Diversity-Preserving Exploitation of Crossover

Johannes Lengler*
ETH Zürich
Zürich, Switzerland

Tom Offermann
ETH Zürich
Zürich, Switzerland

ABSTRACT

Crossover is a powerful mechanism for generating new solutions from a given population of solutions. Crossover comes with a discrepancy in itself: on the one hand, crossover usually works best if there is enough diversity in the population; on the other hand, exploiting the benefits of crossover reduces diversity. This antagonism often makes crossover reduce its own effectiveness.

We introduce a new paradigm for utilizing crossover that reduces this antagonism, which we call *diversity-preserving exploitation of crossover (DiPEC)*. The resulting *Diversity Exploitation Genetic Algorithm (DEGA)* is able to still exploit the benefits of crossover, but preserves a much higher diversity than conventional approaches.

We demonstrate the benefits by proving that the $(2+1)$-DEGA finds the optimum of LeadingOnes with $O(n^{5/3}\log^{2/3} n)$ fitness evaluations. This is remarkable since standard genetic algorithms need $\Theta(n^2)$ evaluations, and among genetic algorithms only some artificial and specifically tailored algorithms were known to break this runtime barrier. We confirm the theoretical results by simulations. Finally, we show that the approach is not overfitted to LeadingOnes by testing it empirically on other benchmarks and showing that it is also competitive in other settings. We believe that our findings justify further systematic investigations of the DiPEC paradigm.

CCS CONCEPTS

• **Mathematics of computing** → **Evolutionary algorithms**; • **Theory of computation** → *Optimization algorithms.*

KEYWORDS

Genetic Algorithm, Runtime Analysis, Diversity, Crossover, Mutation Mask

ACM Reference Format:
Johannes Lengler and Tom Offermann. 2025. Diversity-Preserving Exploitation of Crossover. In *Foundations of Genetic Algorithms XVIII (FOGA '25), August 27–29, 2025, Leiden, Netherlands.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3729878.3746613

1 INTRODUCTION

Crossover, the idea of recombining two or more solutions into a new one, is a key ingredient of genetic algorithms [4, 36]. Crossover often works best when there is enough diversity in the population [2, 6, 11, 25, 30], see also the survey in [31]. In this case, crossover may serve as an exploitation mechanism which is able to find solutions that are fitter than the current population [4]. However, as usual for exploitation mechanism this comes at a cost for population diversity. In this paper we suggest the following new paradigm for utilizing crossover, which maintains some of the benefits of crossover without destroying so much diversity.

The new paradigm: diversity-preserving exploitation of crossover (DiPEC). For illustration, consider optimization of pseudo-Boolean functions $f : \{0,1\}^n \to \mathbb{R}$ with uniform crossover. For two parents x^1 and x^2, uniform crossover creates an offspring y by selecting each bit randomly from either x^1 or x^2. Consider the case that y replaces x^1 or x^2 in the population. Then the Hamming distance of the new pair $\{x^1, y\}$ or $\{x^2, y\}$ is in expectation only half as large as the Hamming distance of the previous pair $\{x^1, x^2\}$. Hence, diversity of this pair, measured by its Hamming distance, is roughly cut in half, which constitutes a massive loss of diversity. In this paper we will focus on population size 2, where this is the normal case, and the diversity of the whole population is cut in half. However, the same principle also applies to larger population sizes.

In order to explain the new DiPEC paradigm, let us consider the example above and rephrase it. For concreteness, assume that the result of crossover y is fitter than x^1, and let $m := x^1 \oplus y$ be the bit-wise xor of x^1 and y. A way to interpret the situation is that m is a *fitness-improving mask* for x^1: if we start from x^1 and *apply the mask* m, i.e., flip all bits in x^1 where the mask has a one-bit, then this yields the search point $x^1 \oplus m = y$ of higher fitness. So a re-interpretation of the situation is that we have found a fitness-improving mask for x^1, where the one-bits in the mask encode bit flips to x^1. The problem of this mask is that applying it destroys diversity by decreasing the Hamming distance from x^1 to x^2, roughly by a factor of 2.

The key idea is that in such a situation, the mask m can often be replaced by a smaller mask, meaning a mask that has fewer one-bits. This is based on the assumption that not all bit flips in m are equally important. Often, a small number of *critical bit flips* in m is responsible for all or most of the fitness improvement, while many other bit flips may contribute little. The key insight is that in such a case, it may be possible to find those critical bit flips efficiently by *subsampling* the mask m. I.e., we randomly compute another mask m' where we independently keep each one-bit with some probability $1/\lambda$, while everything else is a zero-bit. Then we apply the mask m' to x^1, yielding an offspring $y' := x^1 \oplus m'$. Equivalently, y' can be obtained as a *biased crossover*[1] between x^1 and y, where we take each bit from y with probability $1/\lambda$, and from x^1 with probability $1 - 1/\lambda$. Then y' has a decent chance of $1/\lambda$ to contain at least the most critical bit flip from the mask m. By repeating the biased crossover λ times, we have a good (constant) chance of generating at least one biased crossover that contains the

*Both authors contributed equally to this research.

[1]This is not related to the concept of unbiased operators introduced by Lehre and Witt [23]. The biased crossover used here is still an *unbiased* operator in their sense.

FOGA '25, August 27–29, 2025, Leiden, Netherlands

ACM ISBN 979-8-4007-1859-5/2025/08
https://doi.org/10.1145/3729878.3746613

most critical bit flip from the mask m, and hopefully the resulting offspring y' is still strictly fitter than x^1.

Crucially, the Hamming distance $H(x^1, y')$ is much smaller than the Hamming distance $H(x^1, y)$. Hence, if we replace x^1 by y' then we destroy much less diversity than if we replace x^1 by y. Quantitatively, while we lose in expectation a factor $1/2$ with y, i.e., $\mathbb{E}[H(x^2, y)] = \frac{1}{2}H(x^1, x^2)$, we only lose a factor of $1 - \frac{1}{2\lambda}$ with y', i.e., $\mathbb{E}[H(x^2, y')] = (1 - \frac{1}{2\lambda})H(x^1, x^2)$. This is a massive difference for preserving diversity, and we will show in this paper that it is often a worthwhile trade-off. Hence we suggest not to consider y for inclusion into the population, but the fittest biased crossover offspring y' out of λ trials instead. The trade-off in a good (and hopefully typical) situation looks as follows.

- The fitness of y' is almost as large as the fitness of y.
- Including y' destroys very little diversity, whereas including y would massively reduce diversity.
- This comes at the cost of $1 + \lambda$ function evaluations (of y and of λ candidates for y') instead of just 1. The cost can be reduced further, as we will discuss later.

As we will discuss later, the idea of using biased crossover to extract the most beneficial bit has been used before in the $(1+(\lambda, \lambda))$-GA [9], but in a very different situation and with different aim.

Our contribution. Apart from introducing the DiPEC paradigm, in this paper we make some first steps of exploring it. Even though the DiPEC paradigm can be applied to arbitrary population sizes, and to arbitrary pairs of search points of different fitness, we focus here on population size $\mu = 2$ and on $(2 + 1)$ algorithms. This already gives us strong effects on LeadingOnes and allows us to give rigorous proofs without too much technical overhead, but is certainly only a first step. We call the resulting algorithm the $(2 + 1)$ Diversity Exploitation Genetic Algorithm or $(2 + 1)$ DEGA, where we often omit the $(2 + 1)$ for brevity. As our main result, we analyze the $(2 + 1)$ DEGA on the LeadingOnes benchmark, see Section 3 for a definition. This is a promising benchmark for the DEGA since progress is often hard, and typically happens in small steps. This means that for every improvement there is typically a single bit flip which is responsible for a substantial part of the improvement. We show by runtime analysis that the DEGA finds the optimum with $\mathbb{E}[T] = O(\lambda n + n^2 \log n/\sqrt{\lambda})$ function evaluations.[2] For a wide range of λ, this gives a subquadratic runtime bound, in particular $\mathbb{E}[T] = O(n^{5/3}(\log n)^{2/3})$ for $\lambda = (n \log n)^{2/3}$. This is remarkable because $\Theta(n^2)$ seems a strong barrier for other genetic algorithms on LeadingOnes, except for some artificial tailored algorithms. We will discuss those in more detail below, and also explain why we do not think that the DEGA is overfit to LeadingOnes.

As a second contribution, we will describe several realizations of a $(2 + 1)$ DEGA. One is kept as simple as possible, and we use this for our runtime analysis. It is designed to make understanding the principle as easy as possible, but lacks some natural algorithmic tricks that one would apply in practice. In a second step, we will describe some such amendments. They will not make a substantial difference for the performance on LeadingOnes, but will help the algorithm to be more robust, for example in the presence of very small fitness improvements, or in cases like the Jump function where the mutation mask from uniform crossover is irreducible.

Our third contribution is to provide runtime simulations. First we investigate LeadingOnes and confirm the theoretical runtime of $\tilde{O}(n^{5/3})$, where the tilde indicates that we omit log factors. The exact value of λ does not seem to matter too much for LeadingOnes. Then we compare several versions of the $(2 + 1)$-DEGA with other basic algorithms like the $(2 + 1)$-GA, the $(1 + (\lambda, \lambda))$-GA and the UMDA on a wider class of benchmarks: OneMax, LeadingOnes, *Linear Functions* with harmonic weights and *Maximum Independent Set (MIVS)*. Not surprisingly, the DEGA is quite superior on LeadingOnes compared to the other algorithms that we test. But we find that the DEGA is also generally competitive on other benchmarks, especially for moderate $\lambda = \log n$, though here it doesn't flatly outcompete all other algorithms. We emphasize that we did not try to optimize any hyperparameters of the DEGA based on the simulation results (but neither for the other algorithms), except for showing several values of λ. This makes us confident that the DEGA can provide benefits in many situations without a general decrease in performance.

Larger population sizes. In this paper we focus on a small population size $\mu = 2$, which already requires an interesting analysis. However, a natural next step will be to extend the approach to larger population sizes. Since larger population sizes are associated with larger diversities [25, 26], it is tempting to assume that our paradigm loses strength for larger population sizes. However, we give a heuristic argument why this may not be the case.

First of all, why is larger population size associated with higher diversity? A nice answer to this question was given in [26], where the population diversity with and without crossover was analyzed in the situation *without selective pressure*, for a flat fitness function. The diversity was defined as the average Hamming distance in the population. Firstly, it was observed that crossover does not have any influence on the resulting population diversity. Secondly, the diversity at equilibrium was analyzed. This equilibrium diversity indeed increases with μ, so diversity maxes out at a larger value for larger μ. However, the assumption of a flat fitness function is crucial here. As we will argue below, exploitation (fitness improvements) decrease diversity. So the key question is not at what value the diversity maxes out if fitness improvements never occur. The relevant question is rather: if there are frequent fitness improvements that reduce diversity, *how fast* does the diversity recover? It was shown in [26] that reducing diversity can be much faster than gaining diversity. In the following, we try to quantify this effect by a thought experiment.

Consider the case of a diverse population in which one search point x^1 has a particularly important gene that the other individuals of the population do not have, for example obtained by a mutation. If x^1 was to spread its fitness advantage to the rest of the population, assume that it performs crossover with all the $\mu-1$ other individuals (not necessarily all in the same generation, but over time), each time transferring the gene and replacing the other parent with the offspring. We choose this setting, rather than a single generation, because the *average* distance of the population from the optimum is reduced by one, and the population average gives arguably a good scaling when comparing different population sizes. In this setting,

[2] We use $\log n$ for the natural logarithm with base e.

the Hamming distance of x^1 to all the new individuals is cut by half. But also, if y^2 and y^3 are the offspring replacing the parents x^2 and x^3 then their expected Hamming distance is cut by half as well $\mathbb{E}[H(y^2, y^3)] = \frac{1}{2}\mathbb{E}[H(x^2, x^3)]$. This is because for any bit in which x^2 and x^3 differ, one of them also differs from x^1, so the bit has a probability of $1/2$ to be lost during crossover. Hence, in this (admittedly simplistic) scenario, spreading a single new gene to the rest of the population costs $1/2$ of the total diversity, just as for the $\mu = 2$ case. We leave it to future work to validate or falsify this heuristic reasoning.

Related Work. There is a vast amount of work showing the benefits of crossover. We refer to [31] for a comprehensive review of theoretical work and to [11] for a discussion of more recent work, and only mention a few landmark results. Crossover can not only help for tailored benchmarks like hierarchical if-and-only-if [34] and royal-road functions [20, 35], but also for natural applications like the closest string problem [32] and for generic benchmarks like Jump [19, 37]. While a main features of all these examples are local optima that crossover helps overcoming, crossover can also speed up hillclimbing tasks like OneMax [3, 29] and LeadingOnes [2], albeit to a lesser extent.

A common theme of many of these examples is that crossover works better the larger the population diversity is. Especially the extensive theoretical work on the Jump problem [6, 11, 19, 22, 25] can be regarded as a quest for proving better guarantees for the population diversity, or for finding parameters or tweaks of the algorithms that increase diversity. In particular, explicit diversity-enhancing mechanisms have been shown to work extremely well in this situation [6]. However, the Jump function sidesteps a central dilemma of crossover: exploitation of crossover generally reduces diversity, as we have described above. For Jump, this does not play a role since the crossover step ends in the optimum, after which the algorithm stops. In practice, it seems more realistic that crossover should help many times, not just once. This is why our main focus is the LeadingOnes problem, where many equally hard steps follow each other, and crossover may be useful for each of them.

Introduced by Rudolph, the function LeadingOnes is one of the most classical theoretical benchmarks for evolutionary algorithms [28]. Even simple algorithms like random local search or the $(1+1)$-EA find the optimum with $\Theta(n^2)$ function evaluations [18]. However, this quadratic runtime bound turned out to be a sturdy barrier. Lehre and Witt could show that no unbiased[3] mutation-based algorithm can optimize LeadingOnes in less than quadratic time [23], or in other words, that the *unary unbiased black-box complexity* of LeadingOnes is $\Omega(n^2)$. Hence, for unbiased operators this barrier can only be broken by crossover and other recombination mechanisms. This was later extended by Doerr and Lengler to the $(1+1)$ elitist black-box complexity [16], which is also $\Omega(n^2)$ without the unbiasedness condition [17]. For larger population sizes, it was known that $\Omega(n^2)$ is not a principled barrier. The binary unbiased black-box complexity of LeadingOnes is $O(n \log n)$ [12], and becomes even lower for higher arities [1, 15]. In principle, the proofs in [1, 12, 15] give algorithms that optimize LeadingOnes fast. However, those algorithms are strongly tailored to LeadingOnes. Most of them are highly artificial and would immediately break when applied to a different problem. A notable exception is a basic algorithm in [12] with runtime $O(n \log n)$, which contains related ideas if viewed from the right angle. We will explain this in more detail below and also include a version of this algorithmic idea into our runtime simulation. However, the result from [12] has remained a purely theoretical result. The verbatim formulation from [12] only works for LeadingOnes, and we are not aware of attempts to turn the underlying idea into a practical general-purpose optimizer. To our best knowledge, the $\Omega(n^2)$ barrier has not been broken by any practical population-based algorithm.[4] In particular, it has been explicitly shown by Cerf and Lengler that diversity-enhancing mechanisms can improve the runtime of the standard $(2+1)$ GA by a constant factor, but not beyond the regime of $\Theta(n^2)$ [2]. However, the $\Theta(n^2)$ barrier *has* been broken by the significance-based cGA [13], which optimized LeadingOnes in time $O(n \log n)$. Despite its name, this algorithm is not a genetic algorithm and is not based on populations at all, but it is rather an Estimation-of-Distribution Algorithm which works with statistical properties of the fitness landscape. Together with our work, this may be an additional indication that population-based algorithms may still not have reached their full potential.

The idea of diversity-preserving exploitation of crossover bears some resemblance to the idea used in the $(1+(\lambda,\lambda))$-GA [9]. In that algorithm, λ mutations of the same parent x are generated, then the fittest y of those λ offspring is selected (which is usually less fit than the parent), and then λ biased crossover between x and y are performed. This uses the same of idea of hoping that a biased crossover retains the most beneficial mutation. However, the idea is applied in a very different situation and with a very different aim. In the $(1+(\lambda,\lambda))$-GA, y is a mutation of x that is typically less fit, and the aim is to find a new search point that is fitter than both x and y. In our paradigm, y is fitter than x, and we accept that the biased crossover may be less fit than y, as long as it is fitter than x and has low Hamming distance of x to preserve population diversity. Not surprisingly, practically the differences to the $(1+(\lambda,\lambda))$-GA are very large, for example that it works with population size one.

Algorithms design from theoretical benchmarks. Traditionally, the theory community in the field is very cautious with suggesting new algorithms from evidence on theoretical benchmarks. This has good reasons: there is a substantial risk of overfitting the algorithm to the benchmark. It is rather easy to write down an algorithm which works well on one specific benchmark like LeadingOnes, but which fails on other benchmarks. Therefore, most runtime analysis has concentrated on understanding algorithms that are already established in practical work, thus avoiding the overfitting problem. The community thus expects high standards of algorithmic suggestions that stem from runtime analyses. We propose an explicit formulation of such standards through the following conditions.

- *Generality:* The underlying mechanism is general enough to plausibly work in a wide range of situations.

[3] Despite its name our crossover operator is unbiased in the sense of Lehre and Witt.

[4] This is to be understood for the class of all automorphic transformations of LeadingOnes, which allow different target strings and different orderings of the positions. Our runtime bounds of $\tilde{O}(n^{5/3})$ holds for this whole class. There are certainly algorithms which can optimize the LeadingOnes function itself in sub-quadratic time, for example, by starting with the all-ones string, or by testing the bits in sequential order. However, those algorithm do not transfer to automorphic transformations of LeadingOnes.

- *Transparency:* It must be clear in which situation the mechanism gives improvement, and the reason why it gives improvements must be well-understood.
- *Impact:* At least on one benchmark the improvement is substantial, and this should be clearly demonstrated by proof or by experiment. Ideally, the benchmark is not specifically designed for the problem, but has been studied before.

Despite the general caution, there are several algorithmic ideas that have made their way from theory to algorithmic portfolios. This includes the $(1+(\lambda,\lambda))$-GA [9], heavy-tailed mutation operators as in the so-called *fast EAs* [14] and self-adjustment schemes like the one-fifth rule for discrete search spaces [10]. We believe that our results for the diversity-preserving exploitation of crossover introduced in this paper meets those standards as well, and that this justifies further systematic investigations of this paradigm.

2 THE DEGA ALGORITHM

As announced, we first describe a simple version of the algorithm that we will use for the runtime analysis on LeadingOnes. This is designed to be a minimal algorithm to show improvement, with as few components as possible and simpler than the general DiPEC framework that we have described above.[5] In a second step we then list extensions that are likely beneficial for making the algorithm more robust and more widely applicable.

The simple $(2+1)$-DEGA is described in Algorithm 1. It starts with two random antipodal search points.[6] When the two search points in the population have the same fitness, the algorithm simply uses mutation to either find a better search point or to increase diversity, lines 6–7. The operator "mutate" in line 6 is standard bit mutation with mutation rate $1/n$, i.e. it flips each bit independently with probability $1/n$. The "SelectPopulation" operator in line 7 selects for fitness, with diversity as a secondary criterion. In other words, the offspring y is always selected if it is strictly fitter than the two (equally fit) old search points x^1, x^2, and always discarded when it is strictly less fit. When y has the same fitness as x^1 and x^2, the new population is the pair of points with largest Hamming distance among those 3 points, with ties broken randomly.

Whenever the two search points have different fitnesses then the algorithm uses the DiPEC paradigm to replace the less fit x^1 by a biased crossover offspring y that has higher fitness, but small Hamming distance from x^1, thus preserving most of the population diversity (which here is simply the Hamming distance from x^2). This is done by the operator Crossover$(x^1, x^2, 1/\lambda)$, which produces an offspring by taking every bit from x^2 with probability $1/\lambda$, and taking all other bits from x^1. For LeadingOnes we can implement the DiPEC paradigm in a simplified way compared to the introduction, lines 12–15 of Algorithm 1. We describe a full version later that works on other functions as well. Since on LeadingOnes we eventually find a crossover offspring as desired, we can simply perform biased crossover until we find such an offspring. Secondly, we directly perform a biased crossover of x^1 and x^2 in line 13, which selects each bit from x^1 with probability $1-1/\lambda$ and from x^2 with probability $1/\lambda$. The alternative is to go through an intermediate unbiased crossover y and check first whether that is fitter than the less fit parent. This step can be skipped for LeadingOnes since there it always succeeds with large probability $1/2$, but it should not be skipped in the general-purpose version of this algorithm, see below.

Algorithm 1: $(2+1)$-**DEGA**(n,λ) for maximizing $f : \{0,1\}^n \to \mathbb{R}$

1: $x \leftarrow \text{Uniform}(n, \frac{1}{2})$
2: $P \leftarrow \{x, \bar{x}\}$
3: **repeat**
4: $\quad x^1, x^2 \leftarrow P$
5: $\quad$ **if** $f(x^1) = f(x^2)$ **then** ▷ Enhance Diversity
6: $\quad\quad y = \text{mutate}(x^i)$, $i \in \{1,2\}$ random
7: $\quad\quad P \leftarrow \text{SelectPopulation}(x^1, x^2, y)$
8: $\quad$ **else** ▷ Exploitation
9: $\quad\quad$ **if** $f(x^1) > f(x^2)$ **then**
10: $\quad\quad\quad$ swap x^1, x^2 ▷ Ensure $f(x^1) < f(x^2)$
11: $\quad\quad$ **end if**
12: $\quad\quad$ **repeat**
13: $\quad\quad\quad y = \text{Crossover}(x^1, x^2, 1/\lambda)$ ▷ Biased Crossover
14: $\quad\quad$ **until** $f(y) > f(x^1)$
15: $\quad\quad x^1 \leftarrow y$
16: $\quad$ **end if**
17: **until** termination criterion met

In our analysis, we will split the run of the algorithm into *diversity phases* for $f(x^1) = f(x^2)$ and *exploitation phases* for $f(x^1) \neq f(x^2)$. During a diversity phase the $(2+1)$-DEGA uses mutation, and such phase lasts until the first strict improvement is found. Note that selection favors diversity if there are fitness ties, measured by the Hamming distance of the two search points. Hence, the diversity can only increase during a diversity phase. An exploitation phase lasts until two search points of equal fitness are found, and no mutation is used in such a phase. Instead, for the less fit search point, which we assume to be x^1, the algorithm tries to find a fitter search point z close to x^1 by biased crossover. Hence, the diversity can only *decrease* during an exploitation phase. Note that it can happen that x^1 improves so much that it becomes fitter than x^2, in which case the exploitation phase continues. In fact, we will show that for LeadingOnes this is the most common case, and that a single exploitation phase consists of many improvements of both search points before reaching a population of equal fitness. Thus a diversity phase may consist of many runs through lines 5–7, and an exploitation phase may consist of many runs through lines 9–15.

Amendments. As previously described, the previous algorithm is designed to be a minimal example that is efficient on LeadingOnes. However, there are several extensions that are natural to make the algorithm robust and avoid obvious failure modes on other benchmarks. We have already mentioned that in general, we would not let the loop in line 12–14 run indefinitely, but rather terminate it after a while, for example after at most λ steps. Before even entering the loop, we should not skip the step of creating an *unbiased* crossover y of x^1 and x^2 and checking whether it is fitter

[5] We believe that it is generally beneficial to investigate such simplified algorithms. Even if the algorithms in practice contain a lot of additional components, this strategy makes it more transparent which components are responsible for the runtime improvement. The alternative is to use a complex algorithm together with extensive ablation studies.
[6] It seems natural for the algorithm to start with maximal diversity. It simplifies the analysis, but we don't believe that it is crucial for the runtime result.

than x^1. Only in that case should we start creating biased crossover y' between y and x^1. In fact, this idea can be iterated in order to reduce the cost of finding one offspring with the DiPEC paradigm from $\lambda + 1$ to $O(\log \lambda)$. We describe this method as algorithm A_{BB} in Section 5. However, while this algorithm works well on LeadingOnes, it does not show good performances on other benchmarks.

Apart from that, the decision to *exclusively* use crossover in an exploitation phase works for LeadingOnes, but bears the risk of getting stuck when biased crossover does not work. Conversely, for general fitness landscapes crossover may also be beneficial if both parents have the same fitness. Hence, for a practical implementation we suggest that in any case the algorithm performs randomly either a mutation or a uniform crossover (or a combination of both). In case of a crossover y, the algorithm enters the exploitation phase only if y is strictly fitter than the less fit offspring x^1 (where ties between x^1 and x^2 are broken randomly), in which case it performs λ biased crossover between x^1 and y. We consider this the most crucial amendment to turn the $(2 + 1)$-DEGA into a practical algorithm. We do not believe that this would change the runtime on LeadingOnes, but it would make the analysis more complicated.

There are some other extensions which seem natural, but we leave their exploration for future work. For continuous optimization, but also for discrete optimization with many fitness levels, the condition $f(x^1) = f(x^2)$ may not be satisfied often. Hence, it may make sense to replace this by a soft condition, for example that the difference $|f(x^1) - f(x^2)|$ is below its average value over the last n generations. Likewise, the criterion $f(z) > f(x^1)$ for replacing x^1 in line 15 may be too soft in such cases. Every replacement costs diversity, so we want to avoid replacements with negligible fitness improvements. Hence, we may replace the condition by a stronger one, for example that z must retain at least 10% of the fitness advantage that x^2 has over x^1.

Since it is not clear a priori how to choose the parameter λ, it would also be natural to adapt this parameter dynamically, for example by starting for each new population with a large $\lambda = \lambda_0$ (e.g., $\lambda_0 = n^{2/3}$), but if after λ biased offspring no improvement of x^1 was found, then λ is reduced by a factor of $1/2$. Mind that a smaller λ increases the chances of keeping important bits from x^2, but costs more diversity. The whole scheme increases the runtime only by a constant factor of roughly 2, but removes a parameter choice. Moreover, this variant also has a chance of eventually accepting an unbiased crossover and thus be more efficient on benchmarks like Jump, where uniform crossover is beneficial over biased crossover.

Finally, the algorithm extends naturally to larger populations. In this case, we would choose two random parents for crossover, either uniformly or by methods like tournament selection. Then uniform crossover is performed, and if the offspring y is fitter than the worse parent x then λ biased offspring between x and y are performed, the best of which may replace x. Instead, one could also replace an unrelated third search point x', e.g., the least fit in the population, by the fittest of $H(y, x')$ biased crossovers between y and x'. We leave exploration of larger populations to future work.

3 PRELIMINARIES

The main result of this paper is on the LeadingOnes benchmark. The function LeadingOnes or LO : $\{0,1\}^n \to \{0,\dots,n\}$ counts the number of initial consecutive 1s in a bit string $x \in \{0,1\}^n$, formally defined as $\text{LO}(x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$.

The *runtime* T on LeadingOnes is the number of function evaluations until the optimum is found. We measure the *time* generally by the number of fitness evaluations and define a *generation* as the time period of creating and evaluating one search point (either in line 6 or in line 13 of Algorithm 1). As explained in Section 2, a run of the $(2 + 1)$-DEGA on LeadingOnes alternates between diversity and exploitation phases, where the latter can cover many fitness improvements. Starting with $k = 0$ we denote by $P_k = \{x_k^1, x_k^2\}$ the population at the beginning of the k-th phase, where $\text{LO}(x_k^1) \le \text{LO}(x_k^2)$. The i-th bit of x_k^1 is denoted as $(x_k^1)_i$, and similarly for other search points. Sometimes we also want to refer to variables at some time t within a phase, and by overlay of notation we denote this by an index t, for example P_t is the population in generation t. We will use the indices in a way that no confusion between the two options should arise.

At any point during a run we split the n positions into an *optimized* part and a *non-optimized* part. The optimized part is given by the first $\text{LO}(x^2) = \max\{\text{LO}(x^1), \text{LO}(x^2)\}$ bits (i.e., optimized by the fitter search point x^2). The non-optimized part NO is given by the remaining bits, but in a diversity phase (if both search points have the same fitness) we exclude the bit at position $\text{LO}(x^2) + 1$, which we we call the *critical bit*. Hence, $\text{NO} = \{\text{LO}(x^2)+1, \dots, n\}$ in an exploitation phase and $\text{NO} = \{\text{LO}(x^2) + 2, \dots, n\}$ in a diversity phase. Throughout the paper we denote by $\alpha := |\text{NO}|/n$ the fraction of the non-optimized part in the total string. As before, α_k is the value of α at the beginning of the k-th phase. We will focus specifically at the Hamming distance H_k between the non-optimized parts of our two search points at the beginning of the k-th phase, as well as its complement,

$$H_k := \sum_{i \in \text{NO}} (x_k^1)_i \oplus (x_k^2)_i \quad \text{and} \quad \bar{H}_k := \alpha_k n - H_k,$$

where $\oplus$ denotes XOR. Note that the fraction α of the non-optimized part can only decrease over the course of a run.

Finally we introduce some notation for different bits. We have already defined the *critical bit* at position $\text{LO}(x^2)+1$, where $\text{LO}(x^2) \ge \text{LO}(x^1)$. Note that this bit is always zero in x^2. In a diversity phase, since $\text{LO}(x^1) = \text{LO}(x^2)$ the critical bit is also zero in x^1. In an exploitation phase, the position of the first zero-bit in x^1 is called the *improving position* or *improving bit*. Note that this is a one-bit in x^2. In the non-optimized part of size αn, a *blocking bit* is a bit that is zero in both x^1 and x^2, and a *skipping bit* is a bit that is one in both x^1 and x^2. These bits are also illustrated in Figure 1. We denote the number of blocking and skipping bits at the beginning of the k-th phase with B_k and S_k respectively. Note that $\bar{H}_k = B_k + S_k$.

Due to the symmetries of LeadingOnes as discussed in [2], for a given Hamming distance of x^1 and x^2 in the non-optimized part, $\bar{H}_k$ positions in NO in which x^1 and x^2 coincide are located uniformly at random. As the following Lemma 3.1 confirms, this includes the critical bit during an exploitation phase. Finally, a fitness improvement of x^2 increases the fitness not just by one, but by $1+\mathcal{G}(1/2)$, where $\mathcal{G}$ denotes a (truncated) geometric distribution. A more detailed discussion and a proof of Lemma 3.1 are in the full version [24]. In a nutshell, the reason for the symmetry is that none of these bits ever affect selection. For the critical bit of x^1 during

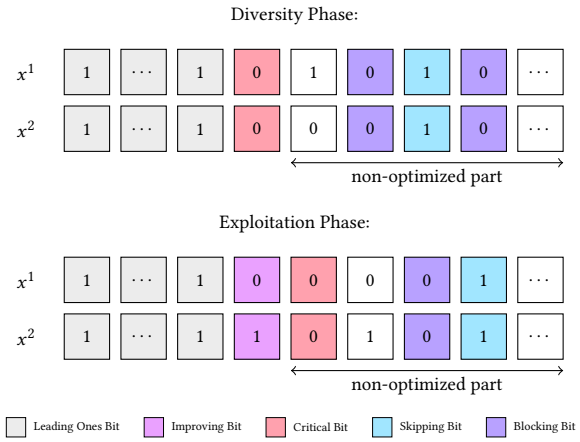

Figure 1: Different bits during optimization

an exploitation phase, it may affect the *fitness* of a search point, but will never be the reason that a search point is accepted or rejected.

Lemma 3.1. *For a run of the DEGA on LeadingOnes and for $i \in [n]$, let t_1 be the first generation in which $LO(x_t^2) = i$ (assuming this event happens), and let $t_2 + 1$ be the first generation in which $LO(x_t^1) \geq i$. Let NO be the non-optimized part in generation t_2, and let H_{t_2+1} be the Hamming distance of $x^1_{t_2+1}$ and $x^2_{t_2+1}$ in NO.*[7] *Then the probability that in generation $t_2 + 1$ the $(i+1)$-th bit differs in x^1 and x^2 is*

$$\Pr\left[(x^1_{t_2+1})_{i+1} \neq (x^1_{t_2+1})_{i+1}\right] = \frac{H_{t_2+1}}{|NO|}. \tag{1}$$

Note that at time $t_2 + 1$ the old search point $x^2_{t_2}$ stays in the population and becomes the new search point $x^1_{t_2+1}$, because it is now at most as fit as the offspring y_{t_2} generated in generation t_2. In case of equal fitness it could be either labeled $x^1_{t_2+1}$ or $x^2_{t_2+1}$, but let us assume the former for consistency. In any case, this search point has a zero-bit at position $i + 1$. Hence, the event in (1) could equivalently be phrased as "$(x^2_{t_2+1})_{i+1} = 1$" or as "$\mathrm{LO}(x^2_{t_2+1}) \geq i+1$" or as "the population P_{t_2+1} contains a one-bit at position $i + 1$".

4 THE $(2 + 1)$-DEGA ON LEADINGONES

Our main result is the following upper bound on the expected runtime $\mathbb{E}[T]$ of the $(2 + 1)$-DEGA on LeadingOnes.

Theorem 4.1. *Let T be the runtime of the $(2 + 1)$-DEGA with parameter $2 \leq \lambda = o(n)$ on LeadingOnes. Then*

$$\mathbb{E}[T] = O\big(\lambda n + n^2 \log n/\sqrt{\lambda}\big). \tag{2}$$

Since the first summand increases and the second decreases in λ, their sum is asymptotically minimal when both terms are equal. This holds for $\lambda = (n \log n)^{2/3}$, which gives

$$\mathbb{E}[T] = O\big(n^{5/3} \log^{2/3} n\big). \tag{3}$$

[7] The slight mismatch of indices in the definition of NO and H_t is needed for a correct statement. It will not play a role later since H_t does not change too much in a single generation.

4.1 Proof of Theorem 4.1

The rest of this section is devoted to proving Theorem 4.1, and throughout the section we will consider the $(2 + 1)$-DEGA with $2 \leq \lambda = o(n)$ on LeadingOnes. Throughout the analysis, $P = \{x^1, x^2\}$ denotes the current population of the algorithm, where $\mathrm{LO}(x^1) \leq \mathrm{LO}(x^2)$. The key ingredient is to show that the diversity remains close to its maximum, or equivalently that its complement $\bar{H}_t$ in the non-optimized part remains bounded. This is formalized by the following theorem. Recall that $\alpha \in [0, 1]$ denotes the non-optimized fraction of the string and that we overload the index of α: for the k-th phase we refer by α_k to the value of α at the beginning of phase k, and for the t-th generation ("at time t") we denote its value by α_t. We omit the index altogether if the considered point in time is clear from the context.

Theorem 4.2. *There is a constant $c > 0$ such that for all times t with $\alpha_t \geq n^{-1/3}$,*

$$\mathbb{E}\left[\bar{H}_{t+1}\right] \leq c \cdot \alpha_t n \log n/\sqrt{\lambda}.$$

We first show how Theorem 4.2 implies Theorem 4.1.

Proof of Theorem 4.1. We will prove the seemingly weaker bound $\mathbb{E}[T] = O(n^{5/3} + \lambda n + n^2 \log n/\sqrt{\lambda})$. Note that the additional term $n^{5/3}$ does nothing: either $\lambda \geq n^{2/3}$ and $n^{5/3}$ is dominated by λn; or $\lambda \leq n^{2/3}$ and $n^{5/3}$ is dominated by the last term.

In our analysis we will focus on the part with $\alpha_t \geq n^{-1/3}$, or equivalently when the non-optimized part has size $\geq n^{2/3}$. The last part with $\alpha_t < n^{-1/3}$ is negligible: we will show below in Lemma 4.3 that the algorithm needs in expectation $O(\lambda)$ function evaluations to improve x^1 when $\mathrm{LO}(x^1) \neq \mathrm{LO}(x^2)$, and it needs in expectation $O(n)$ evaluations to find a fitness improvement when $\mathrm{LO}(x^1) = \mathrm{LO}(x^2)$. The former can happen at most n times, the latter at most $n^{2/3}$ times when $\alpha_t < n^{-1/3}$. Hence, the time spent for the last interval is at most $O(\lambda n + n^{5/3})$, as required.[8] Moreover, the same argument shows that all exploitation phases together (of the whole run, not just the last interval) take time $O(\lambda n)$, and each diversity phase takes time $O(n)$. Hence, it will suffice to show that for the remaining part with $\alpha_t > n^{-1/3}$, there are at most $O(n \log n/\sqrt{\lambda})$ diversity phases in expectation.

Recall the notion of blocking bits from Section 3. We define $\mathcal{B}_i$ for $1 \leq i \leq n - n^{2/3}$ as the event that the algorithm enters the diversity phase with $\mathrm{LO}(x^1) = \mathrm{LO}(x^2) = i$. Assume that $\mathrm{LO}(x^1) < \mathrm{LO}(x^2) = i$ at some point, because otherwise $\mathcal{B}_i$ cannot occur. Then consider the unique point in time t when $\mathrm{LO}(x^1_t) < \mathrm{LO}(x^2_t) = i$ and an offspring y is found with $\mathrm{LO}(y) \geq i$. Then the $\bar{H}_{t+1}$ identical bits of x^2_t and y among the last $\alpha_t n$ positions are distributed uniformly at random by Lemma 3.1 and [2, Corollary 7]. In particular, by Lemma 3.1, for a given value of H_{t+1} the probability that position $i + 1$ is one of those positions is $\bar{H}_{t+1}/(\alpha_t n)$. Since x_2 has a zero-bit at position $i + 1$, this is equivalent to the event $\mathcal{B}_i$, and thus

$$\Pr[\mathcal{B}_i \mid \bar{H}_{t+1}] \leq \frac{\bar{H}_{t+1}}{\alpha_t n}, \tag{4}$$

where the bound would be an equality if we conditioned on the fitness hitting level i at some point during the run. By the law of

[8] The analysis of this part could easily be improved, but improvements here would not yield a better bound since the bottlenecks are elsewhere.

total expectation and by Theorem 4.2,

$$\Pr[\mathcal{B}_i] = \mathbb{E}\left[\Pr[\mathcal{B}_i \mid \bar{H}_{t+1}]\right] \leq \frac{\mathbb{E}[\bar{H}_{t+1}]}{\alpha_t n} \leq c \cdot \log n/\sqrt{\lambda}.$$

Thus the expected number of diversity phases with $\alpha_t > n^{-1/3}$ is $\sum_{i=1}^{n-n^{2/3}} \Pr[\mathcal{B}_i] \leq cn \log n/\sqrt{\lambda}$. This concludes the proof. □

In the proof of Theorem 4.1 we used Theorem 4.2 and also the following lemma, which is proven in the full version [24].

Lemma 4.3. *Assume $LO(x_t^1) < LO(x_t^2)$ for the current population $P_t = \{x_t^1, x_t^2\}$ of the DEGA on LeadingOnes. Let $T_{Improve}$ be the number of LO-evaluations until x^1 is replaced by a fitter search point. Then $\mathbb{E}\left[T_{Improve}\right] = O(\lambda)$.*

4.2 Proof of Theorem 4.2

It remains to prove Theorem 4.2, and this whole section is devoted to that. Recall that $\bar{H}_k$ is the number of bits that align in the non-optimized part of x^1, x^2 at the start of phase k. We start by proving two lemmas which analyze how the diversity changes in a diversity and in an exploitation phase, respectively. In the following, we frequently use these inequalities:

$$1 - x \leq \exp(-x) \quad \text{and} \quad 1 - x \geq \exp(-2x), x \in [0, 1/2]. \tag{5}$$

Combining both gives $(1-x)^r \leq \exp(-xr) \leq 1-xr/2$ if $x \in [0, 1/2]$, $r \geq 0$ and $xr \leq 1$.

Lemma 4.4. *For $k \geq 1$, assume that the k-th phase is a diversity phase with $\alpha_k n \geq n^{2/3}$. Then*

$$\mathbb{E}\left[\bar{H}_{k+1} \mid \bar{H}_k\right] \in \left[\tfrac{1}{2}\bar{H}_k - 1, \left(1 - \tfrac{1}{4e^2}\right)\bar{H}_k\right].$$

Proof. We first prove the lower bound. We may neglect any mutation that flips the optimized part, since the offspring will be rejected. Among the remaining mutations, we compare the critical bit i with any fixed non-optimized bit j. By symmetry, both are equally likely to be flipped before the other. Hence, the probability that bit j is flipped strictly before the first improving mutation is at most $1/2$. So before that mutation, $\bar{H}_k$ decreases by at most a factor of $1/2$ in expectation. Finally, in the improving mutation, in expectation at most one additional bit is flipped.

For the upper bound, let $c_2 = 1 - 1/(4e^2)$. For a fixed generation, let $p_{k,\text{out}}$ be the probability of leaving phase k. Then

$$p_{k,\text{out}} = \tfrac{1}{n}\left(1 - \tfrac{1}{n}\right)^{n-\alpha n-1} \leq \tfrac{1}{n}.$$

Let Y_k be the number of LO-evaluations until leaving phase k. For $y = n/2$ we calculate

$$\Pr[Y_k \geq y] \geq \left(1 - p_{k,\text{out}}\right)^y \overset{(5)}{\geq} \exp\left(-2p_{k,\text{out}}y\right) \geq 1/e,$$

where the second step holds for $n \geq 2$. To calculate the reduction in $\bar{H}_t$, let us consider the event $\mathcal{F}_i$ of flipping bit i in a one-bit flip in phase k; that is, there is a mutation which flips bit i and no other bit. The probability that a fixed mutation is of this type is at least $\frac{1}{n}(1-\frac{1}{n})^{n-1} \geq \frac{1}{en}$. Moreover, by law of total probability, we have

$$\Pr[\mathcal{F}_i] \geq \Pr[\mathcal{F}_i \mid Y_k \geq y] \cdot \Pr[Y_k \geq y]. \tag{6}$$

Let us analyze $\Pr[\mathcal{F}_i \mid Y_k \geq y]$. Since more than y trials only increase the chance of $\mathcal{F}_i$, we get

$$\Pr[\mathcal{F}_i \mid Y_k \geq y] \geq 1 - \left(1 - \tfrac{1}{en}\right)^y \overset{(5)}{\geq} 1 - \exp(-\tfrac{1}{2e}) \overset{(5)}{\geq} \tfrac{1}{4e}.$$

Plugging our finding back into Equation (6) gives $\Pr[\mathcal{F}_i] \geq 1/(4e^2)$. Thus each one-bit in $\bar{H}_k$ has a chance of at least $1/(4e^2)$ to vanish. We conclude that $\mathbb{E}\left[\bar{H}_{k+1}\right] \leq (1 - 1/(4e^2)) \cdot \bar{H}_k$, as required. □

We now analyze how $\bar{H}_k$ develops during an exploitation phase. We first show in Lemma 4.5 that with high probability the progress per exploitation phase is at most $O(\sqrt{\lambda}\log n)$, where we measure progress as the fitness increase of the fitter of the two individuals. Afterwards, we will derive an upper bound on the increase from $\bar{H}_k$ to $\bar{H}_{k+1}$ for an exploitation phase k in Lemma 4.6.

Lemma 4.5. *Let $\gamma > 1$ be constant, $c_{len} \coloneqq 256\gamma$ and assume $2 \leq \lambda = o(n)$. Consider an exploitation phase k with $\alpha_k n \geq n^{2/3}$. Let L_k be the progress made in phase k, i.e., the fitness difference between the fitter search point in the first generation of phase k and the fitter search point in the first generation of phase $k+1$. Then with probability $1 - o(n^{-\gamma})$,*

$$L_k \leq c_{len} \cdot \sqrt{\lambda} \log n =: L_{max}.$$

Proof. The phase ends as soon as the algorithm reaches a blocking bit as the next critical bit, as defined in Section 3. Recall that B_k is the number of blocking bits and S_k is the number of skipping bits in the non-optimized part of the bit-string. Notice that $\bar{H}_k = B_k + S_k$ and $\mathbb{E}[S_k \mid \bar{H}_k] = \mathbb{E}[B_k \mid \bar{H}_k] = \frac{1}{2}\bar{H}_k$, because each bit in $\bar{H}_k$ independently has a chance of $1/2$ to be either a skipping or a blocking bit. We will distinguish two cases.

Case 1: $\bar{H}_k \geq \frac{\alpha_k n}{2}$.
To show an upper bound on the duration of the exploitation phase we will need to show that it is very unlikely to have substantially more S_k than B_k bits. Notice that any bit among $\bar{H}_k$ comes from S_k or B_k independently with probability $1/2$, so $\mathbb{E}[B_k] \geq \alpha_k n/4$. Applying a Chernoff bound for $\alpha_k = \Omega(n^{-1/3})$ yields

$$\Pr\left[B_k < \tfrac{\alpha_k n}{8}\right] \leq \Pr\left[B_k \leq \tfrac{1}{2}\mathbb{E}[B_k]\right] \leq \exp\left(-\frac{\alpha_k n}{32}\right) = o(n^{-\gamma}).$$

In the following let $B_k^{\geq}$ be the event "$B_k \geq \frac{\alpha_k n}{8}$" and $L_k^{>}$ be the event "$L_k > L_{\max}$". Then we can bound $\Pr[L_k^{>}]$ by

$$\begin{aligned}\Pr[L_k^{>}] &= \Pr[L_k^{>} \mid B_k^{\geq}] \cdot \Pr[B_k^{\geq}] + \Pr[L_k^{>} \mid \neg B_k^{\geq}] \cdot \Pr[\neg B_k^{\geq}] \\ &\leq \Pr[L_k^{>} \mid B_k^{\geq}] + \Pr[\neg B_k^{\geq}] \leq \Pr[L_k^{>} \mid B_k^{\geq}] + o(n^{-\gamma}).\end{aligned}$$

To bound $\Pr[L_k^{>} \mid B_k^{\geq}]$, note that $L_k^{>}$ implies that there is no blocking bit among the next $L_{\max}$ bits after the critical bit of the beginning of phase k. We notice that for each bit we have a blocking-probability of $p_B = B_k/(\alpha_k n) \geq 1/8$. Moreover, blocking bits are negatively associated (for a fixed total number, having a blocking bit in position i decreases the chance of having a blocking bit at position $j \neq i$), hence the bound $\Pr[L_k^{>} \mid B_k^{\geq}] \leq (1-p_B)^{L_{\max}}$ applies [21, Property P2]. Not finding a blocking bit in $L_{\max}$ generations is therefore unlikely:

$$\Pr[L_k^{>} \mid B_k^{\geq}] \leq (1-p_B)^{L_{\max}} \leq e^{-p_B \cdot L_{\max}} \overset{\lambda \geq 1}{\leq} n^{-32\gamma} = o(n^{-\gamma}).$$

Case 2: $\bar{H}_k < \frac{\alpha_k n}{2}$.
For this case we know that there are at least $H_k = \alpha_k n - \bar{H}_k \geq \alpha_k n/2$ diverse bits, i.e. bits that differ between x^1 and x^2. We will study the development of B_k throughout the first $\tau = \lambda L_{\max}/16$ generations of phase k. We will show that after τ generations enough blocking bits will have been generated to stop the phase w.h.p. in further τ generations. Note that since this phase only uses crossover, blocking bits can only be generated, not destroyed.

Let us first show that w.h.p. the progress in the first τ generations is at most $L_{\max}/2$. In each generation, the offspring is accepted if and only if the improving bit is copied from the fitter search point x^2 to x^1, which happens with probability $1/\lambda$. Let us call σ the number of accepted offspring in the first τ generations of the phase. Then $\mathbb{E}[\sigma] = L_{\max}/16$, and by the Chernoff bound $\Pr[\sigma > L_{\max}/8] \leq \exp(-L_{\max}/48) = o(n^{-\gamma})$. Hence, we may assume $\sigma \leq L_{\max}/8$. Those are the only generations in which progress can be made, namely if the offspring z is strictly fitter than both x^1 and x^2. In this case, each subsequent bit of z after the critical bit has probability $1/2$ to be a one-bit, hence the progress in that generation is dominated by $1 + \mathcal{G}(1/2)$, where $\mathcal{G}$ denotes a geometric distribution. (Analogously to the argument for diversity phases in Section 3.) Hence, the expected progress with the first $\sigma \leq L_{\max}/8$ accepted offspring is at most $L_{\max}/4$, and the probability that it exceeds $L_{\max}/2$ is at most $\exp(-L_{\max}/16) = o(n^{-\gamma})$ because the sum of geometric random variables is concentrated [7, Theorem 1.10.32]. Hence, we may assume that the progress in the first τ generations is at most $L_{\max}/2$.

Let us pessimistically assume that the phase does not end in the first τ generations, since otherwise we are done. We write $B_{k,\tau}$ for the number of blocking bits after τ generations. Then as argued above, the expected number $\mathbb{E}[\sigma]$ of accepted offspring is $\tau/\lambda = L_{\max}/16$, and again by the Chernoff bound, we have $\sigma \geq L_{\max}/32$ with probability at least $1 - \exp(-L_{\max}/256) = 1 - o(n^{-\gamma})$. Hence, we may assume that at least $\sigma \geq L_{\max}/32$ offspring are accepted. Let us call this event $\mathcal{E}_\sigma$.

We will now study how many new blocking bits are generated under this assumption in the first τ generations. Let us call this number B_{new}. Consider a diverse pair of bits $(0,1)$ or $(1,0)$. By the condition $\bar{H}_k < \alpha_k n/2$, there are at least $\alpha_k n/2$ such pairs at the beginning of the phase. In a generation with accepted offspring the crossover operator copies each bit from x^2 to x^1 with probability $1/\lambda$, thus this is the probability of create equal bits $(0,0)$ or $(1,1)$. Ending up with equal values (either $(0,0)$ or $(1,1)$) in τ generations therefore happens with probability $1 - (1-1/\lambda)^\sigma$. Moreover, since the two options $(1,0)$ and $(0,1)$ for this position were equally likely to start with, the values $(0,0)$ and $(1,1)$ are also equally likely in the end. This yields an additional factor of $1/2$ that the position yields a stopping bit. Hence, the expected number of new blocking bits is

$$\mathbb{E}[B_{\text{new}} \mid \mathcal{E}_\sigma] \geq \frac{\alpha_k n}{2} \cdot \left(1 - (1-\tfrac{1}{\lambda})^{L_{\max}/32}\right) \cdot \frac{1}{2} \begin{cases} \overset{(5)}{\geq} \frac{\alpha_k n}{4} \cdot \frac{L_{\max}}{64\lambda} = \frac{\gamma \alpha_k n \log n}{\sqrt{\lambda}} =: \psi & \text{if } L_{\max} \leq 16\lambda, \\ \geq \frac{\alpha_k n}{4} \cdot (1 - e^{-1/2}) =: \psi & \text{if } L_{\max} > 16\lambda. \end{cases}$$

Moreover, conditioned on $\mathcal{E}_\sigma$, the random variable B_{new} dominates the sum of H_k independent indicator random variables, one for each position, where they are independent because their starting values are independent and crossover operates independently on them. Hence, B_{new} dominates a binomial random variable with expectation ψ. By the Chernoff bound, we have $\Pr[B_{\text{new}} < \psi/2] \leq \exp(-\Omega(\psi)) \leq \exp(-\Omega(n^{1/6})) = o(n^{-\gamma})$, since $\lambda = o(n)$ and $\alpha_k = \Omega(n^{-1/3})$. Therefore, from now on we may assume $B_{\text{new}} \geq \psi/2$. Let us call this event $\mathcal{B}^{\geq}_{\text{new}}$.

Let us call L^2_k the progress in phase k after the first τ generations. It remains to show that conditional on $\mathcal{B}^{\geq}_{\text{new}}$, we have $L^2_k \leq L_{\max}/2$ w.h.p. Note that the converse can only happen if none of the next $L_{\max}/2$ positions after the critical position is a blocking bit. We proceed similarly as in Case 1. Conditional on $\mathcal{B}^{\geq}_{\text{new}}$ each position has a probability of at least $\psi/(\alpha_k n)$ to be a blocking bit. As in Case 1, the blocking bits are negatively associated. Therefore, the probability that none of the next $L_{\max}/2$ bits are blocking bits is at most $(1 - \psi/(\alpha_k n))^{L_{\max}/2}$, and hence

$$\Pr\left[L^2_k > L_{\max}/2 \mid \mathcal{B}^{\geq}_{\text{new}}\right] \leq \left(1 - \tfrac{\psi}{\alpha_k n}\right)^{L_{\max}/2}. \tag{7}$$

If $L_{\max} \leq 16\lambda$ then we have $\psi/(\alpha_k n) = \gamma \log n/\sqrt{\lambda}$, and together with $L_{\max} = 256\gamma\sqrt{\lambda}\log n$ the right hand side of (7) is at most $\exp(-128\gamma^2 \log^2 n) = o(n^{-\gamma})$. If $L_{\max} > 16\lambda$ then $\psi/(\alpha_k n) = (1 - e^{-1/2})/4 \geq 1/12$, and the statement follows from $(1-1/12)^{L_{\max}/2} \leq \exp(-10\gamma\sqrt{\lambda}\log n) = o(n^{-\gamma})$. This concludes the proof. □

Next, we apply Lemma 4.5 to derive an upper bound on the expected increase from $\bar{H}_k$ to $\bar{H}_{k+1}$.

Lemma 4.6. *Let $\gamma > 1$, $c_{len} = 256\gamma$ and $2 \leq \lambda = o(n)$. Let phase k be an exploitation phase with $\alpha_k n = \Omega(n^{2/3})$. Then*

$$\mathbb{E}\left[\bar{H}_{k+1} - \bar{H}_k\right] \leq 5c_{len} \cdot \alpha_k n \cdot \log n/\sqrt{\lambda}.$$

Proof. As in the proof of Lemma 4.5 let $L^{\leq}_k, L^{>}_k$ be the events of staying within or exceeding $c_{\text{len}} \cdot \sqrt{\lambda}\log n$ generations in phase k respectively. Also, let $\Delta\bar{H}_k := \bar{H}_{k+1} - \bar{H}_k$. First we notice that a maximum of $\alpha_k n$ bits can contribute to $\Delta\bar{H}_k$. Therefore,

$$\begin{aligned} \mathbb{E}[\Delta\bar{H}_k] &= \mathbb{E}[\Delta\bar{H}_k \mid L^{\leq}_k] \cdot \Pr[L^{\leq}_k] + \mathbb{E}[\Delta\bar{H}_k \mid L^{>}_k] \cdot \Pr[L^{>}_k] \\ &\leq \mathbb{E}[\Delta\bar{H}_k \mid L^{\leq}_k] + \alpha_k n \cdot o(n^{-\gamma}) \leq \mathbb{E}[\Delta\bar{H}_k \mid L^{\leq}_k] + o(n^{1-\gamma}). \end{aligned} \tag{8}$$

We can choose any $\gamma > 1$, so the latter summand gives us a contribution of only $o(1)$. From now on we will study $\mathbb{E}[\Delta\bar{H}_k \mid L^{\leq}_k]$.

Conditioned on $L^{\leq}_k$, we want to bound the number of improving offspring. Note that every improving offspring increases the fitness of the worse search point by at least one. By slight abuse of notation, let us call x^1_k and x^2_k the two search points at the beginning of the phase. Then x^1_k can improve at most to $\text{LO}(x^2_k) + L_k$, since this is the maximal fitness obtained in this phase. Hence, the number of improving offspring is bounded by $L_k + \text{LO}(x^2_k) - \text{LO}(x^1_k)$. Note that the difference $\text{LO}(x^2_k) - \text{LO}(x^1_k)$ was created by free riders (bits that were 1 by chance) in the generation in which x^2 was created as new offspring. Thus it is distributed as $1 + \mathcal{G}(1/2)$ and is bounded by $2\log_2 n$ with probability at least $1 - 1/n^2$. By the same calculation as in (8), the error event is so unlikely that it is negligible, and we may thus assume that the number of improving offspring is at most $L_{\max} + 2\log_2 n \leq 2L_{\max}$ when n is large enough.

We now use a similar calculation as in the proof of Lemma 4.5, but now we compute an upper bound (instead of a lower bound) for the probability of turning two diverse bits into identical bits. Note that here we do not care whether the result is a blocking bit or skipping bit. We call p_{id} this probability of turning a fixed diverse pair into an identical one in the k-th phase. By using Equation (5) and $\lambda \geq 2$ we find

$$p_{\text{id}} \leq 1 - \left(1 - \tfrac{1}{\lambda}\right)^{2L_{\max}} \leq 1 - \exp(-4L_{\max}/\lambda) \leq 4L_{\max}/\lambda.$$

Therefore, recalling the definition of $L_{\max}$ from Lemma 4.5,

$$\mathbb{E}\left[\Delta\bar{H}_k \mid L_k^{\geq}\right] \leq \alpha_k n \cdot p_{\text{id}} \leq \alpha_k n \cdot 4c_{\text{len}}\sqrt{\lambda}\log n/\lambda.$$

We conclude $\mathbb{E}\left[\Delta\bar{H}_k\right] \leq 4c_{\text{len}} \cdot \alpha_k n \log n/\sqrt{\lambda} + o(1)$ in total. □

Now that we know how the diversity develops in the two respective phases we can finally prove Theorem 4.2.

Proof of Theorem 4.2. Fix arbitrarily $\gamma := 2$ (any other $\gamma > 1$ would also work). Choose $c > 0$ large enough such that $(1-\frac{1}{4e^2})(1+5c_{\text{len}}/c) < 0.98$, where c_{len} is the constant from Lemma 4.5.

We first prove by induction over k that

$$\mathbb{E}\left[\bar{H}_k\right] \leq c \cdot \alpha_k n \log n/\sqrt{\lambda} \tag{9}$$

holds at the beginning of every exploitation phase, where we do the inductive step from k to $k+2$, i.e., we consider the combined effect of an exploitation and a diversity phase. Afterwards, we show that it also holds in between and with a index-shifted α if we increase the factor c, i.e., we show for some constant $c' > 0$

$$\mathbb{E}\left[\bar{H}_t\right] \leq c' \cdot \alpha_{t-1} n \log n/\sqrt{\lambda} \tag{10}$$

for all times t for which $\alpha_{t-1} \geq n^{-1/3}$. Note that this is the statement of Theorem 4.2.

For the start of the induction, note from Algorithm 1 that the algorithm always starts with two antipodal search points, i.e. at maximum diversity. Moreover, this implies that initially $\text{LO}(x^1) \neq \text{LO}(x^2)$, so the first phase is an exploitation phase and $\bar{H}_k = 0$ for $k = 0$. This establishes the base case of the induction.

For the inductive step, assume that (9) holds for some even $k \geq 0$. Recall $L_{\max}$ from Lemma 4.5 and let $\mathcal{L} = L_k^{\leq} \cap L_{k+1}^{\leq}$ be the event that $L_k \leq L_{\max}$ and $L_{k+1} \leq L_{\max}$. Then $\Pr[L_k^{\leq}] = 1 - O(n^{-2})$ by Lemma 4.5, and $\Pr[L_{k+1}^{\leq}] = 1 - O(n^{-2})$ since the progress in a diversity phase is distributed as $1 + \mathcal{G}(1/2)$ and is bounded by $2\log_2 n$ with probability at least $1-1/n^2$. Hence, $\Pr[\mathcal{L}] = 1-O(n^{-2})$. Since $\bar{H}_{k+2} \in [0, n]$,

$$\begin{aligned}\mathbb{E}[\bar{H}_{k+2}] &= \mathbb{E}[\bar{H}_{k+2} \mid \mathcal{L}] \cdot \Pr[\mathcal{L}] + \mathbb{E}[\bar{H}_{k+2} \mid \neg\mathcal{L}] \cdot \Pr[\neg\mathcal{L}] \qquad (11)\\ &= \mathbb{E}[\bar{H}_{k+2} \mid \mathcal{L}](1 - O(n^{-2})) + o(1) = \mathbb{E}[\bar{H}_{k+2} \mid \mathcal{L}] \pm o(1).\end{aligned}$$

In the following we will thus compute $\mathbb{E}[\bar{H}_{k+2} \mid \mathcal{L}]$. Note that by the same calculation as (8) and (11), Lemmas 4.4 and 4.6 also hold when conditioning on $\mathcal{L}$, at the cost of an additive $\pm o(1)$ error term. Hence Lemma 4.6 yields

$$\begin{aligned}\mathbb{E}[\bar{H}_{k+1} \mid \mathcal{L}] &\leq \mathbb{E}[H_k \mid \mathcal{L}] + 5c_{\text{len}} \cdot \alpha_k n \log n/\sqrt{\lambda} + o(1)\\ &\overset{(9)}{\leq} (c + 5c_{\text{len}})\alpha_k n \log n/\sqrt{\lambda} + o(1), \qquad (12)\end{aligned}$$

and by Lemma 4.4,

$$\begin{aligned}\mathbb{E}[\bar{H}_{k+2}] &\leq \mathbb{E}[\bar{H}_{k+2} \mid \mathcal{L}] + o(1) \leq (1 - \tfrac{1}{4e^2})\mathbb{E}[\bar{H}_{k+1} \mid \mathcal{L}] + o(1)\\ &\leq (1 - \tfrac{1}{4e^2})(c + 5c_{\text{len}})\alpha_k n \log n/\sqrt{\lambda} + o(1)\\ &\leq 0.99c \cdot \alpha_k n \log n/\sqrt{\lambda}, \qquad (13)\end{aligned}$$

where we absorbed the $o(1)$ term into the constant in the last term. This is almost the desired inductive bound, except that we need α_{k+2} instead of α_k. However, conditional on $\mathcal{L}$ and for large enough n,

$$\alpha_{k+2} n = \alpha_k n - L_k - L_{k+1} \geq \alpha_k n - 2L_{\max} \geq 0.99\alpha_k n,$$

because $L_{\max} = O(\sqrt{n}\log n)$ and $\alpha n = \Omega(n^{2/3})$. This implies that $0.99\alpha_k \leq \alpha_{k+2}$. Hence we may continue (13) as

$$\mathbb{E}[\bar{H}_{k+2}] \leq \mathbb{E}[\bar{H}_{k+2} \mid \mathcal{L}] + o(1) \leq c \cdot \alpha_{k+2} n \log n/\sqrt{\lambda},$$

which yields (9) for $k+2$. By induction, this shows the statement for all even k. For the remaining points in time, which are not the start of an exploitation phase, we use very similar arguments. Equation (12) tells us that after the k-th phase, k even, we have $\mathbb{E}[\bar{H}_{k+1}] = O(\alpha_k n \log n/\sqrt{n})$. The same argument also shows that $\mathbb{E}[\bar{H}_t] = O(\alpha_k n \log n/\sqrt{n})$ for all times t during the k-th phase. For the $(k+1)$-th phase, since it is a diversity phase $\bar{H}$ can only decrease during this phase, so we also have $\mathbb{E}[\bar{H}_t] \leq \mathbb{E}[\bar{H}_{k+1}] = O(\alpha_k n \log n/\sqrt{n})$ for all times t during the $(k+1)$-th phase. As before α only changes by a $(1 - o(1))$ factor during one phase, hence we also obtain $\mathbb{E}[\bar{H}_t] = O(\alpha_t n \log n/\sqrt{n})$ for all t in either phase k or phase $k+1$, at the expense of a slightly larger hidden constant c'. Therefore, the statement (10) for all times t follows. □

5 EXPERIMENTS

This section is devoted to an empirical evaluation of DEGA to (i) validate our theoretical findings on LeadingOnes, (ii) discuss variants of the DEGA, and (iii) assess performance on classical benchmarks. We investigate the performance on OneMax (OM), LeadingOnes (LO), *Linear Functions* with harmonic weights *(LFHW)* and the *Maximum Independent Vertex Set (MIVS)*. Also, runtimes are compared to the established $(2+1)$-GA, $(1+(\lambda,\lambda))$-GA and UMDA [27]. Details on the parameter settings and benchmarks follow below.

Setup. For LO, OM and *LFHW*, we study the number of evaluations before finding the optimum ("runtime"). For *MIVS* we have to benchmark slightly differently, as we mostly do not reach the global optimum for any algorithm. The setup for *MIVS* is described in the full version [24]. All other experiments are based on 50 independent runs per problem size. We generate 10 log-spaced problem sizes in a range $[n_{\text{start}} = 100, n_{\text{end}}]$. The value of n_{end} may vary for different benchmarks (since simulation on LO is slower than on OM). We denote by $\bar{T}(n)$ the mean runtime (number of fitness evaluations) and by $\tilde{T}(n)$ the median runtime; error bars or intervals indicate one standard deviation where applicable. We normalize the average runtimes $\bar{T}(n)$ by n^2 (LO), $n\log n$ (OM) and $n\log n$ (*LFHW*). For reproducibility, our implementation and data are available on GitHub at https://github.com/FOGA2025-DEGA/DEGA.

LeadingOnes Asymptotics We begin by testing the algorithm as described in Section 2. In Figure 2, the average optimization time, divided by n^2, is shown for a range of λ's. We plot for a given λ_i, $\bar{T}(n)$ as well as the runtime guarantee from Theorem 4.1 for

that specific λ_i. Interestingly, the fit is remarkably tight for the proof-optimal $\lambda_1 = (n \log n)^{2/3}$, despite our bound not being tight for a given λ. This can be explained by the structure of the runtime, which is a sum of two terms.

Recall $\mathbb{E}[T] = O(\lambda n + n^2 \log n/\sqrt{\lambda})$. The λn-term is asymptotically tight, so even if the $n^2 \log n/\sqrt{\lambda}$-term is overestimated, choosing $\lambda = (n \log n)^{2/3}$ to theoretically balance the two terms will yield asymptotic time $\Theta(n^{5/3}(\log n)^{2/3})$. Therefore, we would expect that for $\lambda \geq (n \log n)^{2/3}$ our asymptotic prediction is correct, but for $\lambda = o((n \log n)^{2/3})$ the proven bound may overestimate the actual runtime. This is indeed what we find, as for any $\lambda_i, i > 1$ we have $\lambda_i = o(\lambda_1)$ and the runtime guarantees are overestimating the empirical runtimes in the figure. We also observe that several values of λ that are smaller than the proof-optimal value, namely $\lambda_2, \lambda_4, \lambda_6$ exhibit similar performance on the log-log plot. Additionally, we ran the algorithm with $\lambda = 2$. It is clear that $\lambda = 2$ causes the algorithm to closely resemble a vanilla (2+1)-GA. Unsurprisingly, the runtime is therefore not sub-quadratic for $\lambda = 2$. For all λ_i (from Figure 2), we log-transformed $x_j = \log n_j$ and $y_j = \log T(n_j)$ and performed linear regression on $(X, Y) = (x_j, y_j)_j$ to approximate the polynomial degree of the runtime for a given λ_i. We skipped the first four n values to reduce the influence of the standard deviation. The best fitting a for $Y = aX + b$ are displayed in Table 1. Note that this does not contradict our claim of a $O(n^{5/3} \log n) = O(n^{1.66} \log n)$ runtime. The log-term still influences the slope calculated by the regression. Considering a larger range of n's will result in the log-transformed data not being linear.

	λ_1	λ_2	λ_3	λ_4	λ_5	λ_6
a	1.749	1.695	2.001	1.766	1.859	1.706

Table 1: Empirical polynomial degree of the runtime for different λ.

We now turn our attention to other function classes and examine how DEGA performs against other algorithms. To this end, we also discuss variations of the standard $(2+1)$-DEGA, which we call A from now on. A version A', described in Figure 3, includes the ideas discussed in Section 1 to make the algorithm more robust. Note that in case of mutation, the offspring can only replace its parent but never the other point, even if it is fitter. A' uses an adaptive $\lambda = H(x, y)$ as bias, but still only exchanges one bit in expectation, and we suspect that an option of exchanging more bits would be beneficial on benchmarks like MIVS and Jump.

For another variant A_{BB}, we use an idea proposed in the context of the black-box complexity of LeadingOnes [12] in order to speed up the exploitation phases. The original A requires $\mathbb{E}[T_{\text{improve}}] = \Theta(\lambda)$ LO-evaluations per improving bit. [12] instead uses uniform crossover between x^1 and x^2, where x^1 is the strictly weaker point, until a fitter candidate y is found. The same procedure is then applied between x^1 and y to further update y. This is iterated $O(\log n)$ times, until x^1 and y are Hamming neighbours. Here, we obtain A_{BB} from A' by replacing the lower right box in Figure 3 by a loop of $10 \log n$ uniform crossovers between x^1 and y, where the offspring replaces y if it is fitter than x^1, and y replaces x^1 in the end.

Benchmark Comparison In the following, unless specified otherwise, we will work with $\lambda = n^{2/3}$ for the DEGA. For the $(2+1)$-GA crossover is performed with $p_c = 1/2$ to obtain y. With $1 - p_c$ we let y be a copy of a random parent. Then, y is mutated to generate offspring y'. For the $(1 + (\lambda, \lambda))$-GA we work with $k = \lambda = \sqrt{\log n}$, which yields an expected runtime of $O(n\sqrt{\log n})$ on OneMax [9]. The true asymptotic optimum involves a slightly larger λ [10] that gives only negligible additional speedup for the range of n that we consider. For the UMDA [27] we work with $\lambda = \sqrt{n} \log n, \mu = \log n$. Unsurprisingly, the DEGA (algorithm A) outperforms the other algorithms on LeadingOnes (Figure 5). The black-box version A_{BB} is by far the most efficient on LeadingOnes but is not generally the best choice on other benchmarks (Figures 4 and 5). For OneMax, defined as $\text{OneMax}(x) = \sum_{i=1}^{n} x_i$, the algorithm performs better for smaller λ, see Figure 5. The same figure shows a similar result for *linear functions with harmonic weights* (LFHW), where the fitness function is defined as $f : x \mapsto \sum_i i \cdot x_i$.

From the experiments we observe that a good choice for λ depends heavily on the function to be optimized. But we also notice that, at least for the benchmarks studied, there usually exists a λ that provides a boost in performance. We also observe that the A' adaptation is a good compromise between performance and applicability (see Figure 5), gaining an asymptotic speed-up on LO without losing much on OM and LFHW. A' is outperformed by $A, \lambda = \log n$ on OM and LFHW, so studying other self-adapting strategies for λ would be of great interest. Notice that smaller λ's make A more closely resemble a standard $(2+1)$-GA. One may think that we therefore just get the $(2+1)$-GA runtime for a small λ, like $\log n$. But this is not the case. The $\log n$ variant of the DEGA is the fastest at optimizing OM, beating the $(2+1)$-GA. This may hint at some constant factor improvement. We leave the theoretical analysis of the DEGA on OM to future work. As a last benchmark we consider the maximum independent vertex set (MIVS) problem.

MIVS. The benchmarks considered up to this point exhibit relatively smooth fitness landscapes in which crossover can exploit small improvements. We have not considered yet a benchmark where local maxima are difficult to escape. To test DEGA beyond such friendly settings, we chose the *Maximum Independent Vertex Set* (MIVS) problem. The instance we use is the graph F22 from the PBO suite of IOHprofiler [33]. The global optimum is hard to find, so we benchmark *time-to-target*, where the target is chosen as a fitness achievable by a $(1+1)$-EA. Crossover provides no obvious shortcut here; hence MIVS is deliberately *not* tailored to DEGA and serves as a stress test for its robustness. For A we limit the overall number of function evaluations in a single exploitation phase to $\lambda \log n$. This is necessary to allow at least some mutation from time to time. Further details of the experiments and discussion of the benchmark are in the full version [24].

The right panel of Figure 5 summarises the outcomes. We observe that the DEGA (Algorithm A) with $\lambda = \log n$ remains competitive with the $(2+1)$-GA and UMDA, but the larger $\lambda = n^{2/3}$ and the variant A' perform worse. Likely they waste too many function evaluations when there are two independent sets of different fitnesses.

Conclusion on experiments. Our experiments show that for small λ the DEGA behaves (in the worst case) very much like the well-known (2+1)-GA. But there are problems like OneMax, where the DEGA with small λ slightly outperforms the $(2+1)$-GA. There

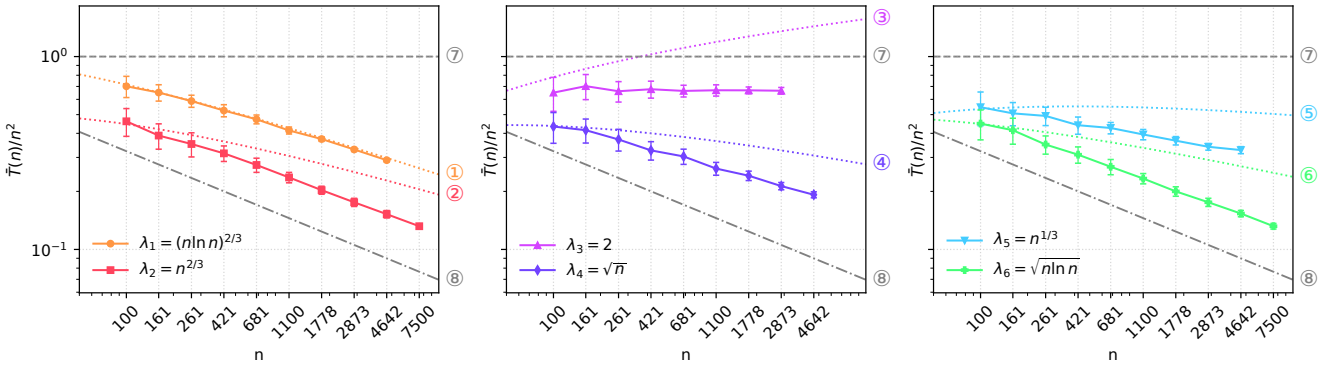

Figure 2: Log–log plot of mean runtime of the $(2+1)$-DEGA for different λ's on LeadingOnes. Dashed lines ①–⑥ show the runtime guarantees from Theorem 4.1 for the respective λ_i's. ⑦ and ⑧ mark the functions n^2 and $n^{5/3}$ for comparison. Bounds are scaled by suitable constants for visibility.

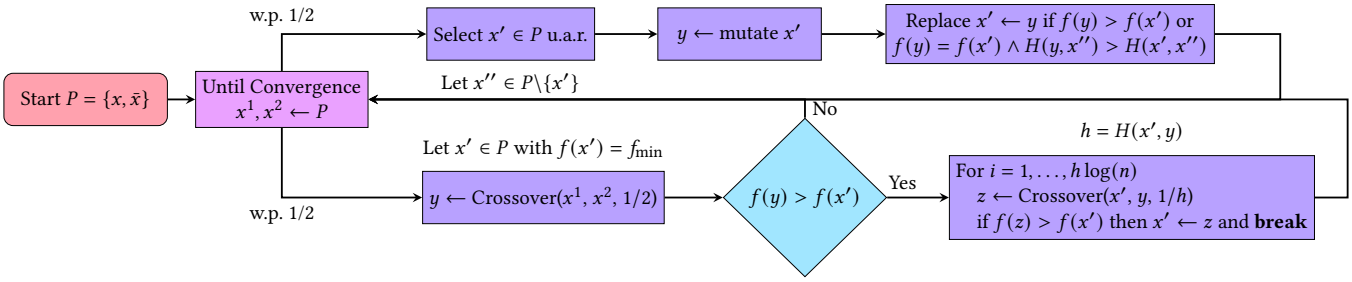

Figure 3: Flowchart of the A' variant of the $(2+1)$-DEGA.

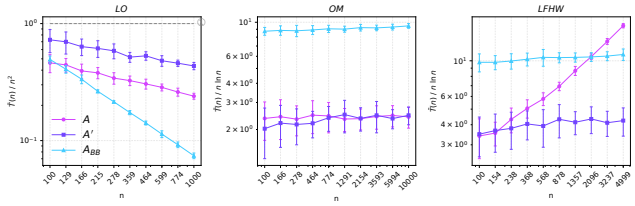

Figure 4: Log-log plot of normalized $\bar{T}(n)$ for DEGA-variants on LeadingOnes (left), OneMax (middle) and *linear functions* with harmonic weights (right). For A we again used $\lambda = n^{2/3}$.

is no DEGA variation that dominates across the benchmark suite. We do not claim that DEGA is universally best; instead, we show that it is competitive and robust. Hence it may be a good addition to algorithm portfolios. On LeadingOnes it is clearly superior. This invites a closer look at the structural properties that make LeadingOnes and OneMax particularly suitable for DEGA.

6 CONCLUSION AND FUTURE WORK

We have introduced a new paradigm of diversity-preserving exploitation of crossover (DiPEC), and given an algorithm based on this paradigm, the Diversity Exploitation Genetic Algorithm DEGA. We believe that we have provided enough evidence to justify exploring the algorithm and the paradigm further in future work. Obvious next steps are to test them on a wider set of benchmarks, and to explore further the different variations of the DEGA. Some important open questions include:

- Do the amendments of the DEGA work in practice? (E.g., adaptive choice of λ, continuous fitness functions.)
- How does the DEGA perform for larger population sizes?
- Which features of a fitness landscape make the DEGA faster than other algorithms, and which make it slower?

A particularly interesting question is how to integrate uniform crossover into the DEGA. There may be fitness landscapes where uniform crossover offspring are preferable over biased ones, with Jump and possibly MIVS as examples. The DEGA is designed to preserve diversity. Hence, the algorithm avoids on purpose accepting uniform crossover offspring. This is because every such offspring comes at a large cost for the diversity, and simply accepting them with a normal rate would defy the idea behind the DEGA. For example, we believe that this would decrease performance on LeadingOnes strongly. However, not accepting uniform crossover offspring at all seems an extreme choice. It is open how to optimally balance the objectives of sometimes accepting uniform crossover offspring, and of keeping the damage for diversity in check. A similar trade-off is yet to be found for fitness plateaus. The DEGA is conservative and accepts crossover offspring only if they give a strict fitness improvement. But this limits its ability to perform random walks on plateaus. Here, a better trade-off is desirable.

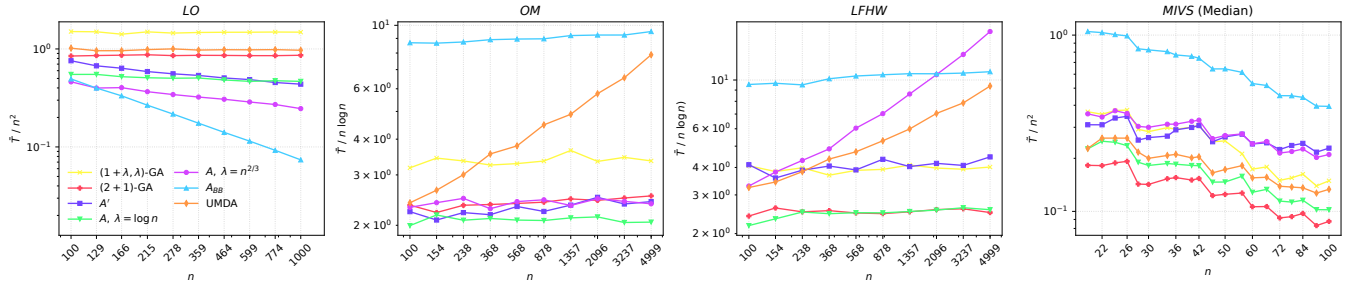

Figure 5: Log–log plot of normalized mean runtime for LeadingOnes (left), OneMax (middle-left) and linear functions with harmonic weights (middle-right). We use median runtime for MIVS (log-log, right). Mean for MIVS can be found in the full version [24]. We omit the standard deviation here to keep the graphic readable; it is reported for DEGA in Figures 2 and 4. Recall that we limit the number of exploration phases of the DEGA on MIVS, but not for LO, OM or LFHW.

REFERENCES

[1] Peyman Afshani, Manindra Agrawal, Benjamin Doerr, Carola Doerr, Kasper Green Larsen, and Kurt Mehlhorn. 2019. The query complexity of a permutation-based variant of Mastermind. *Discrete Applied Mathematics* 260 (2019), 28–50.

[2] Sacha Cerf and Johannes Lengler. 2024. How Population Diversity Influences the Efficiency of Crossover. In *Parallel Problem Solving from Nature (PPSN 2024)*. Springer, Springer, Cham, 102–116.

[3] D Corus and PS Oliveto. 2020. On the benefits of populations for the exploitation speed of standard steady-state genetic algorithms. *Algorithmica* 82, 12 (2020), 3676–3706.

[4] Matej Črepinšek, Shih-Hsi Liu, and Marjan Mernik. 2013. Exploration and exploitation in evolutionary algorithms: A survey. *ACM computing surveys (CSUR)* 45, 3 (2013), 1–33.

[5] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S Krejca, Per Kristian Lehre, Pietro S Oliveto, Dirk Sudholt, and Andrew M Sutton. 2016. Emergence of diversity and its benefits for crossover in genetic algorithms. In *Parallel Problem Solving from Nature (PPSN 2016)*. Springer, Cham, 890–900.

[6] Duc-Cuong Dang, Tobias Friedrich, Timo Kötzing, Martin S Krejca, Per Kristian Lehre, Pietro S Oliveto, Dirk Sudholt, and Andrew M Sutton. 2017. Escaping local optima using crossover with emergent diversity. *IEEE Transactions on Evolutionary Computation* 22, 3 (2017), 484–497.

[7] Benjamin Doerr. 2020. Probabilistic tools for the analysis of randomized optimization heuristics. In *Theory of evolutionary computation: Recent developments in discrete optimization*. Springer, Cham, 1–87.

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

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

[10] Benjamin Doerr, Carola Doerr, and Johannes Lengler. 2021. Self-Adjusting Mutation Rates with Provably Optimal Success Rules. *Algorithmica* 83, 10 (2021), 3108–3147.

[11] Benjamin Doerr, Aymen Echarghaoui, Mohammed Jamal, and Martin S Krejca. 2024. Runtime Analysis of the $(\mu+1)$ GA: Provable Speed-Ups from Strong Drift towards Diverse Populations. In *AAAI Conference on Artificial Intelligence (AAAI 2024)*, Vol. 38. ACM, New York, 20683–20691.

[12] Benjamin Doerr, Daniel Johannsen, Timo Kötzing, Per Kristian Lehre, Markus Wagner, and Carola Winzen. 2011. Faster black-box algorithms through higher arity operators. In *Foundations of Genetic Algorithms (FOGA 2011)*. ACM, New York, 163–172.

[13] Benjamin Doerr and Martin Krejca. 2020. Significance-based estimation-of-distribution algorithms. *IEEE Transactions on Evolutionary Computation* 24, 6 (2020), 1483–1490.

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

[15] Benjamin Doerr and Carola Winzen. 2012. Black-box complexity: Breaking the O (n log n) barrier of LeadingOnes. In *Artificial Evolution (EA 2011)*. Springer, Cham, 205–216.

[16] Carola Doerr and Johannes Lengler. 2017. Introducing elitist black-box models: When does elitist behavior weaken the performance of evolutionary algorithms? *Evolutionary Computation* 25, 4 (2017), 587–606.

[17] Carola Doerr and Johannes Lengler. 2018. The (1+ 1) Elitist Black-Box Complexity of LeadingOnes. *Algorithmica* 80, 5 (2018), 1579–1603.

[18] Stefan Droste, Thomas Jansen, and Ingo Wegener. 2002. On the analysis of the (1+1) Evolutionary Algorithm. *Theoretical Computer Science* 276, 1-2 (2002), 51–81.

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

[20] Thomas Jansen and Ingo Wegener. 2005. Real royal road functions—where crossover provably is essential. *Discrete Applied Mathematics* 149, 1-3 (2005), 111–125.

[21] Kumar Joag-Dev and Frank Proschan. 1983. Negative association of random variables with applications. *The Annals of Statistics* 11, 1 (1983), 286–295.

[22] Timo Kötzing, Dirk Sudholt, and Madeleine Theile. 2011. How crossover helps in pseudo-Boolean optimization. In *Genetic and Evolutionary Computation Conference (GECCO 2011)*. ACM, New York, 989–996.

[23] Per Kristian Lehre and Carsten Witt. 2012. Black-Box Search by Unbiased Variation. *Algorithmica* 64, 4 (2012), 623–642.

[24] Johannes Lengler and Tom Offermann. 2025. Diversity-Preserving Exploitation of Crossover. arXiv:2507.01524 https://arxiv.org/abs/2507.01524

[25] Johannes Lengler, Andre Opris, and Dirk Sudholt. 2024. A Tight $O(4^k/p_c)$ Runtime Bound for a $(\mu+1)$ GA on Jump_k for Realistic Crossover Probabilities. In *Genetic and Evolutionary Computation Conference (GECCO 2024)*. ACM, New York, 1605–1613.

[26] Johannes Lengler, Andre Opris, and Dirk Sudholt. 2024. Analysing Equilibrium States for Population Diversity. *Algorithmica* 86, 9 (2024), 2317–2351.

[27] Heinz Mühlenbein. 1997. The equation for response to selection and its use for prediction. *Evolutionary Computation* 5, 3 (1997), 303–346.

[28] Günter Rudolph. 1997. *Convergence properties of evolutionary algorithms*. Verlag Dr. Kovač, Hamburg.

[29] Dirk Sudholt. 2012. Crossover speeds up building-block assembly. In *Genetic and Evolutionary Computation Conference (GECCO 2012)*. ACM, New York, 689–702.

[30] Dirk Sudholt. 2017. How crossover speeds up building block assembly in genetic algorithms. *Evolutionary computation* 25, 2 (2017), 237–274.

[31] Dirk Sudholt. 2020. The benefits of population diversity in evolutionary algorithms: a survey of rigorous runtime analyses. In *Theory of evolutionary computation: Recent developments in discrete optimization*. Springer, Cham, 359–404.

[32] Andrew M Sutton. 2021. Fixed-parameter tractability of crossover: steady-state GAs on the closest string problem. *Algorithmica* 83 (2021), 1138–1163.

[33] Hao Wang, Diederick Vermetten, Furong Ye, Carola Doerr, and Thomas Bäck. 2022. IOHanalyzer: Detailed performance analyses for iterative optimization heuristics. *ACM Transactions on Evolutionary Learning and Optimization* 2, 1 (2022), 1–29.

[34] Richard A Watson. 2001. Analysis of recombinative algorithms on a non-separable building-block problem. In *Foundations of Genetic Algorithms (FOGA 2001)*. Springer, Cham, 69–89.

[35] Richard A Watson and Thomas Jansen. 2007. A building-block royal road where crossover is provably essential. In *Genetic and Evolutionary Computation Conference (GECCO 2007)*. ACM, New York, 1452–1459.

[36] Darrell Whitley. 2019. Next generation genetic algorithms: a user's guide and tutorial. In *Handbook of Metaheuristics*. Springer, Cham, 245–274.

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