Does Comma Selection Help To Cope With Local Optima?

Benjamin Doerr

Does Comma Selection Help To Cope With Local Optima?

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

ABSTRACT

One hope of using non-elitism in evolutionary computation is that it aids leaving local optima. We perform a rigorous runtime analysis of a basic non-elitist evolutionary algorithm (EA), the (μ, λ) EA, on the most basic benchmark function with a local optimum, the jump function. We prove that for all reasonable values of the parameters and the problem, the expected runtime of the (μ, λ) EA is, apart from lower order terms, at least as large as the expected runtime of its elitist counterpart, the $(\mu + \lambda)$ EA (for which we conduct the first runtime analysis to allow this comparison). Consequently, the ability of the (μ, λ) EA to leave local optima to inferior solutions does not lead to a runtime advantage.

We complement this lower bound with an upper bound that, for broad ranges of the parameters, is identical to our lower bound apart from lower order terms. This is the first runtime result for a non-elitist algorithm on a multi-modal problem that is tight apart from lower order terms.

CCS CONCEPTS

• **Theory of computation** → **Theory and algorithms for application domains**; *Theory of randomized search heuristics*;

KEYWORDS

Comma selection; runtime analysis; theory

ACM Reference Format:
Benjamin Doerr. 2020. Does Comma Selection Help To Cope With Local Optima?. In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancn, Mexico.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3377930.3389823

1 INTRODUCTION

The mathematical runtime analysis of evolutionary algorithms (EAs) and other randomized search heuristics is a young but established subfield of the general research area of heuristic search [6, 27, 37, 48]. This field, naturally, has started with regarding the performance of simple algorithms on simple test problems: The problems usually were *unimodal*, that is, without local optima different from the global optimum, the algorithms were *elitist* and often had *trivial populations*, and the runtime guarantees only estimated the *asymptotic order of magnitude*, that is, gave $O(\cdot)$ upper bounds or $\Omega(\cdot)$ lower bounds.

Despite this restricted scope, many fundamental results have been obtained and our understanding of the working principles of EAs has significantly increased with these works. In this work, we go a step further with a *tight* (apart from lower order terms) analysis of how a *non-elitist* evolutionary algorithm with *both non-trivial parent and offspring populations* optimizes a *multimodal* problem.

In contrast to the practical use of evolutionary algorithms, where non-elitism is often employed, the mathematical analysis of evolutionary algorithms so far could find only little evidence for the use of non-elitism. The few existing works, very roughly speaking (see Section 2.1 for more details) indicate that when the selection pressure is large, then the non-elitist algorithm simulates an elitist one, and when the selection pressure is low, then no function with unique global optimum can be optimized efficiently. The gap between these two regimes is typically very small. Consequently, a possible profit from non-elitism would require a careful parameter choice.

One often named advantage of non-elitist algorithms is their ability to leave local optima to inferior solutions, which can reduce the time spent uselessly in local optima. To obtain a rigorous view on this possible advantage, we analyze the performance of the (μ, λ) EA on the multimodal jump function benchmark. We note that, apart from some sporadic results on custom-tailored example problems and the corollaries from very general results (see Theorems 2.1 and 2.2 further below), this is the first in-depth study of how a non-elitist algorithm optimizes a classic non-trivial class of benchmarks with local optima.

Our main result (see Section 4) is that in this setting, the small middle range between a too small and a too high selection pressure, which could be envisaged from previous works, does not exist. Rather, the two undesired regimes overlap significantly. We note that for the (μ, λ) EA, the selection pressure is reasonably well described by the ratio of the offspring population size λ to the parent population size μ. If the selection pressure is low, more precisely, if $\lambda \leq (1 - \varepsilon)e\mu$ for some constant $\varepsilon > 0$, then the (μ, λ) EA needs an exponential time to optimize any function $f : \{0, 1\}^n \to \mathbb{R}$ with at most a polynomial number of global optima [41]. If the selection pressure is high, more precisely, if $\lambda \geq (1 + \varepsilon)e\mu$ for some constant $\varepsilon > 0$ and λ is at least logarithmic in the envisaged runtime, then the (μ, λ) EA can optimize many classic benchmark functions in a

For reasons of space, many details and all proofs had to be omitted from this paper. An extended version with all details and proofs has been posted to the arxiv preprint server [20].

GECCO '20, July 8–12, 2020, Cancn, Mexico

ACM ISBN 978-1-4503-7128-5/20/07…$15.00
https://doi.org/10.1145/3377930.3389823

runtime at most a constant factor slower than, say, the $(\mu+\lambda)$ EA, see [42] and follow-up works.

Our main result implies (Corollary 4.2) that already when $\lambda \geq 2\mu$, λ is super-constant, and $\lambda = o(n^{k-1})$, the runtime of the (μ,λ) EA on all jump functions with jump size $k \leq n^{1-\varepsilon}$ is at least the runtime of the $(\mu+\lambda)$ EA (apart from lower order terms; to prove this statement, we also conduct the first so far and sufficiently tight runtime analysis of the $(\mu+\lambda)$ EA on jump functions (Theorem 3.1)). Consequently, the two regimes of a too low selection pressure and of no advantage over the elitist algorithm overlap in the range $\lambda \in [2\mu, (1-\varepsilon)e\mu]$, leaving no space for a possible middle regime with runtime advantages from non-elitism. We note that our result, while natural, is not obvious. In particular, as a comparison of the $(1,1)$ EA and the $(1+1)$ EA on highly deceptive functions shows, it is not always true that the elitist algorithm is at least as good as its non-elitist counterpart.

Our result does not generally disrecommend to use non-elitism, in particular, it does not say anything about possible other advantages from using non-elitism. Our result, however, does indicate that the ability to leave local optima to inferior solutions is hard to turn into a runtime advantage (whereas at the same time, as observed in previous works, there a significant risk that the selection pressure is too low to admit any reasonable progress).

We also prove an upper bound for the runtime of the (μ,λ) EA on jump functions (Theorem 5.1), which shows that our lower bound for large ranges of the parameters (but, of course, only for $\lambda \geq (1+\varepsilon)e\mu$) is tight including the leading constant. This appears to be the first precise[1] on runtime result for a non-trivial non-elitist algorithm on a non-trivial problem.

From the technical perspective, it is noteworthy that we obtain precise bounds in settings where the previously used methods (negative drift for lower bounds, level-based analyses of non-elitist population processes for upper bounds) could not give precise analyses, and in the case of negative drift could usually not even determine the right asymptotic order of the runtime. We are optimistic that our methods will be profitable for other runtime analyses as well.

2 STATE OF THE ART AND OUR RESULTS

This work progresses the state of the art in three directions with active research in the recent past, namely non-elitist evolutionary algorithms, precise runtime analyses, and methods to prove lower bounds in the presence of negative drift and upper bounds for non-elitist population processes. We now describe these previous states of the art and detail what is the particular progress made in this work. We concentrate ourselves on classic evolutionary algorithms (also called genetic algorithms) for the optimization in discrete search spaces. We note that non-elitism has been used in other randomized search heuristics such as the Metropolis algorithms, simulated annealing, strong-selection-weak-mutation (SSWM), and memetic algorithms. Letting the selection decisions not only depend on the fitness, e.g., in tabu search or when using fitness sharing, also introduces some form of non-elitism. From a broader perspective, also many probabilistic model building algorithms such as ant colony optimizers or estimation-of-distribution algorithms can be seen as non-elitist, since they often allow moves to inferior models. From an even broader point of view, even restart strategies can be seen as a form of non-elitism. While all these research directions are interesting, it seems to us that the results obtained there, to the extend that we understand them, are not too closely related to our results and therefore not really comparable.

2.1 Non-elitist Algorithms

While non-elitist evolutionary algorithms are used a lot in practice, the mathematical theory of EAs so far was not very successful in providing convincing evidences for the usefulness of non-elitism. This might be due to the fact that rigorous research on non-elitist algorithms has started only relatively late, caused among others by the fact that many non-elitist algorithms require non-trivial populations, which form another challenge for mathematical analyses. Another reason, naturally, could be that non-elitism is not as profitable as generally thought. Our work rather points into the latter direction.

The previous works on non-elitist algorithms can roughly be clustered as follows.

(i) Exponential runtimes when the selection pressure is low: By definition, non-elitist algorithms may lose good solutions. When this happens too frequently (low selection pressure), then the EA finds it hard to converge to good solutions, resulting in a poor performance

The first to make this empirical observation mathematically precise in a very general manner was Lehre in his remarkable work [41]. For a broad class of non-elitist population-based EAs, he gives conditions on the parameters that imply that the EA cannot optimize any pseudo-Boolean function $f : \{0,1\}^n \to \mathbb{R}$ with at most a polynomial number of optima in time sub-exponential in n. Due to their general nature, we have to restrict ourselves here to what Lehre's results imply for the (μ,λ) EA, but we note that analogous results hold for a much wider class of algorithms. For the (μ,λ) EA using the usual mutation rate $\frac{1}{n}$, Lehre shows that when $\lambda \leq (1-\varepsilon)e\mu$, where $\varepsilon > 0$ is any positive constant, then the time to find a (global) optimum of any pseudo-Boolean function with at most a polynomial number of optima is exponential in n with high probability.

We note that more specific results showing the danger of a too low selection pressure have appeared earlier. For example, already in 2007 Jägersküpper and Storch [36, Theorem 1] showed that the $(1,\lambda)$ EA with $\lambda \leq \frac{1}{14}\ln(n)$ is inefficient on any pseudo-Boolean function with a unique optimum. The range of λ for which such a negative performance is observed was later extended to the asymptotically tight value $\lambda \leq (1-\varepsilon)\log_{\frac{e}{e-1}} n$ by Rowe and Sudholt [53]. Happ, Johannsen, Klein, and Neumann [33] showed that two simple (1+1)-type hillclimbers using fitness proportionate selection in the choice of the surviving individual are not efficient on any linear function with positive weights. Neumann, Oliveto, and Witt [47] showed that a mutation-only variant of the Simple Genetic Algorithm with fitness proportionate selection is inefficient on the OneMax function when the population size μ is at most polynomial, and it is inefficient on any pseudo-Boolean function with unique global optimum when $\mu \leq \frac{1}{4}\ln(n)$. Oliveto and Witt [51] showed

[1] We use the term *precise* to denote runtime estimates that are asymptotically tight including the leading constant, that is, where the estimated runtime $\tilde{T}(n)$ and the true runtime $T(n)$ satisfy $\lim_{n\to\infty} \tilde{T}(n)/T(n) = 1$.

that the true Simple Genetic Algorithm (using crossover) cannot optimize OneMax efficiently when $\mu \leq n^{\frac{1}{4}-\varepsilon}$.

We note that the methods in [41] were also used to prove lower bounds for particular objective functions. The following result was given for jump functions [41, Theorem 5]. To be precise, a similar result was proven for a tournament-selection algorithm and it was stated that an analogous statement, which we believe to be the following, holds for the (μ, λ) EA as well.

Theorem 2.1. *Let $n \in \mathbb{Z}_{\geq 1}$, $\varepsilon > 0$ a constant, $k \leq (0.5 - \varepsilon)n$, and $k = \omega(\log n)$. The expected runtime of the (μ, λ) EA with polynomial λ on $\textsc{Jump}_{nk}$ is at least $\exp(\Omega(k))$, where all asymptotics is for n tending to infinity.*

(ii) Pseudo-elitism when the selection pressure is high: When a non-elitist algorithm has the property that, despite the theoretical chance of losing good solutions, it very rarely does so, then its optimization behavior becomes very similar to the one of an elitist algorithm. Again the first to make this effect precise for a broad class of algorithms was Lehre in his first paper on level-based arguments for non-elitist populations [42].

Lehre's level-based theorem assumes that the search space can be partitioned into levels such that (i) the non-elitist population-based algorithm has a reasonable chance to sample a solution in some level j or higher once a constant fraction of the population is at least on level $j-1$ ("base level") and (ii) there is an exponential growth of the number of individuals on levels higher than the base level; more precisely (but still simplified), if there are $\mu_0 < \gamma_0 \mu$ individuals above the base level, then in the next generation the number of individuals above the base level follows a binomial distribution with parameters μ and $p = (1+\delta)\frac{\mu_0}{\mu}$. If in addition the population sizes involved are large enough, then (again very roughly speaking) the runtime of the algorithm is at most a constant factor larger than the runtime guarantee which could be obtained for an elitist analogue of the non-elitist EA. From the assumptions made here, it cannot be excluded that the non-elitist EA loses a best-so-far solution; however, due to the exponential growth of condition (ii) and the sufficiently large population size, this can only happen if there are few individuals above the base level. Hence the assumptions of the level-based method, roughly speaking, impose that that the EA behaves like an elitist algorithm except when it just has found a new best solution. In this case, with positive probability (usually at most some constant less than one) the new solution is lost. This constant probability for losing a new best individual (and the resulting need to re-generate one) may lead to a constant-factor loss in the runtime, but not more. Very roughly speaking, one can say that such non-elitist EAs, while formally non-elitist algorithms, do nothing else than a slightly slowed-down emulation of an elitist algorithm. That said, it has to be remarked that both proving level-based theorems (see [8, 13, 22, 42]) and applying them (see also [14]) is technical and much less trivial than what the rule of thumb "high selection pressure imitates elitism" suggests.

For the optimization of jump functions via the (μ, λ) EA, the work [8] shows the following result. We note that it uses a slightly different definition of jump functions (the fitness in the gap region is uniformly zero), but from the proofs it is clear that the result also holds for the standard definition (going back to [29]) used in this work.

Theorem 2.2 ([8]). *Let $k \in [1..n]$. Let $\varepsilon > 0$ be a constant and let c be a sufficiently large constant (depending on ε). Let $\lambda \geq ck\ln(n)$ and $\mu \leq \frac{\lambda}{(1+\varepsilon)e}$. Then runtime T of the (μ, λ) EA on $\textsc{Jump}_{nk}$ satisfies $E[T] = O(n^k + n\lambda + \lambda \log \lambda)$.*

For the particular case of the $(1, \lambda)$ EA and the $(1+\lambda)$ EA, Jägersküpper and Storch in an earlier work also gave fitness-level theorems [36, Lemma 6 and 7]. They also showed that both algorithms essentially behave identical for t iterations when λ is at least logarithmic in t [36, Theorem 4]. This effect, without quantifying λ and without proof, was already proposed in [38, p. 415]. Jägersküpper and Storch show in particular that when $\lambda \geq 3\ln n$, then the $(1, \lambda)$ EA optimizes OneMax in asymptotically the same time as the $(1+\lambda)$ EA. The actual runtimes given for the $(1, \lambda)$ EA in [36] are not tight since at that time a tight analysis of the $(1+\lambda)$ EA on OneMax was still missing; however, it is clear that the arguments given in [36] also allow to transfer the tight upper bound of $O(n \log n + \lambda n \frac{\log\log n}{\log n})$ for the $(1+\lambda)$ EA from [24] to the $(1, \lambda)$ EA.

The minimum value of λ that ensures an efficient optimization was lowered to the asymptotically tight value of $\lambda \geq \log_{\frac{e}{e-1}} n \approx 2.18 \ln n$ in [53]. Again, only the bound of $O(n \log n + n\lambda)$ was shown. We would not be surprised if with similar arguments also a bound of $O(n \log n + \lambda n \frac{\log\log n}{\log n})$ could be shown, but this is less obvious here than for the result of [36]. For the benchmark function LeadingOnes, the threshold between a superpolynomial runtime of the $(1, \lambda)$ EA and a runtime asymptotically equal to the one of the $(1+\lambda)$ EA was shown to be at $\lambda = (1 \pm \varepsilon) 2 \log_{\frac{e}{e-1}} n$ [53].

(iii) Examples where non-elitism is helpful: The dichotomy described in the previous two subsections suggests that it is not easy to find examples where non-elitism is useful. This is indeed true apart from two exceptions.

Jägersküpper and Storch [36] constructed an artificial example function that is easier to optimize for the $(1, \lambda)$ EA than for the $(1+\lambda)$ EA. The Cliff function $\textsc{Cliff} : \{0,1\}^n \to \mathbb{N}$ is defined by $\textsc{Cliff}(x) = \mathrm{OM}(x)$ if $\mathrm{OM}(x) < n - \lfloor n/3 \rfloor$ and $\textsc{Cliff}(x) = \mathrm{OM}(x) - \lfloor n/3 \rfloor$ otherwise. Jägersküpper and Storch showed that the $(1, \lambda)$ EA with $\lambda \geq 5\ln n$ optimizes Cliff in an expected number of $O(\exp(5\lambda))$ fitness evaluations, whereas the $(1+\lambda)$ EA with high probability needs at least $n^{n/4}$ fitness evaluations. While this runtime difference is enormous, it has be noted that even for the best value of $\lambda = 5\ln n$, the runtime guarantee for the $(1, \lambda)$ EA is only $O(n^{25})$. Also, we remark that the local optimum of the Cliff function has a particular structure which helps to leave the local optimum: Each point on the local optimum has $\lfloor n/3 \rfloor$ neighbors from which it is easy to hill-climb to the global optimum (as long as one does not use a steepest ascent strategy). Also, for each point on the local optimum there are $\Omega(n^2)$ search points in Hamming distance two from which any local search within less than $n/3$ improvements finds the global optimum. This is a notable difference to the $\textsc{Jump}_{nk}$ function, where hill-climbing from any point of the search space that is not the global optimum or one of its n neighbors surely leads to the local optimum. We would suspect that such larger radii of attraction are closer to the structure of difficult real-world problems, but we leave it to the reader to decide which model is most relevant for their applications.

We note that a second, albeit extreme and rather academic, example for an advantage of non-elitism is implicit in the early work [31] by Garnier, Kallel, and Schoenauer. They showed that the $(1,1)$ EA on any function $f : \{0,1\}^n \to \mathbb{R}$ with unique global optimum has an expected optimization time of $(1+o(1))\frac{e}{e-1}2^n$; this follows from Proposition 3.1 in their work. When taking a highly deceptive function like the trap function, this runtime is significantly better than the ones of elitist algorithms, which typically are $n^{\Theta(n)}$. Of course, this is not overly surprising – the $(1,1)$ EA uses no form of selection and hence just performs a random walk in the search space (where the one-step distribution is given by the mutation operator). Therefore, this algorithm does not suffer from the deceptiveness of the trap function as do elitist algorithms. Also, a runtime reduction from $n^{\Theta(n)}$ to $\exp(\Theta(n))$ clearly is not breathtaking. Nevertheless, this is a second example where a (μ,λ) EA significantly outperforms the corresponding $(\mu+\lambda)$ EA.

Since in this work we are only interested in how non-elitism (and more specifically, comma selection) helps to leave local optima *and* by this improve runtimes, we do not discuss in detail other motivations for employing non-elitist algorithms. We note briefly, though, that comma selection is usually employed in self-adaptive algorithms. Self-adaptation means that some algorithm parameters are stored as part of the genome of the individuals and are subject to variation together with the original individual. The hope is that this constitutes a generic way to adjust algorithm parameters. When using plus selection together with self-adaptation, there would be the risk that the population at some point only contains individuals with unsuitable parameter values. Now variation will only generate inferior offspring. These will not be accepted and, consequently, the parameter values encoded in the genome of the individuals cannot be changed. When using comma selection, it is possible to accept individuals with inferior fitness, and these may have superior parameter values. We are not aware of a rigorous demonstration of this effect, but we note that the two runtime analysis papers [15, 28] on self-adaptation both use comma selection. We further note that comma selection is very common in continuous optimization, in particular, in evolution strategies, but since it is generally difficult to use insights from continuous optimization in discrete optimization and vice-versa we do not discuss results from continuous optimization here.

(iv) Our contribution: In Section 4, we show that for all interesting values of the parameters of the problem and the algorithm, the expected runtime of the (μ,λ) EA on jump functions is, apart from possibly lower order terms, at least the expected runtime of the $(\mu+\lambda)$ EA. This shows that for this problem, there can be no significant advantage of using comma selection. This result improves over the $\exp(\Omega(k))$ lower bound in [41] (Theorem 2.1 above).

Our upper bound in Theorem 5.1, provided mostly to show that our analysis is tight including the leading constant, improves Theorem 2.2 by making the leading constant precise and being applicable for all offspring population sizes $\lambda \geq C\ln(n)$, C a constant independent of the jump size k. To the best of our knowledge, this is the first time that the runtime of a non-elitist algorithm was proven with this precision.

2.2 Precise Runtime Analyses

Traditionally, algorithm analysis aims at gaining a rough understanding how the runtime of an algorithm depends on the problem size. As such, most result only show statements on the asymptotic order of magnitude of the runtime, that is, results in big-Oh notation. For classic algorithmics, this is justified among others by the fact that the predominant performance measure, the number of elementary operations, already ignores constant factor differences in the execution times of the elementary operations.

In evolutionary computation, where the classic performance measure is the number of fitness evaluations, this excuse for ignoring constant factors is not valid, and in fact, in the last few years more and more *precise runtime results* have appeared, that is, results which determine the runtime asymptotically precise apart from lower order terms. Such results are useful, obviously because constant factors matter in practice, but also because many effects are visible only at constant-factor scales. For example, is was shown in [21] that all $\Theta(\frac{1}{n})$ mutation rates lead to a $\Theta(n\log n)$ runtime of the $(1+1)$ EA on all pseudo-Boolean linear functions, but only Witt's seminal result [58] that the runtime is $(1+o(1))\frac{e^c}{c}n\ln n$ for the mutation rate $\frac{c}{n}$, $c>0$ a constant, allows to derive that $\frac{1}{n}$ is the asymptotically best mutation rate.

Overall, not too many non-trivial precise runtime results are known. In a very early work [31], it was shown that the $(1+1)$ EA with mutation rate $\frac{c}{n}$ optimize the OneMax function in an expected time of $(1+o(1))\frac{e^c}{c}n\ln n$ and the Needle function in time $(1+o(1))\frac{1}{1-e^e}2^n$. More than ten years later, in independent works [7, 55] the precise runtime of the $(1+1)$ EA on LeadingOnes was determined; here [7] also regarded general mutation rates and deduced from their result that the optimal mutation rate of approximately $\frac{1.59}{n}$ is higher than the usual recommendation $\frac{1}{n}$, and that a fitness dependent mutation rate gives again slightly better results (this was also the first time that a fitness dependent parameter choice was proven to be superior to static rates by at least a constant factor difference in the runtime). More precise runtime results for LeadingOnes have recently appeared in [16]. A series of recent works [1, 26, 45] obtained precise runtimes of different hyper-heuristics on LeadingOnes and thus allowed to discriminate them by their runtime. The precise expected runtime of the $(1+1)$ EA with general unbiased mutation operator on the Plateau_k function was determined [2] to be $(1+o(1))\binom{n}{k}p_{1:k}^{-1}$, where $p_{1:k}$ is the probability that the mutation operator flips between one and k bits. Apparently,
important – only the probability to flip between one and k bits has an influence on the runtime.

The only precise runtime analysis for an algorithm with a non-trivial population can be found in [32], where the runtime of the $(1+\lambda)$ EA with mutation rate $\frac{c}{n}$, c a constant, on OneMax was shown to be $(1+o(1))(\frac{e^c}{c}n\ln n + n\lambda\frac{\ln\ln\lambda}{2\ln\lambda})$. This result has the surprising implication that here the mutation rate is only important when λ is small.

The only precise runtime analysis for a multi-modal objective function was conducted in [25], where the runtime of the $(1+1)$ EA with arbitrary mutation rate was determined for jump functions; this work led to the development of a heavy-tailed mutation operator that appears to be very successful.

In summary, there is only a small number of precise runtime analyses, but many of them could obtain insights that would not have been possible with less precise analyses.

Our result, an analysis of the (μ, λ) EA on jump functions that is precise for $k \leq 0.1n$, $\lambda = o(n^{k-1})$, $\lambda \geq (1+\varepsilon)e\mu$, and $\lambda = \Omega(\log n)$ sufficiently large, is the second precise analysis for a population-based algorithm (after [32]), is the second precise analysis for a multimodal fitness function (after [25]), and is the first precise analysis for a non-elitist algorithm (apart from fact that the result [32] could be transfered to the $(1, \lambda)$ EA for large λ via the argument [36] that in this case the $(1+\lambda)$ EA and the $(1, \lambda)$ EA have essentially identical performances).

2.3 Methods: Negative Drift and Level-based Analyses

To obtain our results, we also develop new techniques for two classic topics, namely the analysis of processes showing a drift away from the target ("negative drift") and the analysis of non-elitist population processes via level-based arguments.

2.3.1 Negative Drift. It is natural that a random process $X_0, X_1, \ldots$ finds it hard to reach a certain target when the typical behavior is taking the process away from the target. *Negative drift theorems* are an established tool for the analysis of such situations. They roughly speaking state the following. Assume that the process starts at some point b or higher, that is, $X_0 \geq b$, and that we aim at reaching a target $a < b$. Assume that whenever the process is above the target value, that is, $X_t > a$, we have an expected progress $E[X_{t+1} - X_t] \geq \delta$, δ some constant, away from the target, and that this progress satisfies some concentration assumption like two-sided exponential tails. Then the expected time to reach or undershoot the target is at least exponential in the distance $b - a$.

The first such result in the context of evolutionary algorithms was shown by Oliveto and Witt [49] (note the corrigendum [50]). Improved versions were subsequently given in [40, 44, 51, 53, 59]. The comprehensive survey [43, Section 2.4.3] gives a complete coverage of this topic. What is important to note for our purposes is that (i) all existing negative drift results are quite technical to use due to the concentration assumptions, that (ii) they all give a lower bound that is only exponential in the length of the interval in which the (constant) negative drift is observed, and that (iii) they all do not give tight bounds, but only bounds of type $\exp(\Omega(b-a))$ with the implicit constant (of the exponent!) not specified.

Earlier than the general negative drift theorem, Lehre [41] proved a negative drift theorem for population-based processes via multi-type branching processes. Just as the general negative drift theorems described above, it only gives lower bounds exponential in the length of the negative drift regime and the base of the exponential function is not made explicit. Consequently, in [41, Theorem 5] (Theorem 2.1 in this work), only an $\exp(\Omega(k))$ lower bound for the runtime of the (μ, λ) EA on $\textsc{Jump}_{nk}$ was derived

Since we aim at an $\Omega(n^k)$ lower bound caused by a negative drift in the short gap region (of length k) of the jump function, and since further we aim at results that give the precise leading constant of the runtime, we cannot use these tools. We therefore resort to the *additive drift applied to a rescaled process* argument first made explicit in [5]. The basic idea is very simple: For a suitable function $g : \mathbb{R} \to \mathbb{R}$ one regards the process $(g(X_t))_t$ instead of the original process $(X_t)_t$, shows that it makes at most a slow progress towards the target, say $E[g(X_{t+1}) - g(X_t) \mid X_t > a] \geq -\delta$, and concludes from the classic additive drift theorem [35] that the expected time to reach or undershoot a when starting at b is at least $\frac{g(b)-g(a)}{\delta}$. While the basic approach is simple and natural, the non-trivial part is finding a rescaling function g which both gives at most a slow progress towards the target and gives a large difference $g(b) - g(a)$. The rescalings used in [5] and [17] were both of exponential type, that is, g was roughly speaking an exponential function. By construction, they only led to lower bounds exponential in $b - a$, and in both cases the lower bound was not tight (apart from being exponential).

Our progress: Hence the technical novelty of this work is that we devise a rescaling for our problem that (i) leads to a lower bound of order n^k for a process having negative drift only is an interval of length k, and (ii) such that these lower bounds are tight including the leading constant. Clearly, our rescalings (as all rescalings used previously) are specific to our problem. Nevertheless, they demonstrate that the rescaling method, different from the classic negative drift theorems, can give very tight lower bounds and lower bounds that are super-exponential in the length of the interval in which the negative drift is observed. We are optimistic that such rescalings will find other applications in the future.

2.3.2 Level-based Analyses. The *level-based analysis* first proposed by Lehre [42] is a general method to analyze non-elitist population-based processes. Since we gave a high-level description of this method in Section 2.1 (ii), we now explain without further explanations what is our progress over the state of the art of this method.

Similar to the state of the art in negative drift theorems, all existing variants of the level-based method do not give results that are tight including the leading constant. Also, from the complexity of the proofs of these results, it appears unlikely that such tight results can be obtained in the near future.

Our progress: For our problem of optimizing jump functions, we can exploit the fact that the most difficult, and thus time consuming, step is generating the global optimum from a population that has fully converged into the local optimum. To do so, we use the non-tight level-based methods only up to the point when the population only consists of local optima (we call this an *almost perfect population*). This can be done via a variation of the existing level-based results. From that point on, we estimate the remaining runtime by computing the waiting time for generating the optimum from a local optimum. Of course, since we are analyzing a non-elitist process, we are not guaranteed to keep an almost perfect population. For that reason, we also need to analyze the probability of losing an almost perfect population and to set up a restart argument to regain an almost perfect population. Naturally, this has to be done in a way that the total runtime spent here is only a lower-order fraction of the time needed to generate the global optimum from an almost perfect population.

A side effect of this approach is that we only need a logarithmic offspring population size, that is, it suffices to have $\lambda \geq C \ln(n)$ for some constant C that is independent of the jump size k. This is different from using the level-based method for the whole process, as

done in the proof of Theorem 2.2, which would require an offspring population size at least logarithmic in the envisaged runtime, hence here $\Omega(\log n^k) = O(k \log n)$, which is super-logarithmic when k is super-constant.

While our arguments exploit some characteristics of the jump functions, we are optimistic that they can be employed for other problems as well, in particular, when the optimization process typically contains one step that is more difficult than the remaining optimization.

3 PRELIMINARIES

In this section, we define the algorithm and the optimization problem regarded in this paper together with the most relevant works on these.

3.1 The (μ, λ) EA

The (μ, λ) EA for the maximization of pseudo-Boolean functions $f : \{0,1\}^n \to \mathbb{R}$ is made precise in Algorithm 1. It is a simple non-elitist algorithm working with a parent population of size μ and an offspring population of size $\lambda \geq \mu$. Here and in the remainder by a population we mean a multiset of individuals (elements from the search space $\{0,1\}^n$). Each offspring is generated by selecting a random parent (independently and with replacement) from the parent population and mutating it via standard bit mutation, that is, by flipping each bit independently with probability $1/n$.[2] The next parent population consist of those μ offspring which have the highest fitness (breaking ties arbitrarily).

Algorithm 1: The (μ, λ) EA to maximize a function $f : \{0,1\}^n \to \mathbb{R}$.

```
Initialize P_0 with μ individuals chosen independently and
  uniformly at random from {0,1}^n;
for t = 1, 2, ... do
    for i ∈ [1..λ] do
        Select x_i ∈ P_{t−1} uniformly at random;
        Generate y_i from x_i via standard bit mutation;
    Select P_t from the multi-set {y_1, ..., y_λ} by choosing μ
      best individuals (breaking ties arbitrarily);
```

The $(\mu + \lambda)$ EA, to which we compare the (μ, λ) EA, differs from the (μ, λ) EA only in the selection of the next generation. Whereas the (μ, λ) EA selects the next generation only from the offspring population (comma selection), the $(\mu + \lambda)$ EA selects it from the parent and offspring population (plus selection). In other words, to obtain the $(\mu + \lambda)$ EA from Algorithm 1, we only have to replace the selection by "select P_t from the multi-set $P_{t-1} \cup \{y_1, \dots, y_\lambda\}$ by choosing μ best individuals (breaking ties arbitrarily)". Often, the tie breaking is done by giving preference to offspring, but for all our purposes there is no difference.

[2] To ease the presentation, we only consider the standard mutation rate $1/n$, but we are confident that our results in an analogous fashion hold for general mutation rates χ/n, χ a constant. Previous works have shown that the constant χ has an influence (again by constant factors) on where the boundary between the "imitating elitism" and "no efficient progress" regimes is located. Since our result is that the (μ, λ) EA for no realistic parameter settings beats the $(\mu + \lambda)$ EA, we do not expect that a constant factor change of the mutation rate leads to substantially different findings.

When talking about the performance of the (μ, λ) EA or the $(\mu + \lambda)$ EA, as usual in runtime analysis [6, 27, 37, 48], we count the number of fitness evaluations until for the first time an optimal solution is evaluated. We assume that each individual is evaluated immediately after being generated. Consequently, if an optimum is generated in iteration t, then the runtime T satisfies

$$\mu + (t-1)\lambda + 1 \leq T \leq \mu + t\lambda. \tag{1}$$

Since we described the most important results on the (μ, λ) EA already in Section 2.1, let us briefly mention the most relevant results for the $(\mu + \lambda)$ EA. Again, due to the difficulties in analyzing population-based algorithms, not too much is known. The runtimes of the $(1 + \lambda)$ EA, among others on OneMax and LeadingOnes, were first analyzed in [38]. The asymptotically tight runtime on OneMax for all polynomial λ was determined in [24], together with an analysis on general linear functions. In [57], the runtime of the $(\mu + 1)$ EA on OneMax and LeadingOnes, among others, was studied. The runtime of the $(\mu + \lambda)$ EA with both non-trivial parent and offspring population sizes was determined in [3].

3.2 The Jump Function Class

To define the jump functions, we first recall that the n-dimensional *OneMax function* is defined by

$$\mathrm{OM}(x) = \|x\|_1 = \sum_{i=1}^{n} x_i$$

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

Now the n-dimensional *jump function* with *jump parameter (jump size)* $k \in [1..n]$ is defined by

$$\textsc{Jump}_{nk}(x) = \begin{cases} \|x\|_1 + k & \text{if } \|x\|_1 \in [0..n-k] \cup \{n\}, \\ n - \|x\|_1 & \text{if } \|x\|_1 \in [n-k+1\,..\,n-1]. \end{cases}$$

Hence for $k = 1$, we have a fitness landscape isomorphic to the one of OneMax, but for larger values of k there is a fitness valley ("gap")

$$G_{nk} := \{x \in \{0,1\}^n \mid n - k < \|x\|_1 < n\}$$

consisting of the $k-1$ highest sub-optimal fitness levels of the OneMax function. This valley is hard to cross for evolutionary algorithms using standard bit mutation. When using the common mutation rate $\frac{1}{n}$, the probability to generate the optimum from a parent on the local optimum is only $p_k := (1 - \frac{1}{n})^{n-k} n^{-k} < n^{-k}$. For this reason, e.g., the classic $(\mu + \lambda)$ EA has a runtime of at least n^k when k is not excessively large. This was proven formally for the $(1 + 1)$ EA in the classic paper [29], but the argument can easily be extended to all $(\mu + \lambda)$ EAs (as we do now for reasons of completeness). We also prove an upper bound, which will later turn out to agree with our lower bound for the (μ, λ) EA for large ranges of the parameters.

Theorem 3.1. *Let $\mu, \lambda \in \mathbb{Z}_{\geq 1}$. Let $n \in \mathbb{Z}_{\geq 2}$ and $k \in [2..n]$. Let $p_k := (1 - \frac{1}{n})^{n-k} n^{-k}$. Let T denote the runtime, measured by the number of fitness evaluations until the optimum is found, of the $(\mu+\lambda)$ EA on the $\textsc{Jump}_{nk}$ function.*

Lower bound: Let $h(n) := \sqrt{2n \log(\mu n)}$. If $k \leq \frac{n}{2} - h(n)$, then

$$E[T] \geq \left(1 - \frac{1}{n}\right)\left(\mu + \frac{1}{p_k}\right),$$

otherwise $E[T] \geq (1-\frac{1}{n})\left(\mu + \frac{1}{p_{k'}}\right)$ *with* $k' := \frac{n}{2} - h(n)$.

Upper bound: For all $k \in [2..n]$, *we have* $E[T] \leq \frac{1}{p_k} + O\left(n \log n + n\mu + n\lambda \frac{\log^+ \log^+(\lambda/\mu)}{\log^+(\lambda/\mu)} + (\mu+\lambda)\log\mu\right)$, *where we write* $\log^+ x := \max\{1, \ln x\}$ *for all* $x > 0$. *If* $\mu \leq \lambda$, $\lambda \leq 10^n$, *and* $\lambda = o(\frac{1}{np_k})$, *then* $E[T] \leq (1+o(1))\frac{1}{p_k}$.

For the upper bound, and assuming $k \leq \frac{n}{2} - h(n)$ in this short sketch, we simply argue that with high probability, none of the initial individuals lies in the gap or is the optimum (Chernoff bound, union bound). Consequently, none of the parents ever lies in the gap, and thus the optimum can only be generated from an individual below the gap, which happens with probability at most p_k. For the lower bound, we invoke the recent analysis of how the $(\mu+\lambda)$ EA optimizes OneMax [3] to estimate the time to find the first individual on the local optimum. With an estimate of the takeover time from [54], we estimate the time to have the whole population on the local optimum. Then again each new offspring is the optimum with probability p_k.

The above result supports our claim that $1/p_k$ is a typical runtime of an elitist mutation-based EA using standard bit mutation with mutation rate $\frac{1}{n}$. By using larger mutation rates or a heavy-tailed mutation operator, the runtime of the $(1+1)$ EA can be improved by a factor of $k^{\Theta(k)}$ [25], but the runtime remains $\Omega(n^k)$ for k constant. Asymptotically better runtimes can be achieved when using crossover, though this is not as easy as one might expect. Since these results and runtime analyses for even more distant algorithms are less relevant for this work, for reasons of space we direct the reader to the original works [4, 9–12, 19, 30, 34, 39, 46, 52, 56] or the more detailed overview in [18, Section 2.3].

4 A LOWER BOUND FOR THE RUNTIME OF THE (μ, λ) EA ON JUMP FUNCTIONS

In this section, we show our main result, a lower bound for the runtime of the (μ, λ) EA on jump functions which shows that for a large range of parameter values, the (μ, λ) EA cannot even gain a constant factor speed-up over the $(\mu+\lambda)$ EA. With its $\Omega(n^k)$ order of magnitude, our result improves significantly over the only previous result on this problem, the $\exp(\Omega(k))$ lower bound in [41] (Theorem 2.1 in this work).

Before stating the precise result, we quickly discuss two situations which, in the light of previous results, do not appear overly interesting and for which we therefore did not make an effort to fully cover them by our result.

(i) When $\lambda \leq (1-\varepsilon)e\mu$ for some constant $\varepsilon \geq 0$ and λ is at most polynomial in n, the results of Lehre [41] imply that the (μ, λ) EA has an exponential runtime on any function with a polynomial number of optima (and consequently, also on jump functions). We guess that the restriction to polynomial-size λ was made in [41, Corollary 1] only for reasons of mathematical convenience (together with the fact that super-polynomial population sizes raise some doubts on the implementability and practicability of the algorithm). We do not see any reason why Lehre's result, at least in the case of the (μ, λ) EA, should not be true for any value of λ (possibly with a sub-exponential number of iterations, but still an exponential number of fitness evaluations).

(ii) Rowe and Sudholt [53, Theorem 10] showed that for all constants $\varepsilon > 0$ the $(1, \lambda)$ EA with population size $\lambda \leq (1-\varepsilon)\log_{\frac{e}{e-1}} n$ has an expected optimization time of at least $\exp(\Omega(n^{\varepsilon/2}))$ on any function $f : \{0,1\}^n \to \mathbb{R}$ with a unique optimum. From inspecting the proof given in [53], we strongly believe that the same result also holds for the (μ, λ) EA.

Let us declare the parameter settings just discussed as not so interesting since previous works show or strongly indicate that the (μ, λ) EA is highly inefficient on any objective function with unique optimum. Let us further declare exponential population sizes as not so interesting. With this language, our following result shows that the runtime of the (μ, λ) EA on jump functions with jump size $k \leq 0.1n$ for all interesting parameter choices is, apart from lower order terms, at least the one of the $(\mu+\lambda)$ EA. For $k > 0.1n$, this runtime is at least $n^{\Omega(n)}$.

Theorem 4.1. *Let* $c \leq 0.1$ *and* C *be large enough such that* $(4c)^{C/2} \leq e^{-2}$. *Let* $n \geq \frac{2}{c}$. *Let* $C\ln(n) \leq \lambda \leq \frac{2}{3}\exp(0.16n)$ *and* $\mu \leq \frac{\lambda}{2}$. *Let* $c' = \frac{1}{e} + c$ *and* $h(n,\lambda) := \exp(-\frac{(1-2c')^2}{2}\lambda) + \frac{2n-1}{n^2-n}$. *Let* $k \in [2..n]$ *and* $p_k := (1-\frac{1}{n})^{n-k}n^{-k}$.

If $k \leq cn$, *then the expected runtime, measured by the number of fitness evaluations until the optimum is evaluated, of the* (μ, λ) *EA on jump functions with jump size* k *is at least*

$$T_k := (1-\exp(-0.16n))\left(\mu + (1-h(n,\lambda))\frac{1}{p_k}\right) = (1-o(1))(\mu + \frac{1}{p_k}),$$

where the asymptotic expression is for $n \to \infty$.

For $k > cn$, *the expected runtime is at least* $T_{\lfloor cn \rfloor}$.

We phrased our result in the above form since we felt that it captures best the most interesting aspect, namely a runtime of essentially $\frac{1}{p_k}$ when $k \leq 0.1n$ and $\lambda = \Omega(\log n)$ suitably large. Since our result is non-asymptotic, both c and C do not have to be constants. Hence if we are interested in the smallest possible value for λ that gives an $(1-o(1))\frac{1}{p_k}$ runtime, then by taking $c = \frac{k}{n}$ and $C = 4/\ln(n/(4k))$, we obtain the following result.

Corollary 4.2. *Let* $k \geq 2$. *Let* $n \geq 10k$ *and*

$$\frac{4}{\ln(\frac{n}{4k})}\ln(n) \leq \lambda \leq \frac{2}{3}\exp(0.16n).$$

Let $\mu \leq \frac{\lambda}{2}$. *Let* $c' = \frac{1}{e} + \frac{k}{n}$. *With* $h(n,\lambda) := \exp(-\frac{(1-2c')^2}{2}\lambda) + \frac{2n-1}{n^2-n}$ *and* $p_k := (1-\frac{1}{n})^{n-k}n^{-k}$, *the expected runtime of the* (μ, λ) *EA on* $\textsc{Jump}_{nk}$ *is at least*

$$T_k := (1-\exp(-0.16n))\left(\mu + (1-h(n,\lambda))\frac{1}{p_k}\right) = (1-o(1))(\mu + \frac{1}{p_k}),$$

where the asymptotic expression holds for $n \to \infty$ *and* $\lambda = \omega(1)$.

In particular, if $k = O(n^{1-\varepsilon})$ *for a constant* $\varepsilon > 0$, *then it suffices to have* $\lambda = \omega(1)$ *for the lower bound* $(1-o(1))(\mu + \frac{1}{p_k})$ *to hold.*

We briefly explain the main ideas of the proof of Theorem 4.1. As discussed earlier, this proof is an example for proving lower bounds by applying the additive drift theorem to a suitable rescaling of a natural potential function. The heart, and art, of this method is defining a suitable potential function. The observation that the difficult part of the optimization process is traversing the region $\{x \in \{0,1\}^n \mid \mathrm{OM}(x) \in [n-k..n]\}$ together with the fact that the

lower bound given by the additive drift theorem depends on the difference in potential of starting point and target suggested to us the following potential function. For a population P, let $\mathrm{OM}(P)$ denote the maximum OneMax value in the population. For $\mathrm{OM}(P) > n-k$, the potential of P will, ignoring some technical details, essentially be $\min\{n^{\mathrm{OM}(P)-(n-k)}, \frac{1}{\lambda p_k}\}$. All other populations have a potential of zero. This definition gives the desired large potential range of $\frac{1}{\lambda p_k}$ and, after proving that the expected potential gain is at most one and the initial potential is zero with high probability, gives the desired lower bound on the runtime.

5 A TIGHT UPPER BOUND

While our main target in this work was showing a lower bound that demonstrates that the (μ, λ) EA has little advantage in leaving the local optima of the jump functions, we now also present an upper bound on the runtime. It shows that our lower bound for large parts of the parameter space is tight including the leading constant. This might be the first non-trivial upper bound for a non-elitist evolutionary algorithm that is tight including the leading constant. This result also shows that our way to exploit negative drift in the lower bound analysis, namely not via the classic negative drift theorems, but via additive drift applied to an exponential rescaling, can give very precise results, unlike the previously used methods.

Theorem 5.1. *Let K be a sufficiently large constant and $\lambda \geq K \ln n$. Let $0 < \delta < 1$ be a constant and $\mu \leq \frac{1}{(1+\delta)e}\lambda$. Let $k \in [2..n]$ and $p_k = (1-\frac{1}{n})^{n-k} n^{-k}$. Then runtime T of the (μ, λ) EA on Jump_{nk} satisfies $E[T] \leq \frac{\lambda}{1-n^{-1/2}} \left(8Cn + 1 + 9\sqrt{\frac{Cn}{p_k\lambda}} + \frac{8Cn}{p_k\lambda\lfloor n^{3/2}\rfloor} + \frac{1}{p_k\lambda}\right)$, where C is a constant depending on δ only.*

Consequently, for $\lambda = o(1/np_k) = o(n^{k-1})$, we have $E[T] \leq (1 + o(1))\frac{1}{p_k}$, and for $\lambda = \Omega(1/np_k) = \Omega(n^{k-1})$, we have $E[T] = O(\lambda n)$.

We note that when $\lambda = o(n^{k-1})$, $\lambda \geq K \ln n$ with K a sufficiently large constant, $\mu \leq \frac{1}{(1+\delta)e}\lambda$, and $k \leq 0.1n$, our upper bound and our lower bound of Theorem 4.1 agree including the leading constant. So we have a precise runtime analysis in this regime.

The result in Theorem 5.1 above improves over the $O(n^k + n\lambda + \lambda \log \lambda)$ upper bound for the runtime of the (μ, λ) EA on Jump_{nk} proven in [8] (see Theorem 2.2) in three ways. First, as discussed above, we make the leading constant precise (and tight for large ranges of the parameters). Second, we obtain a better, namely at most linear, dependence of the runtime on λ. Third, we reduce the minimum offspring population size required for the result to hold, which is $\Omega(k \log n)$ in [8] and $\Omega(\log n)$ in our result.

We give a brief outline of the proof of Theorem 5.1. A central step in our proof is an analysis of how the (μ, λ) EA progresses to a parent population consisting only of individuals on the local optimum. Since the (μ, λ) EA is a non-elitist algorithm, this asks for tools like the level-based theorem first introduced by Lehre [42] and then improved by various authors [8, 13, 22, 23]. Unfortunately, all these results are formulated for the problem of finding one individual of a certain minimum quality. Consequently, they all cannot be directly employed to analyze the time needed to have the full parent population consist of individuals of at least a certain quality. Fortunately, in their proofs all previous level-based analyses proceed by analyzing the time until a certain number of individuals of a certain quality have been obtained and then building on this with an analysis on how better individuals are generated. Possibly being the one most explicit, we extend the level-based theorem of [23] to show that after $O(n)$ iterations, the whole population consists of individuals on the local optimum. We call this an *almost perfect* population. From this point on, we cannot use the level-based method anymore, since the small probability for going from the local to the global optimum would require a large value of λ, a requirement we try to avoid. This requirement is necessary in the level-based method because there one tries to ensure that once a decent number of individuals are on at least a certain level, this state is never lost. When λ is only logarithmic in n, there is an inverse-polynomial probability to completely lose a level. Since for, say, $k = \Theta(n)$, we expect a runtime of roughly n^k/λ, in this time it will regularly happen that we lose a level, including the cases that we lose a level in each of several iterations or that we lose several levels at once.

We overcome this difficulty with a restart argument. Since the probability for such an undesirable event is only inverse-polynomial in n, we see that we keep an almost perfect population for at least n^2 iterations (with high probability). Since it took us only $O(n)$ iterations to reach (or regain) an almost perfect population, we obtain that in all but a lower order fraction of the iterations we have an almost perfect parent population. Hence apart from this lower order performance loss, we can assume that we are always in an almost perfect population. From such a state, we reach the optimum in one iteration with probability $1-(1-p_k)^\lambda$, which quickly leads to the claimed result.

6 CONCLUSION

We observed that for all reasonable parameter values, the (μ, λ) EA cannot optimize jump functions faster than the $(\mu+\lambda)$ EA. The (μ, λ) EA thus fails to profit from its ability to leave local optima to inferior solutions. While we prove this absence of advantage formally only for the basic (μ, λ) EA and jump functions, we feel that our proofs do not suggest that this result is caused by very special characteristics of the (μ, λ) EA or the jump functions, but that it rather follows from the fact that if leaving a local optimum via comma selection is easy, then reaching the local optimum in the first place is very difficult. Hence this work indicates that the role of comma selection in evolutionary computation deserves some clarification. Interesting directions for future research could be to try to find convincing examples where comma selection is helpful or a general result going beyond particular examples that shows in which situations comma selection cannot speed up the optimization of multimodal objective functions. From a broader perspective, any result giving a mildly general advice which of the existing approaches to cope with local optima are preferable in which situations, would be highly desirable.[3] The new analysis methods developed in this work, which can yield precise runtime bounds for non-elitist population processes and negative drift situations, could be helpful as they now allow to prove or disprove constant-factor advantages.

[3] We note that a reviewer of this work, based on informal considerations, suggests that tournament selection should give superior results on jump functions. This would be a highly interesting result.

REFERENCES

[1] Fawaz Alanazi and Per Kristian Lehre. 2014. Runtime analysis of selection hyper-heuristics with classical learning mechanisms. In *Congress on Evolutionary Computation, CEC 2104*. IEEE, 2515–2523.

[2] Denis Antipov and Benjamin Doerr. 2018. Precise runtime analysis for plateaus. In *Parallel Problem Solving From Nature, PPSN 2018, Part II*. Springer, 117–128.

[3] Denis Antipov, Benjamin Doerr, Jiefeng Fang, and Tangi Hetet. 2018. Runtime analysis for the $(\mu + \lambda)$ EA optimizing OneMax. In *Genetic and Evolutionary Computation Conference, GECCO 2018*. ACM, 1459–1466.

[4] 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. To appear.

[5] 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.

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

[7] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. 2010. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Parallel Problem Solving from Nature, PPSN 2010*. Springer, 1–10.

[8] Dogan Corus, Duc-Cuong Dang, Anton V. Eremeev, and Per Kristian Lehre. 2018. Level-based analysis of genetic algorithms and other search processes. *IEEE Transactions on Evolutionary Computation* 22 (2018), 707–719.

[9] Dogan Corus, Pietro Simone Oliveto, and Donya Yazdani. 2017. On the runtime analysis of the Opt-IA artificial immune system. In *Genetic and Evolutionary Computation Conference, GECCO 2017*. ACM, 83–90.

[10] Dogan Corus, Pietro Simone Oliveto, and Donya Yazdani. 2018. Fast artificial immune systems. In *Parallel Problem Solving from Nature, PPSN 2018, Part II*. Springer, 67–78.

[11] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro Simone Oliveto, Dirk Sudholt, and Andrew M. Sutton. 2016. Escaping local optima with diversity mechanisms and crossover. In *Genetic and Evolutionary Computation Conference, GECCO 2016*. ACM, 645–652.

[12] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Per Kristian Lehre, Pietro Simone Oliveto, Dirk Sudholt, and Andrew M. Sutton. 2018. Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation* 22 (2018), 484–497.

[13] Duc-Cuong Dang and Per Kristian Lehre. 2016. Runtime analysis of non-elitist populations: from classical optimisation to partial information. *Algorithmica* 75 (2016), 428–461.

[14] Duc-Cuong Dang, Per Kristian Lehre, and Phan Trung Hai Nguyen. 2019. Level-based analysis of the univariate marginal distribution algorithm. *Algorithmica* 81 (2019), 668–702.

[15] Duc-Cuong Dang and Per Kristian Lehre. 2016. Self-adaptation of mutation rates in non-elitist populations. In *Parallel Problem Solving from Nature, PPSN 2016*. Springer, 803–813.

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

[17] Benjamin Doerr. 2019. An exponential lower bound for the runtime of the compact genetic algorithm on jump functions. In *Foundations of Genetic Algorithms, FOGA 2019*. ACM, 25–33.

[18] Benjamin Doerr. 2019. The Runtime of the Compact Genetic Algorithm on Jump Functions. *CoRR* abs/1908.06527 (2019). arXiv:1908.06527

[19] Benjamin Doerr. 2019. A tight runtime analysis for the cGA on jump functions: EDAs can cross fitness valleys at no extra cost. In *Genetic and Evolutionary Computation Conference, GECCO 2019*. ACM, 1488–1496.

[20] Benjamin Doerr. 2020. Does comma selection help to cope with local optima? *CoRR* abs/2004.01274 (2020). arXiv:2004.01274

[21] Benjamin Doerr and Leslie A. Goldberg. 2013. Adaptive drift analysis. *Algorithmica* 65 (2013), 224–250.

[22] Benjamin Doerr and Timo Kötzing. 2019. Multiplicative up-drift. In *Genetic and Evolutionary Computation Conference, GECCO 2019*. ACM, 1470–1478.

[23] Benjamin Doerr and Timo Kötzing. 2019. Multiplicative Up-Drift. *CoRR* abs/1806.01331 (2019). arXiv:1806.01331

[24] Benjamin Doerr and Marvin Künnemann. 2015. Optimizing linear functions with the $(1 + \lambda)$ evolutionary algorithm—different asymptotic runtimes for different instances. *Theoretical Computer Science* 561 (2015), 3–23.

[25] 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.

[26] Benjamin Doerr, Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2018. On the runtime analysis of selection hyper-heuristics with adaptive learning periods. In *Genetic and Evolutionary Computation Conference, GECCO 2018*. ACM, 1015–1022.

[27] Benjamin Doerr and Frank Neumann (Eds.). 2020. *Theory of Evolutionary Computation—Recent Developments in Discrete Optimization*. Springer.

[28] Benjamin Doerr, Carsten Witt, and Jing Yang. 2018. Runtime analysis for self-adaptive mutation rates. In *Genetic and Evolutionary Computation Conference, GECCO 2018*. ACM, 1475–1482.

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

[30] Tobias Friedrich, Timo Kötzing, Martin S. Krejca, Samadhi Nallaperuma, Frank Neumann, and Martin Schirneck. 2016. Fast building block assembly by majority vote crossover. In *Genetic and Evolutionary Computation Conference, GECCO 2016*. ACM, 661–668.

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

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

[33] Edda Happ, Daniel Johannsen, Christian Klein, and Frank Neumann. 2008. Rigorous analyses of fitness-proportional selection for optimizing linear functions. In *Genetic and Evolutionary Computation Conference, GECCO 2008*. ACM, 953–960.

[34] Václav Hasenöhrl and Andrew M. Sutton. 2018. On the runtime dynamics of the compact genetic algorithm on jump functions. In *Genetic and Evolutionary Computation Conference, GECCO 2018*. ACM, 967–974.

[35] Jun He and Xin Yao. 2001. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence* 127 (2001), 51–81.

[36] Jens Jägersküpper and Tobias Storch. 2007. When the plus strategy outperforms the comma strategy and when not. In *Foundations of Computational Intelligence, FOCI 2007*. IEEE, 25–32.

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

[38] 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.

[39] Thomas Jansen and Ingo Wegener. 2002. The analysis of evolutionary algorithms—a proof that crossover really can help. *Algorithmica* 34 (2002), 47–66.

[40] Timo Kötzing. 2016. Concentration of first hitting times under additive drift. *Algorithmica* 75 (2016), 490–506.

[41] Per Kristian Lehre. 2010. Negative drift in populations. In *Parallel Problem Solving from Nature, PPSN 2010*. Springer, 244–253.

[42] Per Kristian Lehre. 2011. Fitness-levels for non-elitist populations. In *Genetic and Evolutionary Computation Conference, GECCO 2011*. ACM, 2075–2082.

[43] Johannes Lengler. 2020. Drift analysis. In *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, Benjamin Doerr and Frank Neumann (Eds.). Springer, 89–131. Also available at https://arxiv.org/abs/1712.00964.

[44] Johannes Lengler and Angelika Steger. 2018. Drift analysis and evolutionary algorithms revisited. *Combinatorics, Probability & Computing* 27 (2018), 643–666.

[45] Andrei Lissovoi, Pietro Simone Oliveto, and John Alasdair Warwicker. 2017. On the runtime analysis of generalised selection hyper-heuristics for pseudo-Boolean optimisation. In *Genetic and Evolutionary Computation Conference, GECCO 2017*. ACM, 849–856.

[46] Andrei Lissovoi, Pietro Simone 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.

[47] Frank Neumann, Pietro Simone Oliveto, and Carsten Witt. 2009. Theoretical analysis of fitness-proportional selection: landscapes and efficiency. In *Genetic and Evolutionary Computation Conference, GECCO 2009*. ACM, 835–842.

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

[49] Pietro Simone Oliveto and Carsten Witt. 2011. Simplified drift analysis for proving lower bounds in evolutionary computation. *Algorithmica* 59 (2011), 369–386.

[50] Pietro Simone Oliveto and Carsten Witt. 2012. Erratum: Simplified Drift Analysis for Proving Lower Bounds in Evolutionary Computation. *CoRR* abs/1211.7184 (2012). arXiv:1211.7184

[51] Pietro Simone Oliveto and Carsten Witt. 2015. Improved time complexity analysis of the Simple Genetic Algorithm. *Theoretical Computer Science* 605 (2015), 21–41.

[52] Jonathan E. Rowe and Aishwaryaprajna. 2019. The benefits and limitations of voting mechanisms in evolutionary optimisation. In *Foundations of Genetic Algorithms, FOGA 2019*. ACM, 34–42.

[53] 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.

[54] Dirk Sudholt. 2009. The impact of parametrization in memetic evolutionary algorithms. *Theoretical Computer Science* 410 (2009), 2511–2528.

[55] Dirk Sudholt. 2013. A new method for lower bounds on the running time of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation* 17 (2013), 418–435.

[56] Darrell Whitley, Swetha Varadarajan, Rachel Hirsch, and Anirban Mukhopadhyay. 2018. Exploration and exploitation without mutation: solving the jump function in $\Theta(n)$ time. In *Parallel Problem Solving from Nature, PPSN 2018, Part II*. Springer, 55–66.

[57] Carsten Witt. 2006. Runtime analysis of the (μ + 1) EA on simple pseudo-Boolean functions. *Evolutionary Computation* 14 (2006), 65–86.
[58] Carsten Witt. 2013. Tight bounds on the optimization time of a randomized search heuristic on linear functions. *Combinatorics, Probability & Computing* 22 (2013), 294–318.
[59] Carsten Witt. 2019. Upper bounds on the running time of the univariate marginal distribution algorithm on OneMax. *Algorithmica* 81 (2019), 632–667.