the population if it is not dominated by a current member of the population. The pseudocode for this algorithm, which we call $(1 + (\lambda, \lambda))$ GSEMO, is given in Algorithm 2.

We note that in all algorithms, we did not specify a termination criterion. This is justified by the fact that in this scientific work we are only interested in the first point in time when we have reached a certain target. In a practical application, of course, one needs to specify a termination criterion.

Algorithm 2: The $(1 + (\lambda, \lambda))$ GSEMO.

Generate $x \in \{0,1\}^n$ uniformly at random and $P \leftarrow \{x\}$;
while *not stop condition* **do**
  Uniformly at random select one individual x from P;
  Sample ℓ from a binomial distribution $\mathcal{B}(n, \frac{k}{n})$;
  Generate $x_1, x_2, ..., x_\lambda \in \{0,1\}^n$ each by flipping ℓ random bits of x;
  Select $x^+, x^- \in \{x_1, x_2, ..., x_\lambda\}$ such that x^+ maximizes f_1 and x^- maximizes f_2;
  Generate $x_1^+, x_2^+, ..., x_\lambda^+ \in \{0,1\}^n$ via $\text{cross}_c(x, x^+)$;
  Generate $x_1^-, x_2^-, ..., x_\lambda^- \in \{0,1\}^n$ via $\text{cross}_c(x, x^-)$;
  for $y \in \{x_1^+, x_2^+, ..., x_\lambda^+, x_1^-, x_2^-, ..., x_\lambda^-\}$ **do**
    if *there is no* $z \in P$ *such that* $y \preceq z$ **then**
      $P = \{z \in P | z \not\preceq y\} \cup \{y\}$;
  return P

4.4 Our Results

We analyse the $(1 + (\lambda, \lambda))$ GSEMO algorithm by theoretical means and through experiments. Similar to the first works on the single-objective $(1 + (\lambda, \lambda))$ GA, which all regarded the OneMax benchmark, we restrict ourselves in this first analysis of the $(1 + (\lambda, \lambda))$ GSEMO to the OneMinMax problem, which is a bi-objective version of the OneMax problem.

We show that the expected runtime (that is, the expected number of fitness evaluations until an optimal solution is found) of our algorithm is $O\big(\big(\frac{1}{k} + \frac{1}{\lambda}\big)n^2 \log n + (k + \lambda)n^2\big)$ when the crossover bias is taken as $c = \frac{1}{k}$, which is what the intuition given in Section 4.1 suggest. Consequently, quite a broad selection of choices of k and λ leads to expected optimization times better than the classic $\Theta(n^2 \log n)$. This runtime bound suggests to take $k = \Theta(\sqrt{\log n})$ and $\lambda = \Theta(\sqrt{\log n})$, we obtain an expected optimization time of $O(n^2\sqrt{\log n})$. Note that all other choices of $\lambda \in [\omega(1), o(\log n)]$ give an asymptotically better runtime as well, so there is some indication that this approach is useful also for problems for which analyzing the optimal parameter choices is not possible.

The insight into the working principles of the $(1 + (\lambda, \lambda))$ GSEMO gained in the theoretical analysis can be used to design a state-dependent choice of λ giving an even better expected runtime. If in each iteration we chose λ to be of order $\sqrt{n/d}$, where d is the minimum of the fitness-distance to the optimum of the two objective functions, the resulting algorithm has a quadratic expected optimization time only.

Since the state-dependent parameter choice was very successful (giving provably a quadratic expected runtime), but possibly hard to find without theoretical analyses, we also investigate a simple self-adjusting choice of λ. To this aim, we imitate the one-fifth success rule from evolution strategies, which was independently discovered in [14, 47, 50]. For a suitable constant $F > 1$, we multiply λ by $F^{1/(5n-1)}$ after each unsuccessful iteration and we divide it by F after each iteration that found a superior solution. As we shall show, this also leads to a quadratic runtime.

5 RUNTIME ANALYSIS FOR STATIC PARAMETERS

In this section, we conduct a rigorous runtime analysis of the $(1 + (\lambda, \lambda))$ GSEMO with static parameters on the OneMinMax benchmark. For all runtime analysis results, we recall that the standard performance measure is the optimization time (also "runtime") defined as follows.

Definition: The *optimization time* of a MOEA A on a function f is the random variable $T = T(A, f)$ that denotes the number of fitness evaluations performed until the first time the whole Pareto front P^* is covered by the population of A.

Observe that one iteration of the $(1 + (\lambda, \lambda))$ GSEMO requires 3λ function evaluations. Assume that we are working with a static value for λ. If t^* is the first iteration after which the $(1 + (\lambda, \lambda))$ GSEMO has the whole Pareto front covered by the population, then the optimization time T of this run is $3t^*\lambda + 1$; recall that also the initial search point has to be evaluated. Hence the optimization time and the first iteration t^* to cover the Pareto front deviate basically by a factor of 3λ. We shall thus argue with either of the two notions, but state the main results in terms of the optimization time. This will be different in Sections 6 and 7, where a varying λ forbids this simplification.

The main result of this section is the following runtime bound, which in particular shows that our $(1+(\lambda,\lambda))$ GSEMO for all $k, \lambda \in [\omega(1), o(\log n)]$ is faster than the GSEMO on OneMinMax.

Theorem 1. *Let $k, \lambda \geq 2$, possibly depending on n. The expected optimization time of the $(1+(\lambda,\lambda))$ GSEMO with mutation rate $p = \frac{k}{n}$ and crossover bias $c = \frac{1}{k}$ on the OneMinMax function is*

$$O\left(\left(\frac{1}{k} + \frac{1}{\lambda}\right) n^2 \log n + (k+\lambda) n^2\right).$$

In particular, for both k and λ in $\Theta(\sqrt{\log n})$, the expected optimization time is of order at most $n^2\sqrt{\log n}$.

To prove this result, we first estimate the time it takes to generate a neighbor of an existing point in the population. More precisely, in the following three lemmas, we regard the situation that P contains an element x with $f(x) = (n-d, d)$, but no element y with $f(y) = (n-d+1, d-1)$; we shall then try to bound the time it takes until a y with $f(y) = (n-d+1, d-1)$ is contained in the population.

In the following two lemmas, analyzing separately the mutation and crossover phase, let us condition on a fixed outcome ℓ of the number of bits flipped in the mutation phase. We say that the mutation phase is successful if x was chosen as parent individual and at least one of the λ offspring of x has an f_1-value greater than $n-d-\ell$. Note that then also the mutation winner x^+ has such an f_1-value.

Lemma 2. *Then the success probability of the mutation phase is at least $\frac{1}{n}(1-(1-\frac{d}{n})^{\lambda\ell})$.*

Proof. We have a probability of at least $\frac{1}{n}$ of picking x as a parent. Conditional on this, consider a fixed offspring. The probability that it has an f_1-value of $n-d-\ell$ is

$$\binom{n-d}{\ell} \Big/ \binom{n}{\ell} \leq \left(\frac{n-d}{n}\right)^{\ell}.$$

Since the λ offspring are sampled independently, the probability that at least one of them has an f_1-value of more than $n-d-\ell$, that is, that the mutation phase is successful, is at least $1-(1-\frac{d}{n})^{\ell\lambda}$. □

We now turn to the analysis of the crossover phase. Assuming the mutation phase to be successful, we call the crossover phase successful if it leads to the creation of a solution y with $f_1(y) = n-d+1$.

Lemma 3. *If the mutation phase was successful, the crossover phase is successful with probability at least $1-\left(1-c(1-c)^{\ell-1}\right)^{\lambda}$.*

Proof. Since the mutation phase was successful, the mutation winner x^+ has an f_1-value of more than $n-d-\ell$. Consequently, there is at least one bit position out of the ℓ positions x and x^+ differ in such that a crossover offspring inheriting this bit from x^+ and the $\ell-1$ others from x has an f_1-value of exactly $n-d+1$. The probability that a fixed crossover offspring x_i^+ is of this kind is $c(1-c)^{\ell-1}$. Hence the probability that at least one crossover offspring is of this kind is

$$\Pr[\exists i \in [\lambda] : f_1(x_i^+)) = n-d+1] \geq 1-\left(1-c(1-c)^{\ell-1}\right)^{\lambda}. \quad \square$$

With the two lemmas above, we can now show a lower bound for the probability of finding a desired element on the Pareto front.

Lemma 4. *Assume that at a time t, we have $x \in P$ with $f(x) = (n-d, d)$, but there is no $y \in P$ with $f(y) = (n-d+1, d-1)$. Then there is a constant $C > 0$ such that the probability that at time $t+1$ we have such a y in P is at least*

$$p_{n-d}^+ := \frac{C}{n}\left(1-\left(\frac{d}{n}\right)^{\lambda k/2}\right)\left(1-e^{-\lambda/(8k)}\right).$$

The expected time it takes to have such a y in the population (counted from iteration t on), is at most $t_{n-d}^+ := 1/p_{n-d}^+$ iterations.

Proof. Let E denote the event that at time $t+1$, we have a $y \in P$ with $f_1(y) = (n-d+1)$. Let L be the random variable describing the value of ℓ sampled in line 4 of Algorithm 2. By the law of total probability, we have

$$\Pr[E] \geq \sum_{\ell=\lceil k/2 \rceil}^{\lfloor 3k/2 \rfloor} \Pr[E \mid L=\ell] \Pr[L=\ell].$$

Let us denote by E_1 the event that the mutation phase was successful and by E_2 the event that the crossover phase was successful. We note that $E \supseteq E_1 \wedge E_2$. Using Lemma 2 and 3, we estimate, for a given $\ell \in [k/2..3k/2]$,

$$\begin{aligned}\Pr[E \mid L=\ell] &\geq \Pr[E_1 \mid L=\ell] \Pr[E_2 \mid E_1 \wedge L=\ell] \\ &\geq \frac{1}{n}\left(1-\left(\frac{d}{n}\right)^{\lambda\ell}\right)\left(1-\left(1-c(1-c)^{\ell-1}\right)^{\lambda}\right).\end{aligned}$$

Using the facts that $k \geq 2$, $c = 1/k$, and that we are only interested in values $\ell \in [k/2..3k/2]$, we compute

$$\begin{aligned}\left(1-c(1-c)^{\ell-1}\right)^{\lambda} &\leq \left(1-\frac{1}{k}\left(1-\frac{1}{k}\right)^{3k/2}\right)^{\lambda} \\ &\leq (1-1/(8k))^{\lambda} \leq e^{-\lambda/(8k)},\end{aligned}$$

where we use in the second step the fact that for all $m \geq 2$ we have $(1-1/m)^m \geq 1/4$ and in the third step that for all $m \geq 2$ we have $1/e \geq (1-1/m)^m \geq 1/(2e)$; in the following, we shall use these inequalities without explicit mention. Since we are interested only in $\ell \geq k/2$, we may estimate $\left(\frac{d}{n}\right)^{\lambda\ell} \leq \left(\frac{d}{n}\right)^{\lambda k/2}$. Thus, in total we obtain

$$\Pr[E \mid L=\ell] \geq \left(1-\left(\frac{d}{n}\right)^{\lambda k/2}\right)\left(1-e^{-\frac{\lambda}{8k}}\right),$$

which is independent of $\ell \in [k/2..3k/2]$.

Finally, it is not hard to see that $\sum_{\ell=\lceil k/2 \rceil}^{\lfloor 3k/2 \rfloor} \Pr[L=\ell]$ is constant. For $k = \omega(1)$ this follows easily from Chernoff's bound. For constant k we trivially have $\Pr[L=k] = \Theta(1)$. This proves the claim on p_{n-d}^+.

The estimate for t_{n-d}^+ simply follows from the fact that the critical assumption that P contains an x with $f(x) = (n-d, d)$ is true in all future iterations as well. Hence by the above, the time to find the desired y is stochastically dominated [15] by a geometric random variable with success rate p_{n-d}^+, hence its expectation is at most $1/p_{n-d}^+$. □

From Lemma 4, we now easily prove Theorem 1.

Proof of Theorem 1. By definition of the $(1+(\lambda,\lambda))$ GSEMO, which is a variant of the the GSEMO, once a point of the Pareto front is covered by the population, it remains so forever. We can use this observations together with Lemma 4, which gives an estimate on the time it takes for a neighbor of a covered point to be covered as well, to estimate the runtime by the sum of the estimates (from Lemma 4) for the waiting times to cover an additional point.

Let us suppose that the random initial individual is x with $f(x) = (n-d, d)$ for some $d > 0$. We estimate the time until for all $j \in [n-d+1..n]$ we have an individual y with $f(y) = (j, n-j)$ in the population by

$$\sum_{j=n-d+1}^{n} t^+_{j-1} \leq \frac{n}{C} \sum_{i=1}^{n} \left(1 - \left(1 - \frac{i}{n}\right)^{\lambda k/2}\right)^{-1} \left(1 - e^{-\lambda/(8k)}\right)^{-1}. \quad (1)$$

For $\lambda k i > 2n$, we have that $\left(1-\frac{i}{n}\right)^{\lambda k/2} \leq e^{-\frac{i\lambda k}{2n}} \leq e^{-1}$. On the other hand, by Bernoulli's inequality it holds that $(1-x)^m \leq (1+mx)^{-1}$ for $m \in \mathbb{Z}_{>0}$ and $x \in [0,1]$. Hence for $\lambda k i \leq 2n$ we get that $\left(1-\frac{i}{n}\right)^{\lambda k/2} \leq \left(1+\frac{\lambda k i}{2n}\right)^{-1} = 1 - \frac{\lambda k i}{2n+\lambda k i} \leq 1 - \frac{\lambda k i}{4n}$. Thus we have that

$$\begin{aligned}\sum_{i=1}^{n}\left(1 - \left(1-\frac{i}{n}\right)^{\lambda k/2}\right)^{-1} &\leq \sum_{i=1}^{n}\left(1 - \max\left\{1 - \frac{\lambda k i}{4n}, e^{-1}\right\}\right)^{-1} \\ &\leq \sum_{i=1}^{n} \max\left\{\frac{4n}{\lambda k i}, (1-e^{-1})^{-1}\right\} \quad (2) \\ &\leq O\left(\frac{n\log(n)}{\lambda k} + n\right).\end{aligned}$$

Finally, we estimate the factor $\left(1-e^{-\frac{\lambda}{8k}}\right)^{-1}$ in (1). Clearly, if $\frac{\lambda}{8k} \geq 1$, then $\left(1-e^{-\frac{\lambda}{8k}}\right)^{-1} \geq 1 - e^{-1} = O(1)$. For $\frac{\lambda}{8k} < 1$, we use the fact that for all $x > 0$ we have $\exp(-x) < 1 - x + \frac{x^2}{2}$ and bound

$$e^{-\frac{\lambda}{8k}} < 1 - \frac{\lambda}{8k} + \frac{1}{2}\left(\frac{\lambda}{8k}\right)^2 \leq 1 - \frac{\lambda}{8k} + \frac{1}{2}\frac{\lambda}{8k} = 1 - \frac{\lambda}{16k}.$$

This shows that for $\frac{\lambda}{8k} < 1$ we have

$$\left(1-e^{-\frac{\lambda}{8k}}\right)^{-1} < \left(\frac{\lambda}{16k}\right)^{-1} = O(k/\lambda).$$

Consequently, in both cases, we have

$$\left(1-e^{-\frac{\lambda}{8k}}\right)^{-1} = O\left(\max\left\{1, \frac{\lambda}{k}\right\}\right). \quad (3)$$

Replacing the terms in (1) by (2) and (3) it is thus easy to see that the overall number of iterations needed is

$$O\left(\max\left\{1, \frac{k}{\lambda}\right\}\left(\frac{n\log n}{\lambda k} + n\right)\right).$$

Since one iteration of Algorithm 2 requires 3λ fitness evaluations, the expected time (in terms of fitness evaluations) is

$$\begin{aligned}&O\left(n\lambda\left(\frac{n\log n}{\lambda k} + n\right) + kn\left(\frac{n\log n}{\lambda k} + n\right)\right) \\ &\quad = O\left(\left(\frac{1}{k} + \frac{1}{\lambda}\right)n^2\log n + (k+\lambda)n^2\right). \quad (4)\end{aligned}$$

We recall that this estimates the time to find a solution y with $f(y) = (j, n-j)$ for all $j \in [n-d+1..n]$ from an initial solution x with $f(x) = (n-d, d)$, and more generally, from an arbitrary initial state containing such an x. By symmetry, and since the above estimate does not depend on d, the same estimate holds for the time to find a y with $f(y) = (j, n-j)$ for all $j \in [0..n-d-1]$. Hence the entire expected runtime is the sum of these two (identical) expressions, and this finished the proof. □

6 STATE-DEPENDENT PARAMETERS

In this section we prove that a suitable state-dependent choice of λ, ensuring larger λ values towards the more difficult end of the optimization process, together with the standard parameter setting for k and c, provably yields an asymptotic speed-up, namely an expected optimization time of $O(n^2)$. We are not aware of any previous results showing more than a constant-factor gain through a dynamic parameter choice for a MOEA for discrete search spaces.

To describe the quality of a state, we need the following definition.

Definition: Let P be a population. For $b \in \{1,2\}$, let $o_b(P) := \min\{j \in [0..n-1] \mid (\exists x \in P : f_b(x) = j \wedge (\forall y \in P : f_b(y) \neq j+1)\}$.

The idea behind this definition is to grasp the missing solution that is easiest to find. If $o_1(P) = n-d$ for some d, then we are in the situation of Lemma 4, that is, we have a solution $x \in P$ with $f(x) = (n-d, d)$, but we do not have a solution y with $f(y) = (n-d+1, d-1)$, and among all such pairs this pair (x, y) is the one that most easily allows to generate y from x. We then choose λ in such a way that an expected constant number of iterations is sufficient to find y.

Theorem 5. *Consider the $(1+(\lambda,\lambda))$ GSEMO with standard parameters $p = \frac{\lambda}{n}$ and $c = \frac{1}{\lambda}$ together with a state-dependent choice of λ such that in the beginning of each iteration λ is set to $\lambda^* = \sqrt{\frac{n}{n-\min(o_1(P),o_2(P))}}$. Then the expected optimization time on OneMinMax is $O(n^2)$.*

Proof. Consider one iteration of this $(1+(\lambda,\lambda))$ GSEMO. By symmetry, assume that $\min(o_1(P), o_2(P)) = o_1(P)$. Applying Lemma 4 with $n-d = o_1(P)$ and $\lambda = k = \lambda^*$, we see that we have a probability of $\Omega(1/n)$ to find a search point y with $f(y) = (n-d+1, d-1)$ in one iteration. Consequently, we use each different value of λ^* only for an expected number of $O(n)$ iterations.

This bounds the entire expected optimization time by

$$O\left(n\sum_{d=0}^{n-1}\sqrt{n/(n-d)}\right) = O\left(n\sqrt{n}\int_1^n \sqrt{1/i}\,\mathrm{d}i\right) = O(n^2).$$

□

7 SELF-ADJUSTING PARAMETER CHOICES

The previous section showed that a non-trivial state-dependent parameter choice gave better results than the best static parameter setting we found. The question is how an algorithm user would find this state-dependent parameter setting. Fortunately, as in the single-objective case [16], there is a way to let the algorithm discover a good dynamic parameter setting itself, namely via a self-adjusting choice of λ inspired by the classic one-fifth rule (and then following the standard setting $k = \lambda/n$ and $c = 1/\lambda$).

We first design such a self-adjusting mechanisms for the $(1 + (\lambda, \lambda))$ GSEMO and then analyze the resulting runtime. We note that while dynamic parameter choise have been designed for MOEAs, e.g., [12, 32, 36], we are not aware of any such works in discrete search spaces.

To design our self-adjusting parameter setting, we first recall that the idea of the one-fifth rule is to adjust the parameters via multiplicative changes in such a way that roughly each fifth iteration is successful. Here success usually means that a strictly better solution was found. This definition makes not too much sense for multi-objective optimization, but for a GSEMO variant, it is natural to speak of *success* if the population at the end of the iteration covers more points of the Pareto front.

We then observe that asking for a success roughly each five iterations might be too much to ask for in a GSEMO variant. Since in each iterations an offspring is generated from a random parent (chosen from a population of, here, up to $n + 1$ elements), it might just take very long until a parent is chosen which has a reasonable chance to create an interesting offspring. For that reason, we rather aim at a success every roughly $5n$ iterations. With a multiplicative parameter update, this means that for a constant update strength parameter $F > 1$, we replace λ by λ/F in case of a success, and we replace λ by $\lambda F^{1/(5n-1)}$ otherwise. We set the mutation strength k and the crossover bias c depending on λ following the standard setting, that is, $k = \lambda/n$ and $c = 1/\lambda$. We allow that λ takes non-integral values and assume that values rounded to the nearest integer are used whenever integers are required. We initialize λ cautiously with $\lambda = 1$ and we ensure that λ never leaves the interval $[1, \lambda]$. See Algorithm 3 for the pseudo-code of this self-adjusting $(1 + (\lambda, \lambda))$ GSEMO.

We now conduct a runtime analysis for the self-adjusting version for our $(1 + (\lambda, \lambda))$ GSEMO and show that it can optimize the OneMinMax problem in quadratic time.

Theorem 6. *The expected optimization time of the self-adjusting $(1 + (\lambda, \lambda))$ GSEMO on OneMinMax is $O(n^2)$ when the hyper-parameter $F > 1$ is chosen sufficiently small.*

We omit the formal proof of this result for reasons of space. A main ingredient of the proof is that the population size λ evolved by the one-fifth success rule is usually not very far from the state-dependent choice λ^* analyzed in the previous section.

More precisely, we note that a λ value smaller than λ^* is not critical. Due the multiplicative update and the fact that F is a constant larger than one, it takes only $O(n\lambda^*)$ fitness evaluations to bring λ up to λ^*. This ignores the possibly finding of new points of the Pareto front; since the missing point in the definition of $o_b(P)$ is the point easiest to find, this event is actually a positive one (we found a search point with a smaller value (that is, cost of one iteration) than estimated). Taking $O(n\lambda)$ fitness evaluations to adjust λ to λ^* is uncritical, since even with the optimal value $\lambda = \lambda^*$ we allow for $O(n)$ iterations, hence $O(n\lambda^*)$ fitness evaluations, to find the desired individual.

Once we have $\lambda \geq \lambda^*$, as computed in the proof of Theorem 5, we have an $\Omega(1/n)$ probability to find the desired individual. Hence the probability to fail for γn iterations, is $\exp(-\Omega(\gamma))$. Such a sequence of failures increases the λ value by a factor of $(F^{1/(5n-1)})^{\gamma n}$. If F is chosen sufficiently small (but larger than one), then the cost incurred by this too high value of λ is outnumbered by the small probability of $\exp(-\Omega(\gamma))$ of this negative event.

Algorithm 3: The self-adjusting $(1 + (\lambda, \lambda))$ GSEMO. We always have $k = \lambda$ and $c = 1/\lambda$. We assume that λ is rounded to the nearest integer where an integer is required. Success means that the iteration has increased the population.

```
λ ← 1;
Generate x ∈ {0,1}^n uniformly at random and P ← {x};
while not stop condition do
    Uniformly at random select one individual x from P;
    Sample ℓ from a binomial distribution B(n, k/n);
    Generate x_1, x_2, ..., x_λ ∈ {0,1}^n via randomly flipping ℓ
     bits of x ;
    Select x^+, x^- ∈ {x_1, x_2, ..., x_λ} such that x^+ maximizes f_1
     and x^- maximizes f_2;
    Generate x_1^+, x_2^+, ..., x_λ^+ ∈ {0,1}^n via cross_c(x, x^+) ;
    Generate x_1^-, x_2^-, ..., x_λ^- ∈ {0,1}^n via cross_c(x, x^-);
    for y ∈ {x_1^+, x_2^+, ..., x_λ^+, x_1^-, x_2^-, ..., x_λ^-} do
        if there is no z ∈ P such that y ⪯ z then
            P = {z ∈ P | z ⋠ y} ∪ {y}
    if Success then
        λ ← max{1, λ/F}
    else
        λ ← min{n, λF^{1/(5n-1)}}
return P
```

8 EXPERIMENTS

To see if the asymptotic runtime differences of the algorithms regarded in this work are visible already for realistic problem sizes, we implemented the algorithms and ran them on the OneMinMax problem of size $n = 10, 20, \ldots, 140$.

For the $(1 + (\lambda, \lambda))$ GSEMO we use the standard parameter setting $k = \lambda = 1/c$ with $\lambda = 7 \log n$. Unfortunately, we could not find parameters that led to an interesting performance of the self-adjusting $(1 + (\lambda, \lambda))$ GSEMO.

The average runtimes of the GSEMO and $(1 + (\lambda, \lambda))$ GSEMO are displayed in Figure 1. The superiority of the $(1 + (\lambda, \lambda))$ GSEMO is clearly visible, being more than a factor of 5 faster for the largest problem sizes.

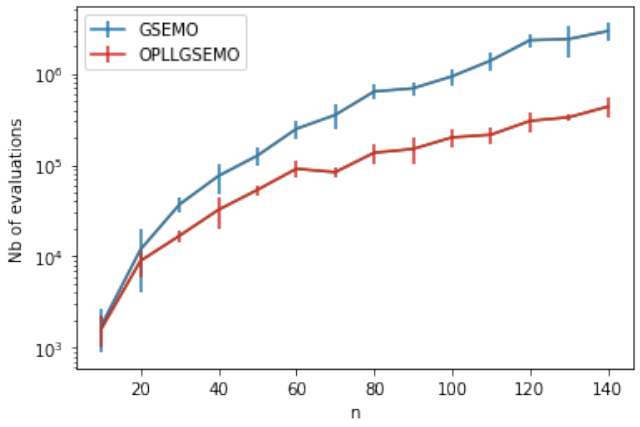

Figure 1: The mean number of function evaluations with standard deviation (in 10 independent runs) of the GSEMO and the $(1 + (\lambda, \lambda))$ GSEMO on OneMinMax.

9 CONCLUSION

In this work, we showed how to incorporate the main building block of the single-objective $(1 + (\lambda, \lambda))$ GA algorithm into a MOEA, namely the GSEMO algorithm. Our mathematical runtime analysis on the OneMinMax benchmark showed that it profits from the same speed-ups that the $(1 + (\lambda, \lambda))$ GA did on the OneMax benchmark, and this for fixed parameters, parameters depending on the spread of the population, and a self-adjusting parameter choice (where the latter two needed some modification compared to the $(1 + (\lambda, \lambda))$ GA).

Given these positive results, a natural continuation of this line of research is to see which other advantages of the $(1 + (\lambda, \lambda))$ GA transfer to the multi-objective case. Given the good performance of the $(1 + (\lambda, \lambda))$ GA on multimodal problems, an interesting next test problem could be the multimodal multi-objective benchmark designed in [23]. Given that the $(1 + (\lambda, \lambda))$ GA showed a good performance with heavy-tailed parameter choices [1, 3], both on simple and multimodal problems, seeing if these advantages continue into the multi-objective world is an equally interesting direction for future research.

From a broader perspective, this work also shows that the central idea of the $(1 + (\lambda, \lambda))$ GA can be encapsulated into new complex mutation operator, which can easily be combined with existing algorithms. We are optimistic that this idea, here only done for the GSEMO, can be profitable also for other algorithms, let it be multi-objective ones such as the NSGA-II or classic EAs.

Overall, this work shows that it can be interesting to try to transfer the recent theory developments in the better understood single-objective world into the theory of MOEAs.

ACKNOWLEDGMENTS

This work was supported by a public grant as part of the Investissements d'avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH.

REFERENCES

[1] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. 2020. Fast mutation in crossover-based algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2020*. ACM, 1268–1276.

[2] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. 2020. First steps towards a runtime analysis when starting with a good solution. In *Parallel Problem Solving From Nature, PPSN 2020, Part II*. Springer, 560–573.

[3] Denis Antipov, Maxim Buzdalov, and Benjamin Doerr. 2021. Lazy parameter tuning and control: choosing all parameters randomly from a power-law distribution. In *Genetic and Evolutionary Computation Conference, GECCO 2021*. ACM, 1115–1123.

[4] Denis Antipov and Benjamin Doerr. 2021. Precise runtime analysis for plateau functions. *ACM Transactions on Evolutionary Learning and Optimization* 1 (2021), 13:1–13:28.

[5] Denis Antipov and Benjamin Doerr. 2021. A tight runtime analysis for the $(\mu + \lambda)$ EA. *Algorithmica* 83 (2021), 1054–1095.

[6] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. 2019. A tight runtime analysis for the $(1 + (\lambda, \lambda))$ GA on LeadingOnes. In *Foundations of Genetic Algorithms, FOGA 2019*. ACM, 169–182.

[7] Denis Antipov, Benjamin Doerr, and Vitalii Karavaev. 2020. The $(1 + (\lambda, \lambda))$ GA is even faster on multimodal problems. In *Genetic and Evolutionary Computation Conference, GECCO 2020*. ACM, 1259–1267.

[8] Denis Antipov, Benjamin Doerr, and Quentin Yang. 2019. The efficiency threshold for the offspring population size of the (μ, λ) EA. In *Genetic and Evolutionary Computation Conference, GECCO 2019*. ACM, 1461–1469.

[9] Anne Auger and Benjamin Doerr (Eds.). 2011. *Theory of Randomized Search Heuristics*. World Scientific Publishing.

[10] Chao Bian and Chao Qian. 2022. Running Time Analysis of the Non-dominated Sorting Genetic Algorithm II (NSGA-II) using Binary or Stochastic Tournament Selection. *CoRR* abs/2203.11550 (2022).

[11] Dimo Brockhoff, Tobias Friedrich, and Frank Neumann. 2008. Analyzing hypervolume indicator based algorithms. In *Parallel Problem Solving from Nature, PPSN 2008*. Springer, 651–660.

[12] Dirk Büche, Sibylle D. Müller, and Petros Koumoutsakos. 2003. Self-adaptation for multi-objective evolutionary algorithms. In *Evolutionary Multi-Criterion Optimization, EMO 2003*. Springer, 267–281.

[13] Maxim Buzdalov and Benjamin Doerr. 2017. Runtime Analysis of the $(1 + (\lambda, \lambda))$ Genetic Algorithm on Random Satisfiable 3-CNF Formulas. In *Genetic and Evolutionary Computation Conference, GECCO 2017*. ACM, 1343–1350.

[14] Luc Devroye. 1972. *The compound random search*. Ph.D. dissertation, Purdue Univ., West Lafayette, IN.

[15] Benjamin Doerr. 2019. Analyzing randomized search heuristics via stochastic domination. *Theoretical Computer Science* 773 (2019), 115–137.

[16] Benjamin Doerr and Carola Doerr. 2018. Optimal static and self-adjusting parameter choices for the $(1 + (\lambda, \lambda))$ genetic algorithm. *Algorithmica* 80 (2018), 1658–1709.

[17] Benjamin Doerr and Carola Doerr. 2020. Theory of parameter control for discrete black-box optimization: provable performance gains through dynamic parameter choices. In *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, Benjamin Doerr and Frank Neumann (Eds.). Springer, 271–321. Also available at https://arxiv.org/abs/1804.05650.

[18] Benjamin Doerr, Carola Doerr, and Franziska Ebel. 2015. From black-box complexity to designing new genetic algorithms. *Theoretical Computer Science* 567 (2015), 87–104.

[19] Benjamin Doerr, Wanru Gao, and Frank Neumann. 2016. Runtime analysis of evolutionary diversity maximization for OneMinMax. In *Genetic and Evolutionary Computation Conference, GECCO 2016*. ACM, 557–564.

[20] Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. 2017. Fast genetic algorithms. In *Genetic and Evolutionary Computation Conference, GECCO 2017*. ACM, 777–784.

[21] Benjamin Doerr and Frank Neumann (Eds.). 2020. *Theory of Evolutionary Computation—Recent Developments in Discrete Optimization*. Springer. Also available at https://cs.adelaide.edu.au/~frank/papers/TheoryBook2019-selfarchived.pdf.

[22] Benjamin Doerr and Zhongdi Qu. 2022. A First Runtime Analysis of the NSGA-II on a Multimodal Problem. *CoRR* abs/2204.07637 (2022). arXiv:2204.07637

[23] Benjamin Doerr and Weijie Zheng. 2021. Theoretical analyses of multi-objective evolutionary algorithms on multi-modal objectives. In *Conference on Artificial Intelligence, AAAI 2021*. AAAI Press, 12293–12301.

[24] Stefan Droste, Thomas Jansen, and Ingo Wegener. 2002. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science* 276 (2002), 51–81.

[25] Stefan Droste, Thomas Jansen, and Ingo Wegener. 2006. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems* 39 (2006), 525–544.

[26] Mario Alejandro Hevia Fajardo and Dirk Sudholt. 2020. On the choice of the parameter control mechanism in the $(1 + (\lambda, \lambda))$ genetic algorithm. In *Genetic and Evolutionary Computation Conference, GECCO 2020*. ACM, 832–840.

[27] Josselin Garnier, Leila Kallel, and Marc Schoenauer. 1999. Rigorous hitting times for binary mutations. *Evolutionary Computation* 7 (1999), 173–203.

[28] Oliver Giel. 2003. Expected runtimes of a simple multi-objective evolutionary algorithm. In *Congress on Evolutionary Computation, CEC 2003*. IEEE, 1918–1925.

[29] Oliver Giel and Per Kristian Lehre. 2010. On the effect of populations in evolutionary multi-objective optimisation. *Evolutionary Computation* 18 (2010), 335–356.

[30] Zhengxin Huang and Yuren Zhou. 2020. Runtime analysis of somatic contiguous hypermutation operators in MOEA/D framework. In *Conference on Artificial Intelligence, AAAI 2020*. AAAI Press, 2359–2366.

[31] Zhengxin Huang, Yuren Zhou, Zefeng Chen, and Xiaoyu He. 2019. Running time analysis of MOEA/D with crossover on discrete optimization problem. In *Conference on Artificial Intelligence, AAAI 2019*. AAAI Press, 2296–2303.

[32] Christian Igel, Nikolaus Hansen, and Stefan Roth. 2007. Covariance matrix adaptation for multi-objective optimization. *Evolutionary Computation* 15 (2007), 1–28.

[33] Thomas Jansen. 2013. *Analyzing Evolutionary Algorithms – The Computer Science Perspective*. Springer.

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

[35] Thomas Jansen and Christine Zarges. 2014. Performance analysis of randomised search heuristics operating with a fixed budget. *Theoretical Computer Science* 545 (2014), 39–58.

[36] Marco Laumanns, Günter Rudolph, and Hans-Paul Schwefel. 2001. Mutation control and convergence in evolutionary multi-objective optimization. In *Proceedings of the 7th International Mendel Conference on Soft Computing, MENDEL 2001*. 24–29.

[37] Marco Laumanns, Lothar Thiele, and Eckart Zitzler. 2004. Running time analysis of multiobjective evolutionary algorithms on pseudo-Boolean functions. *IEEE Transactions on Evolutionary Computation* 8 (2004), 170–182.

[38] Marco Laumanns, Lothar Thiele, Eckart Zitzler, Emo Welzl, and Kalyanmoy Deb. 2002. Running time analysis of multi-objective evolutionary algorithms on a simple discrete optimization problem. In *Parallel Problem Solving from Nature, PPSN 2002*. Springer, 44–53.

[39] Per Kristian Lehre and Carsten Witt. 2012. Black-Box Search by Unbiased Variation. *Algorithmica* 64 (2012), 623–642.

[40] Yuan-Long Li, Yu-Ren Zhou, Zhi-Hui Zhan, and Jun Zhang. 2016. A primary theoretical study on decomposition-based multiobjective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 20 (2016), 563–576.

[41] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2019. On the time complexity of algorithm selection hyper-heuristics for multimodal optimisation. In *Conference on Artificial Intelligence, AAAI 2019*. AAAI Press, 2322–2329.

[42] Heinz Mühlenbein. 1992. How genetic algorithms really work: mutation and hillclimbing. In *Parallel Problem Solving from Nature, PPSN 1992*. Elsevier, 15–26.

[43] Frank Neumann and Carsten Witt. 2010. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Springer.

[44] Phan Trung Hai Nguyen and Dirk Sudholt. 2020. Memetic algorithms outperform evolutionary algorithms in multimodal optimisation. *Artificial Intelligence* 287 (2020), 103345.

[45] Edgar Covantes Osuna, Wanru Gao, Frank Neumann, and Dirk Sudholt. 2020. Design and analysis of diversity-based parent selection schemes for speeding up evolutionary multi-objective optimisation. *Theoretical Computer Science* 832 (2020), 123–142.

[46] Amirhossein Rajabi and Carsten Witt. 2020. Self-adjusting evolutionary algorithms for multimodal optimization. In *Genetic and Evolutionary Computation Conference, GECCO 2020*. ACM, 1314–1322.

[47] Ingo Rechenberg. 1973. *Evolutionsstrategie*. Friedrich Fromman Verlag (Günther Holzboog KG), Stuttgart.

[48] Jonathan E. Rowe and Dirk Sudholt. 2014. The choice of the offspring population size in the $(1, \lambda)$ evolutionary algorithm. *Theoretical Computer Science* 545 (2014), 20–38.

[49] Günter Rudolph. 1997. *Convergence Properties of Evolutionary Algorithms*. Verlag Dr. Kovăc.

[50] Michael A. Schumer and Kenneth Steiglitz. 1968. Adaptive step size random search. *IEEE Trans. Automat. Control* 13 (1968), 270–276.

[51] Dirk Sudholt and Carsten Witt. 2019. On the choice of the update strength in estimation-of-distribution algorithms and ant colony optimization. *Algorithmica* 81 (2019), 1450–1489.

[52] Carsten Witt. 2006. Runtime analysis of the $(\mu + 1)$ EA on simple pseudo-Boolean functions. *Evolutionary Computation* 14 (2006), 65–86.

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

[54] Weijie Zheng and Benjamin Doerr. 2022. Better approximation guarantees for the NSGA-II by using the current crowding distance. In *Genetic and Evolutionary Computation Conference, GECCO 2022*. ACM. Also available at https://arxiv.org/abs/2203.02693.

[55] Weijie Zheng, Yufei Liu, and Benjamin Doerr. 2022. A first mathematical runtime analysis of the Non-Dominated Sorting Genetic Algorithm II (NSGA-II). In *Conference on Artificial Intelligence, AAAI 2022*. AAAI Press. Preprint at https://arxiv.org/abs/2112.08581.