SN Computer Science (2022) 3:324
https://doi.org/10.1007/s42979-022-01203-z

ORIGINAL RESEARCH

Runtime Analysis of the $(\mu + 1)$-EA on the Dynamic BinVal Function

Johannes Lengler[1] · Simone Riedi[1]

Received: 14 July 2021 / Accepted: 13 May 2022 / Published online: 10 June 2022

Abstract

We study evolutionary algorithms in a dynamic setting, where for each generation a different fitness function is chosen, and selection is performed with respect to the current fitness function. Specifically, we consider Dynamic BinVal, in which the fitness functions for each generation is given by the linear function BinVal, but in each generation the order of bits is randomly permuted. For the $(1 + 1)$-EA it was known that there is an efficiency threshold c_0 for the mutation parameter, at which the runtime switches from quasilinear to exponential. Previous empirical evidence suggested that for larger population size μ, the threshold may increase. We prove that this is at least the case in an ε-neighborhood around the optimum: the threshold of the $(\mu + 1)$-EA becomes arbitrarily large if the μ is chosen large enough. However, the most surprising result is obtained by a second-order analysis for $\mu = 2$: the threshold *in*creases with increasing proximity to the optimum. In particular, the hardest region for optimization is *not* around the optimum.

Keywords Evolutionary algorithm · Populations · Dynamic linear functions · Dynamic BinVal · DynBV · Mutation rate

Introduction

Evolutionary algorithms are optimization heuristics that are based on the idea of maintaining a population of solutions that evolves over time. This incremental nature is an important advantage of population-based optimization heuristics over non-incremental approaches. At any point in time the population represents a set of solutions. This makes population-based optimization heuristics very flexible. For example, the heuristic can be stopped after any time budget (predefined or chosen during execution), or when some desired quality of the solutions is reached. For the same reason, population-based algorithms are naturally suited for dynamic environments, in which the optimization goal ("fitness function") may change over time. In such a setting, it is not necessary to restart the algorithm from scratch when the fitness function changes, but rather we can use the current population as starting point for the new optimization environment. If the fitness function changes slowly enough, then population-based optimization heuristics may still find the optimum, or track the optimum over time [2, 7, 11, 12, 22–24, 26–28]. We refrain from giving a detailed overview over the literature, since an excellent review has recently been given in [25]. All the settings have in common that either the fitness function changes with very low frequency, or it changes only by some small local differences, or both.

Recently, a new setting, called *dynamic linear functions* was proposed by Lengler and Schaller [19]. They argued that it might either be called noisy linear functions or dynamic linear functions, but we prefer the term dynamic. A class of dynamic linear functions is determined by a distribution $\mathcal{D}$ on the positive reals $\mathbb{R}^+$. For the τth generation, n weights $W_1^\tau, \dots, W_n^\tau$ are chosen independently identically distributed (i.i.d.) from $\mathcal{D}$, and the fitness function for this generation is given by $f^\tau : \{0,1\}^n \to \mathbb{R}^+; f^\tau(x) = \sum_{i=1}^n W_i^\tau x_i$. Therefore, the fitness in each generation is given by a linear function with positive weights, but the weights are drawn randomly in each generation. Note that for any fitness function, a one-bit in the ith position will always yield a better fitness than a

This article is part of the topical collection "Bio-inspired Algorithms for Combinatorial Optimization" guest edited by Aniko Ekart, Christine Zarges and Sébastien Verel.

An extended abstract of the paper has appeared in the proceedings of the EvoCOP conference [18]. The extended abstract was missing many proofs, in particular the whole derivation of the second-order expansion in Sect. 5.

✉ Johannes Lengler
johannes.lengler@inf.ethz.ch

[1] Department of Computer Science, ETH Zürich, Universitätsstrasse 6, 8092 Zürich, Switzerland

zero-bit. In particular, all fitness functions share a common global maximum, which is the string OPT = (1...1). Hence, the fitness function may change rapidly and strongly from generation to generation, but the direction of the signal remains unchanged: one-bits are preferred over zero-bits.

Crucially, by dynamic environments we mean that *selection* is performed according to the current fitness function as in [8]. That is, all individuals from parent and offspring population are compared with respect to *the same* fitness function. Other versions exist, e.g., [5] studies the same problem as [8] without re-evaluations, i.e., there algorithms would compare fitnesses such as $f^{\tau}(x)$ and $f^{\tau+1}(y)$ with each other, which will never happen in our setting.

Several applications of dynamic linear functions are discussed in [19]. One of them is a chess engine that can switch databases for different openings ON or OFF. The databases strictly improve performance in all situations, but if the engine is trained against varying opponents, then an opening may be used more or less frequently; so the weight of the corresponding bit may be high or low. Obviously, it is desirable that an optimization heuristic manages to switch all databases ON in such a situation. However, as we will see, this is not automatically achieved by many simple optimization heuristics. Rather, it depends on the parameter settings whether the optimal configuration (all databases ON) is found.

In [19], the runtime (measured as the number of iterations until the optimum is found) of the well-known $(1+1)$-EA on dynamic linear functions was studied. The $(1+1)$-EA, or "$(1+1)$ Evolutionary Algorithm", is a simple hill-climbing algorithm for maximizing a pseudo-Boolean function $f : \{0,1\}^n \rightarrow \mathbb{R}$. It only maintains a population size of $\mu = 1$, so it maintains a single solution $x^{\tau} \in \{0,1\}^n$. In each round (also called *generation*), a randomized *mutation operator* is applied to x^{τ} to generate an *offspring* y^{τ}. Then the *fitter* of the two is maintained, so we set $x^{\tau+1} := x^{\tau}$ if $f(x^{\tau}) > f(y^{\tau})$, and $x^{\tau+1} := y^{\tau}$ if $f(x^{\tau}) < f(y^{\tau})$. In case of equality, we break ties randomly. The mutation operator of the $(1+1)$-EA is *standard bit mutation*, which flips each bit of x^{τ} independently with probability c/n, where c is called the *mutation parameter*. The authors of [19] gave a full characterization of the optimization behavior of the $(1+1)$-EA on dynamic linear functions in terms of the mutation parameter c. It was shown that there is a threshold $c^* = c^*(\mathcal{D}) \in \mathbb{R}^+ \cup \{\infty\}$ such that for $c < c^*$ the $(1+1)$-EA optimizes the dynamic linear function with weight distribution $\mathcal{D}$ in time $O(n \log n)$. On the other hand, for $c > c^*$, the algorithm needs exponential time to find the optimum. The threshold $c^*(\mathcal{D})$ was given by an explicit formula. For example, if $\mathcal{D}$ is an exponential distribution then $c^*(\mathcal{D}) = 2$, if it is a geometric distribution $\mathcal{D} = \text{GEOM}(p)$ then $c^* = (2-p)/(1-p)$. Moreover, the authors in [19] showed that there is $c_0 \approx 1.59..$ such that $c^*(\mathcal{D}) \geq c_0$ for every distribution $\mathcal{D}$, but for any $\varepsilon > 0$ there is a distribution $\mathcal{D}$ with $c^*(\mathcal{D}) < c_0 + \varepsilon$. As a consequence, if $c < c_0$ then the $(1+1)$-EA with mutation parameter c/n needs time $O(n \log n)$ to optimize any dynamic linear function, while for $c > c_0$ there are dynamic linear functions on which it needs exponential time.

While it was satisfying to have such a complete picture for the $(1+1)$-EA, a severe limitation was that the $(1+1)$-EA is very simplistic. In particular, it was unclear whether a non-trivial population size $\mu > 1$ would give a similar picture. This question was considered in the experimental paper [16, 17] by Lengler and Meier. Instead of working with the whole class of dynamic linear functions, they defined the *dynamic binary value function* DynBV as a limiting case. In DynBV, in each generation a uniformly random permutation $\pi^{\tau} : \{1,\dots,n\} \rightarrow \{1,\dots,n\}$ of the bits is drawn, and the fitness function is then given by $f^{\tau}(x) = \sum_{i=1}^{n} 2^{n-i} x_{\pi^{\tau}(i)}$. Therefore, in each generation, DynBV evaluates the so-called BinVal function with respect to a permutation of the search space. Lengler and Meier observed that the proof in [19] for the $(1+1)$-EA extends to DynBV with threshold $c^* = c_0$, i.e., the $(1+1)$-EA needs time $O(n \log n)$ for mutation parameter $c < c_0$, and exponential time for $c > c_0$. In this sense, DynBV is the hardest dynamic linear function, although it is not formally a member of the class of dynamic linear functions.

The papers [16, 17] performed experiments on DynBV for two population-based algorithms, the $(\mu+1)$-EA (using only mutation) and the $(\mu+1)$-GA (using randomly mutation or crossover; GA stands for "Genetic Algorithm"). In $(\mu+1)$ algorithms, a population of size μ is maintained, see also Algorithm 1. In each generation, a single offspring is generated, and the least fit of the $\mu+1$ search points is discarded, breaking ties randomly. Thus they generalize the $(1+1)$-EA. In the $(\mu+1)$-EA, the offspring is generated by picking a random *parent* from the population and performing standard bit mutation as in the $(1+1)$-EA. In the $(\mu+1)$-GA, it is also possible to generate the offspring by *crossover*: two random parents x_1, x_2 are selected from the population, and each bit is taken randomly either from x_1 or x_2. In each generation of the $(\mu+1)$-GA, it is decided randomly with probability 1/2 whether the offspring is produced by a mutation or by a crossover.[1]

[1] Other conventions are possible, e.g., that both crossover and mutation are applied subsequently in the same generation. Here we describe the version in [16] and [17].

Algorithm 1: A generic $(\mu + \lambda)$ algorithm in dynamic environments. In this paper, we use the $(\mu + 1)$-EA. That means that we use $\lambda = 1$, standard bit mutation with mutation parameter c (mutation rate c/n), and elitist selection (greedy selection by fitness).

```
Initialize $P^0$ with $\mu$ strings chosen u.a.r. from $\{0,1\}^n$;
for $\tau = 0, 1, 2, 3...$ do
    From $P^\tau$, generate $\lambda$ offspring $\hat{x}_1, .., \hat{x}_\lambda$. ;
    $S \leftarrow P^\tau \cup \{\hat{x}_1, .., \hat{x}_\lambda\}$;
    Based on the current fitnesses $f^\tau(x)$ for $x \in S$, select $\mu$ individuals from $S$ to
      obtain $P^{\tau+1}$
```

Lengler and Meier ran experiments for $\mu \in \{1, 2, 3, 5\}$ on DynBV and found two main results. As they increased the population size μ from 1 to 5, the efficiency threshold c_0 increased moderately for the $(\mu + 1)$-EA (from 1.6 to 3.4, and strongly for the $(\mu + 1)$-GA (from 1.6 to more than 20). Therefore, with larger population size, the algorithms have a larger range of feasible parameter settings, and even more so when crossover is used.

Moreover, they studied which range of the search space was hardest for the algorithms, by estimating the drift towards the optimum with Monte Carlo simulations. For the $(\mu + 1)$-GA, they found that the hardest region was around the optimum, as one would expect. Surprisingly, for the $(\mu + 1)$-EA with $\mu \geq 2$, this did not seem to be the case. They gave empirical evidence that the hardest regime was bounded away from the optimum. That is, there were parameters c for which the $(\mu + 1)$-EA had positive drift (towards the optimum) in a region around the optimum. However, it had *negative* drift in an intermediate region that was further away from the optimum. This finding is remarkable, since it contradicts the commonplace that optimization gets harder closer to the optimum.[2] Notably, a very similar phenomenon was proven by Lengler and Zou [21] for the $(\mu + 1)$ -EA on certain monotone functions ("HotTopic "), see the discussion below. Strikingly, such an effect was neither built into the fitness environments (not for HotTopic, and not for DynBV) nor into the algorithms. Rather, it seems to originate in a complex (and detrimental!) population dynamics that unfolds only in a regime of weak selective pressure, i.e., in a regime, where offspring are often accepted even if they are less fit than the parent. If selective pressure is strong, then the population often degenerates into copies of the same search point. As a consequence, diversity is lost, and the $(\mu + 1)$-EA degenerates into the $(1 + 1)$-EA. In these regimes, diversity *decreases* the ability of the algorithms to make progress. For HotTopic functions, these dynamics are well-understood [13, 21]. For dynamic linear functions, even though we can prove this behavior in this paper for the $(2 + 1)$-EA (see below), we are still far from a real understanding of these dynamics. Most likely, they are different from the dynamics for HotTopic functions.

Our Results

We complement the experiments in [16, 17] with rigorous mathematical analysis. To this end, we study the *degenerate population drift* (see Sect. 2) for the $(\mu + 1)$-EA with mutation parameter $c > 0$ on DynBV in an ε-neighbourhood of the optimum. That is, we assume that the search points in the current population have at least $(1 - \varepsilon)n$ one-bits, for some sufficiently small constant $\varepsilon > 0$. We find that for every constant $c > 0$ there is a constant μ_0 such that for $\mu \geq \mu_0$ the drift is positive (multiplicative drift towards the optimum). This means that with high probability the $(\mu + 1)$-EA will need time $O(n \log n)$ to improve from $(1 - \varepsilon)n$ one-bits to the optimum, if μ is large enough. This implies that larger population sizes are helpful, since the drift of the $(1 + 1)$-EA around the optimum is negative for all $c > c_0 \approx 1.59..$ (which implies exponential optimization time). So for any $c > c_0$, increasing the population size to a large constant decreases the runtime from exponential to quasi-linear, provided that the algorithm starts in an ε-neighbourhood of the optimum. This is consistent with the experimental findings in [16] for $\mu = \{1, 2, 3, 5\}$, and it proves that population size can compensate for arbitrarily large mutation parameters.

For the $(2 + 1)$-EA, we perform a second-order analysis (i.e., we determine not just the main order term of the drift, but also the second-order term) and prove that in an ε -neighborhood of the optimum, the drift decreases with the distance from the optimum. In particular, there are some values of c for which the drift is positive around the optimum, but negative in an intermediate distance. It follows from standard arguments that there are $\varepsilon, c > 0$ such that the runtime is $O(n \log n)$ if the algorithm is started in an ε-neighborhood of the optimum, but that it takes exponential time to reach this ε-neighborhood. Thus we formally prove that the hardest part of optimization is not around the optimum,

[2] https://en.wikipedia.org/wiki/Pareto_principle.

as was already experimentally concluded from Monte Carlo simulations in [16].

Related Work

Jansen [9] introduced a pessimistic model for analyzing the $(1+1)$-EA on linear functions, later extended in [1], which is *also* a pessimistic model for dynamic linear functions and DynBV *and* for monotone functions. A monotone function $f:\{0,1\}^n \rightarrow \mathbb{R}$ is a function, where for every $x \in \{0,1\}^n$, the fitness of x strictly increases if we flip any zero-bit of x into a one-bit. Thus, as for dynamic linear functions and DynBV, a one-bit is always better than a zero-bit, the optimum is always at $(1,\dots,1)$, and there are short fitness-increasing paths from any search point to the optimum. Thus it is reasonable to call all these setting "easy" from an optimization point of view, which makes it all the more surprising that such a large number of standard optimization heuristics fail so badly. Keep in mind that despite the superficial similarities between monotone functions and DynBV or dynamic linear functions, the basic setting is rather different. Monotone functions were studied in static settings, i.e., we have only a single static function to optimize, and a search point never changes its fitness. Nevertheless, the performance of some algorithms is surprisingly similar on monotone functions and on dynamic linear functions or DynBV. In particular, the mutation parameter c plays a critical role in both settings. It was shown in [6] that the $(1+1)$-EA needs exponential time to optimize some monotone functions if the mutation parameter c is too large, while it is efficient on all monotone functions if $c<1$.[3] The construction of hard monotone instances was simplied in [20] and later called HotTopic functions. HotTopic functions were analyzed for a large set of algorithms in [13]. For the $(1+\lambda)$-EA, the $(1+(\lambda,\lambda))$-GA, the $(\mu+1)$-EA, and the $(1+\lambda)$-fEA, thresholds for the mutation parameter c or related quantities were determined such that a larger mutation rate leads to exponential runtime, and a smaller mutation rate leads to runtime $O(n\log n)$. (For details on these algorithms, see [13].) Interestingly, the population size μ and offspring population size λ of the algorithms had no impact on the threshold. Crucially, all these results were obtained for parameters of HotTopic functions in which only the behavior in an ε-neighborhood around the optimum mattered. This dichotomy between quasilinear and exponential runtime is very similar to the situation for DynBV. However, for the $(\mu+1)$-EA on HotTopic functions the threshold c_0 was independent of μ, while we show that on DynBV it becomes arbitrarily large as μ grows. Thus large population sizes help for DynBV, but not for HotTopic.

As we prove, for the $(2+1)$-EA the region around the optimum is not the hardest region for optimization, and there are values of c for which there is a positive drift around the optimum, but a negative drift in an intermediate region. As Lengler and Zou showed [21], the same phenomenon occurs for the $(\mu+1)$-EA on HotTopic functions. In fact, they showed that larger population size even hurts: for any $c>0$ there is a μ_0 such that the $(\mu+1)$-EA with $\mu \geq \mu_0$ has negative drift in some intermediate region (and thus exponential runtime), even if c is much smaller than one! This surprising effect is due to population dynamics in which it is not the genes of the fittest individuals who survive in the long terms. Rather, individuals which are strictly dominated by others (and substantially less fit) serve as the seeds for new generations. Importantly, the analysis of this dynamics relies on the fact that for HotTopic functions, the weight of the positions stay fixed for a rather long period of time (as long as the algorithm stays in the same region/level of the search space). Thus, the results do not transfer to DynBV functions. Nevertheless, the picture looks similar insofar as the hardest region for optimization is not around the optimum in both cases. Since our analysis for DynBV is only for $\mu=2$, we can't say whether the efficiency threshold in c is increasing or decreasing with μ. The experiments in [16, 17] find increasing thresholds (so the opposite effect as for HotTopic), but are only for $\mu \leq 5$.

Preliminaries

Dynamic Optimization and the Dynamic Binary Value Function DynBV

The general setting of a $(\mu+\lambda)$ algorithm in dynamic environments on the hypercube $\{0,1\}^n$ is as follows. A population P^τ of μ search points is maintained. In each generation τ, λ offspring are generated. Then a *selection operator* selects the next population $P^{\tau+1}$ from the $\mu+\lambda$ search points according to the fitness function f^τ. A pseudocode description can be found in Algorithm 1.

In this paper, we will study the $(\mu+1)$-Evolutionary Algorithm ($(\mu+1)$-EA) with *standard bit mutation* and *elitist selection*. Therefore, for offspring generation, a parent x is chosen uniformly at random from P^τ, and the offspring is generated by flipping each bit of x independently with probability c/n, where c is the *mutation parameter*. For selection, we simply select the μ individuals with largest f^τ-values to form population $P^{\tau+1}$.

For the dynamic binary value function DynBV, for each $\tau \geq 0$ a uniformly random permutation $\pi^\tau:\{1,\dots,n\} \rightarrow \{1,\dots,n\}$ is drawn, and the fitness function for generation τ is then given by $f^\tau(x)=\sum_{i=1}^n 2^{n-i}x_{\pi^\tau(i)}$.

[3] This was later extended to $c<1+\varepsilon$ in [15].

Notation and Setup

Throughout the paper, we will assume that the population size μ and the mutation parameter c are constants, whereas n tends to ∞. In particular, we will hide factors c and μ in Landau notation $O(\cdot)$. On the other hand, we will frequently describe the distance from the optimum in the form εn, where $\varepsilon = \varepsilon(n)$ may depend on n. In particular, we never hide the dependency on ε in Landau notation. We *will* choose ε_0 to be constant for the main theorem in the end, but for the analysis it is crucial to understand the asymptotics of ε. We will give more details on the role of ε below.

We use the expression "with high probability" or whp for events $\mathcal{E}_n$ such that $\Pr(\mathcal{E}_n) \to 1$ for $n \to \infty$. We write $x = O(y)$, where x and y may depend on n, if there is $C > 0$ such that $|x| \leq Cy$ for sufficiently large n. Note that we take the absolute value of x. The statement $x = O(y)$ does not imply that x must be positive. Consequently, if we write an expression, such as $\Delta = \varepsilon + O(\varepsilon^2)$, then we mean that there is a constant $C > 0$ such that $\varepsilon - C\varepsilon^2 \leq \Delta \leq \varepsilon + C\varepsilon^2$ for sufficiently large n. It does not imply anything about the sign of the error term $O(\varepsilon^2)$. We will sometimes use minus signs or "±" to ease the flow of reading, e.g., we write $1 - o(1)$ for probabilities. However, this is a cosmetic decision, and is equivalent to $1 + o(1)$.

For two bit-strings $x, y \in \{0,1\}^n$, we say that x *dominates* y if $x_i \geq y_i$ for all $i \in \{1, \dots, n\}$.

Our main tool will be drift theory. To apply this, we need to identify states that we can adequately describe by a single real value. Following the approach in [13] and [16], we call a population *degenerate* if it consists of μ copies of the same individual. If the algorithm is in a degenerate population, we will study how the *next degenerate population* looks like, so we define

$$\Phi^t := \{\text{\# of zero-bits in an individual} \\ \text{n the tth degenerate population}\}. \quad (1)$$

We will use the convention that τ is the index of the τth generation, and t is the index of the tth *degenerate* population. Therefore, the time between Φ^t and Φ^{t+1} can span several generations. Our main object of study will be the *degenerate population drift* (or simply *drift* if the context is clear). For $0 \leq \varepsilon \leq 1$, it is defined as

$$\Delta(\varepsilon) := \Delta^t(\varepsilon) := \mathbb{E}\left[\Phi^t - \Phi^{t+1} \mid \Phi^t = \lfloor \varepsilon n \rfloor\right]. \quad (2)$$

The expression is independent of t, since the considered algorithms are time-homogeneous. If we want to stress that $\Delta(\varepsilon)$ depends on the parameters μ and c, we also write $\Delta(\mu, c, \varepsilon)$. Note that the number of generations to reach the $(t+1)$st degenerate population is itself a random variable. Therefore, the number of generations to go from Φ^t to Φ^{t+1} is random. As in [13], its expectation is $O(1)$ if μ and c are constants, and it has an exponentially decaying tail bound, see Lemma 1 below. In particular, the probability that during the transition from one degenerate population to another the same bit is touched by two different mutations is $O(\varepsilon^2)$, and likewise the contribution of this case to the drift is $O(\varepsilon^2)$, as we will prove formally in Lemma 2.

As mentioned above, we do not assume constant ε, i.e., $\varepsilon = \varepsilon(n)$ may depend on n. For our main result we *will* choose ε to be a sufficiently small constant, but we need to choose it such that we can determine the sign of terms, such as $\varepsilon \pm O(\varepsilon^2) \pm o(1)$. Note that this is possible: if $\varepsilon > 0$ is a sufficiently small constant then $\varepsilon \pm O(\varepsilon^2) \geq \varepsilon/2$, and if afterwards we choose n to be sufficiently large then the $o(1)$ term is at most $\varepsilon/4$. Since this is subtle point, we will not treat ε as a constant. In particular, all O-notation is with respect to $n \to \infty$, and does not hide dependencies on ε. This is why we have to keep error terms, such as $O(\varepsilon^2)$ and $o(1)$ separate.

To compute the degenerate population drift, we will frequently need to compute the expected change of the potential provided that we visit an intermediate state S. Here, a state S is simply given by a population of μ search points. We will call this change the *drift from state* S, and denote it by $\Delta(S, \varepsilon)$. Formally, if $\mathcal{E}(S, t)$ is the event that the algorithm visits state S between the tth and $(t+1)$st degenerate population,

$$\Delta(S, \varepsilon) := \mathbb{E}\left[\Phi^t - \Phi^{t+1} \mid \Phi^t = \varepsilon n \text{ and } \mathcal{E}(S, t)\right]. \quad (3)$$

This term is closely related to the *contribution to the degenerate population drift from state* S, which also contains the probability to reach S as a factor:

$$\Delta_{\text{con}}(S, \varepsilon) := \Pr[\mathcal{E}(S, t) \mid \Phi^t = \varepsilon n] \cdot \Delta(S, \varepsilon). \quad (4)$$

We will study DynBV around the optimum, i.e., we consider any $\varepsilon = \varepsilon(n) \to 0$, and we compute the asymptotic expansion of $\Delta(\varepsilon)$ for $n \to \infty$. As we will see, the drift is of the form $\Delta(\varepsilon) = a\varepsilon + O(\varepsilon^2) + o(1)$ for some constant $a \in \mathbb{R}$, where the $o(1)$ term is independent of ε. In the end, this will allow us to prove existence of a *constant* ε such that the sign of $\Delta(\varepsilon)$ equals the sign of a for sufficiently large n. Analogously to [13] and [21], standard drift theorems imply that if a is *positive* (multiplicative drift), then the algorithm starting with at most $\varepsilon_0 n$ zero-bits for some suitable constant ε_0 whp needs $O(n \log n)$ generations to find the optimum. On the other hand, if a is *negative* (negative drift/updrift), then whp the algorithm needs exponentially many generations to find the optimum (regardless of whether it is initialized randomly or with $\varepsilon_0 n$ zero-bits). These two cases are typical. There is no term independent of ε in the drift, since for a degenerate population P^τ we have $P^{\tau+1} = P^\tau$ with probability $1 - O(\varepsilon)$.

This happens whenever mutation does not touch any zero-bit, since then the offspring is rejected.[4]

We will prove that, as long as we are only interested in the first- order expansion (i.e., in a results of the form $a\varepsilon + O(\varepsilon^2) + o(1)$), we may assume that between two degenerate populations, the mutation operators always flip different bits. In this case, we use the following naming convention for search points. The individuals of the tth degenerate population are all called x^0. We call other individuals $x^{(m_1-m_2)}$, where m_1 stands for the extra number of ones and m_2 for the extra number of zeros compared to x^0. Hence, if x^0 has m zero-bits then $x^{(m_1-m_2)}$ has $m + m_2 - m_1$ zero-bits. Following the same convention, we will denote by X_k^z a set of k copies of x^z, where the index z may be 0 or $(m_1 - m_2)$. In particular, X_μ^0 denotes the tth degenerate population.

Duration Between Degenerate Populations

We formalize the assertions in Sect. 2.2 that the number of steps between two degenerate populations satisfies exponential tail bounds, and that it is unlikely to touch a bit by two different mutations as we transition from one degenerate population to the next. We give proofs for completeness, but similar statements are well-known in the literature.

Lemma 1 *For all constant μ, c there is a constant $a > 0$ such that the following holds for the $(\mu + 1)$-EA with mutation parameter c in any population X on* DynBV. *Let K be the number of generations until the algorithm reaches the next degenerate population. Then for all $k \in \mathbb{N}_0$:*

$$\Pr(K \geq k \cdot \mu) \leq e^{-a \cdot k}.$$

Proof Let $x^0 \in X$ be the individual with the least number of zero-bits, and let $\hat{p}$ be the probability to degenerate in the next μ steps. Clearly, $\hat{p}$ is at least the probability that in each step we copy x^0 and accept it into the population. The probability of selecting x^0 and mutating no bits is at least $\frac{1}{\mu}(1 - \frac{c}{n})^n \xrightarrow{n\to\infty} e^{-c}/\mu$, so for sufficiently large n this probability is at least $e^{-c}/(2\mu)$. Since x^0 is the individual with the least number of zeros in the population, the probability that it is not worst in the population and thus all copies are kept is at least 1/2: any other individual $y \neq x^0$ will have at least as many zeros as x and, therefore, will be ranked lower than x^0 with probability at least 1/2. Therefore, for sufficiently large n:

$$\hat{p} \geq \left(\frac{e^{-c}}{4\mu}\right)^\mu.$$

(In fact, one could replace μ by $\mu - 1$ and obtain a stronger bound, since $\mu - 1$ rounds of inserting x^0 already suffice, but this would only make the final formula slightly more complicated.) This bound works for any starting population. Therefore, if we don't degenerate in the first μ steps of the algorithm we again have probability $\hat{p}$ to degenerate in the successive μ steps, and so on. Therefore, we can simply bound the probability not to degenerate in the first $k \cdot \mu$ steps by $(1 - \hat{p})^k \leq e^{-\hat{p} \cdot k}$, where the last step uses the inequality $(1 + x) \leq e^x \forall x \in \mathbb{R}$ [3]. □

The next lemma formalizes the well-known phenomenon that close to the optimum, the course of the algorithm is dominated by events in which at most one zero-bits flips at a time. Here we go slightly further. Even when we consider the period in which the algorithm transitions from one degenerate population to another, then the event that two or more zero-bits are flipped in this period is negligible, and contributes only an $O(\varepsilon^2)$ term to the drift.

Lemma 2 *Consider the $(\mu + 1)$-EA with mutation parameter c on* DynBV. *Let X^t and X^{t+1} denote the tth and $(t + 1)$st degenerate population respectively. Let $\varepsilon > 0$, and let X be a degenerate population with at most εn zero-bits.*

(a) *Let $\mathcal{E}_2$ be the event that the mutations during the transition from X^t to X^{t+1} flip at least two zero-bits. Then $\Pr[\mathcal{E}_2 \mid X^t = X] = O(\varepsilon^2)$. Moreover, the contribution of this case to the degenerate population drift Δ is*

$$\Delta^*(\varepsilon) := \Pr[\mathcal{E}_2 \mid X^t = X] \cdot \mathbb{E}[\Delta^t(\varepsilon) \mid \mathcal{E}_2 \wedge X^t = X] = O(\varepsilon^2).$$

(b) *Let S be any non-degenerate state such that there is at most one position which is a one-bit in some individuals in S, but a zero-bit in X. Let $\mathcal{E}(S,t)$ be the event that state S is visited during the transition from X^t to X^{t+1}, and let $\mathcal{E}_1$ be the event that a zero-bit is flipped in the transition from S to X^{t+1}. Then $\Pr[\mathcal{E}_1 \mid \mathcal{E}(S,t) \wedge X^t = X] = O(\varepsilon)$, and the contribution to $\Delta(S,\varepsilon)$ is*

$$\begin{aligned}\Delta^*(S,\varepsilon) &:= \Pr[\mathcal{E}_1 \mid \mathcal{E}(S,t) \wedge X^t = X] \\ &\cdot \mathbb{E}[\Delta^t(\varepsilon) \mid \mathcal{E}_1 \wedge \mathcal{E}(S,t) \wedge X^t = X] \\ &= O(\varepsilon).\end{aligned} \qquad (5)$$

The contribution of the case $\mathcal{E}(S,t) \wedge \mathcal{E}_1$ to the degenerate population drift is

$$\Delta^*_{\text{con}}(S,\varepsilon) := \Pr[\mathcal{E}(S,t)] \cdot \Delta^*(S,\varepsilon) = O(\varepsilon^2).$$

[4] Here and later we use the convention that if an offspring is identical to the parent, and they have lowest fitness in the population, then the offspring is rejected. Since the outcome of ejecting offspring or parent is the same, this convention does not change the course of the algorithm.

In both parts, the hidden constants do not depend on S and X.
Proof (a) If the offspring in the first iteration is not accepted into the population or is identical to x^0, then the population is immediately degenerate again, and there is nothing to show. Therefore, let us consider the case that the offspring is different and is accepted into the population. Then the mutation needed to flip at least one zero-bit, since otherwise the offspring is dominated by x^0 and rejected. Thus the probability of this case is $O(\varepsilon)$. Moreover, the probability of flipping at least two zero-bits in this mutation is $O(\varepsilon^2)$, so we may assume that *exactly one* zero-bit is flipped.

Let k be such that the number of subsequent iterations until the next degenerate population is in $[\mu k, \mu(k+1)]$. Conditioning on k, we have at most $\mathcal{O}(k)$ mutations until the population degenerates, each with a probability of at most εc of flipping a zero-bit. Using Lemma 1 we have for some $a > 0$,

$$\Pr[\mathcal{E}_2 \mid X^t = X] \le \mathcal{O}(\varepsilon) \cdot \sum_{k=1}^{\infty} e^{-a\cdot k} \cdot \mathcal{O}(\varepsilon \cdot k) = \mathcal{O}(\varepsilon^2). \qquad (6)$$

Recall from (1) the notion Φ^t for the number of zero-bits in the tth degenerate population. To bound the contribution to the drift of the case, where an additional zero-bit flip happens, we will bound the expectation of $|\Phi^t - \Phi^{t+1}|$, conditioned on being in this case. That difference is at most the number of bit flips until the next degenerate population, which is $\mathcal{O}(k)$ in expectation. Again, summing over all possible k and using Lemma 1 we get

$$\begin{aligned}\Delta^*(\varepsilon) &\le \mathcal{O}(\varepsilon) \cdot \sum_{k=1}^{\infty} \mathcal{O}(\varepsilon \cdot k) \cdot e^{-a\cdot k} \cdot \mathcal{O}(k) \\ &= \mathcal{O}\left(\varepsilon^2 \sum_{k=1}^{\infty} k^2 e^{-a\cdot k}\right) = \mathcal{O}(\varepsilon^2).\end{aligned} \qquad (7)$$

(b) Analogously to (6) and (7), but without the factor $O(\varepsilon)$, since we already start in state S, we have

$$\Pr[\mathcal{E}_1 \mid \mathcal{E}(S,t) \wedge X^t = X] \le \sum_{k=1}^{\infty} e^{-a\cdot k} \cdot \mathcal{O}(\varepsilon \cdot k) = \mathcal{O}(\varepsilon)$$

and

$$\Delta^*(S, \varepsilon) \le \sum_{k=1}^{\infty} \mathcal{O}(\varepsilon \cdot k) \cdot e^{-a\cdot k} \cdot \mathcal{O}(k) = \mathcal{O}(\varepsilon).$$

The probability $\Pr[\mathcal{E}(S,t)]$ is bounded by the probability that the transition from X^t to X^{t+1} visits any state different from X^t at all, which is $O(\varepsilon)$. This proves the last statement. □

Before we start to analyze the algorithms, we prove a helpful lemma to classify how the population can degenerate if no zero-bit is flipped. As we have explained in Sect. 2.2 (and made formal in Lemma 1 and Lemma 2), this assumption holds with high probability. In this case, the population degenerates to copies of an individual which is not dominated by any other search point.

Lemma 3 *Consider the* $(\mu+1)$*-EA in a non-degenerate population* X*. Let* $x_1, x_2, ..., x_k$ *be search points in* X *that dominate all the rest of the population. Then either at least one zero-bit is flipped until the next degenerate population, or the next degenerate population consists of copies of one of the search points* $x_1, x_2, ..., x_k$*.*

Proof Assume that, starting from X, the algorithm does not flip any additional zero-bits. We start by inductively showing that for all subsequent time steps, every individual in the population is still dominated by one of the search points $x_1, x_2, ..., x_k$. Suppose, for the sake of contradiction, that eventually there are individuals which are not dominated by any of the search points in $\{x_1, x_2, ..., x_k\}$, and let x^* be the first such individual. Since we assumed that the algorithm doesn't flip any additional zero-bits, x^* must have been generated by mutating an individual $\bar{x}$ and only flipping one-bits. Therefore, $\bar{x}$ dominates x^*. On the other hand, $\bar{x}$ is dominated by one of the search points $x_1, x_2, ..., x_k$ by our choice of x^*. This is a contradiction, since domination is transitive. Therefore, using transitivity, the algorithm will not generate any individual that is not dominated by any search point in $\{x_1, x_2, ..., x_k\}$. Furthermore, the population will never degenerate to any other individual $\tilde{x} \notin \{x_1, x_2, ..., x_k\}$. In fact, let x_i be the search point in $\{x_1, x_2, ..., x_k\}$ that dominates $\tilde{x}$. We have that $f(\tilde{x}) < f(x_i)$ in all iterations and for all permutations; therefore, x_i will never be discarded before $\tilde{x}$, which concludes the proof. □

Analysis of the Degenerate Population Drift

In this section, we will find a lower bound for the drift $\Delta(\varepsilon) = \Delta(\mu, c, \varepsilon)$ of the $(\mu+1)$-EA close to the optimum, when $n \to \infty$. The main result of this section will be the following.

Theorem 1 *For all constants* $c > 0$ *there exist constants* $\delta, \varepsilon_0 > 0$ *such that for all* $\varepsilon \le \varepsilon_0$ *and* $\mu \ge \mu_0 := e^c + 2$*, if* n *is sufficiently large*:

$$\Delta(c, \mu, \varepsilon) \ge \delta \cdot \varepsilon.$$

Lemma 3 allows us to describe the transition from one degenerate population to the next by a relatively simple Markov chain, provided that at most one zero-bit is flipped during the transition. This zero-bit needs to be flipped to

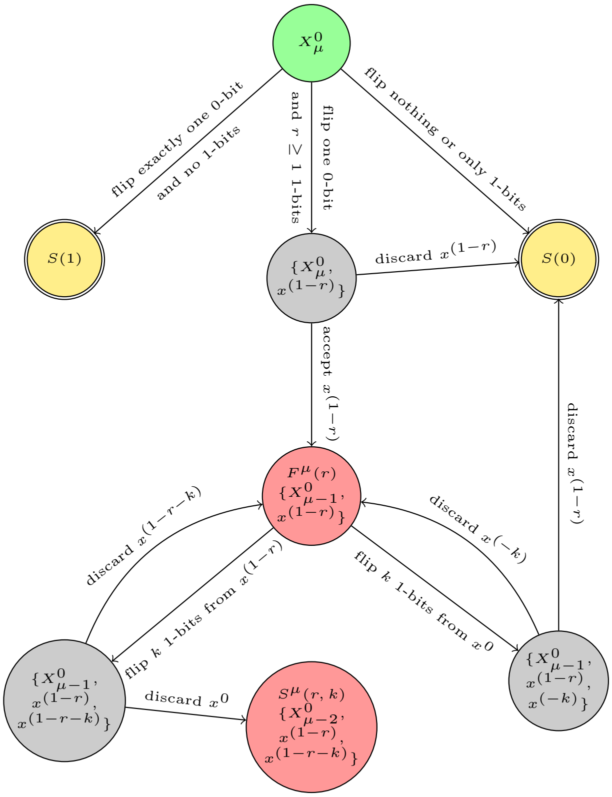
X_μ^0
flip exactly one 0-bit
and no 1-bits
and $r \geq 1$ 1-bits
flip one 0-bit
flip nothing or only 1-bits
$S(1)$
$\{X_\mu^0, x^{(1-r)}\}$
discard $x^{(1-r)}$
$S(0)$
accept $x^{(1-r)}$
discard $x^{(1-r)}$
$F^\mu(r)$ $\{X_{\mu-1}^0, x^{(1-r)}\}$
discard $x^{(1-r-k)}$
discard $x^{(-k)}$
flip k 1-bits from $x^{(1-r)}$
flip k 1-bits from x^0
$\{X_{\mu-1}^0, x^{(1-r)}, x^{(1-r-k)}\}$
discard x^0
$S^\mu(r,k)$ $\{X_{\mu-2}^0, x^{(1-r)}, x^{(1-r-k)}\}$
$\{X_{\mu-1}^0, x^{(1-r)}, x^{(-k)}\}$

Fig. 1 State diagram for the $(\mu + 1)$-EA

leave the starting state, so we assume for this chain that no zero-bit is flipped afterwards. This assumption is justified by Lemma 2. The Markov chain (or rather, a part of it) is shown in Fig. 1. The starting state, which is a degenerate population, is depicted in green. The yellow states $S(k)$ represent degenerate populations, where the number of one-bits is exactly k larger than that of the starting state, so $\Phi^{t+1} - \Phi^t = k$. In later diagrams, we will also see negative values of k. We have included intermediate states depicted in gray, in which an offspring has been created, but selection has not yet taken place. In other words, the gray states have $\mu + 1$ search points, and it still needs to be decided which of them should be discarded from the population. As we will see in the analysis, it is quite helpful to separate offspring creation from this selection step. The remaining states are depicted in red. We denote by $F^\mu(r)$ the state reached from X_μ^0 by flipping one 0-bit and $r \geq 1$ one-bits and accepting the offspring, and we denote by $S^\mu(r, k)$ the state reached from $F^\mu(r)$ by flipping k 1-bits from the new individual $x^{(1-r)}$ and accepting the offspring. Note that we have only drawn part of the Markov chain, since from the bottom-most state $S^\mu(r, k)$, we have not drawn outgoing arrows or states.

Moreover, the states of the Markov chain do not correspond one-to-one to the generations: we omit intermediate states, where Lemma 3 allows us to do that. For example, following the first arrow to the left we reach a state in which one individual $x^{(1)}$ (the offspring) dominates all other individuals. By Lemma 3, such a situation must degenerate into μ copies of $x^{(1)}$, so we immediately mark this state as a degenerate state with $\Phi^{t+1} - \Phi^t = 1$.

The key step will be to give a lower bound for the contribution to the drift from state $F^\mu(r)$. Once we have a bound on this, it is straightforward to compute a bound on the degenerate population drift. Before we turn to the computations, we first introduce a bit more auxiliary notation.

Definition 1 Consider the $(\mu + 1)$-EA in state $F^\mu(r)$ in generation $\tau - 1$. We re-sort the n positions of the search points descendingly according to the next fitness function f^τ. Therefore, by the "first" position we refer to the position which has highest weight according to f^τ, and the jth bit of a bitstring z is given by $(\pi^\tau)^{-1}(j)$. Then, we define:

- B_z := position of the first zero-bit in$z$;
- B_0^τ := position of the first flipped bit in the mutation of the τ − th generation;
- $z_1^\tau := \arg\min \{f^\tau(z) \mid z \in \{X_{\mu-1}^0, X_1^{(1-r)}\}\}$;
- $z_2^\tau := \arg\max \{f^\tau(z) \mid z \in \{X_{\mu-1}^0, X_1^{(1-r)}\}\}$.

In particular, the search point to be discarded in generation τ is either z_1^τ or the offspring generated by the τth mutation. We define B_0^τ to be ∞ if no bits are flipped in the τth mutation.

Now we are ready to bound the drift of state $F^\mu(r)$. We remark that the statement for $\mu = 2$ was also proven in [17], but the proof there was much longer and more involved, since it did not make use of the hidden symmetry of the selection process that we will use below.

Lemma 4 *Consider the $(\mu + 1)$-EA on the* DynBV *function in the state $F^\mu(r)$ for some $r \geq 1$, and let $\varepsilon > 0$. Then the drift from $F^\mu(r)$ is*

$$\Delta(F^\mu(r), \varepsilon) \geq \frac{1 - r}{1 + (\mu - 1) \cdot r} + \mathcal{O}(\varepsilon).$$

For $\mu = 2$, consider a state $S = \{x_1, x_2\}$ that is reached from some degenerate population with individual x. Assume that x_1 has $i_1 = O(1)$ additional one-bits and $i_2 = O(1)$ additional zero-bits compared to x_2. Moreover, assume that the total number of zero-bits in x is by $j_1 \in \mathbb{Z}$ larger than in x_1 and by $j_2 \in \mathbb{Z}$ larger than in x_2. Then

$$\Delta(S, \varepsilon) = \frac{i_1}{i_1 + i_2} \cdot j_1 + \frac{i_2}{i_1 + i_2} \cdot j_2 + \mathcal{O}(\varepsilon).$$

Proof We first consider the case of general μ. Let us assume that the algorithm will not flip an additional zero-bit through mutation before it reaches the next degenerate population. In fact, the contribution to the drift in case it does flip another zero-bit can be summarized by $\mathcal{O}(\varepsilon)$ due to Eq. (5) in Lemma 2. So from now on, we assume that the algorithm doesn't flip an additional zero-bit until it reaches the next degenerate population.

The idea is to follow the Markov chain as shown in Fig. 1. We will compute the conditional probabilities of reaching different states from $F^{\mu}(r)$, conditional on actually leaving $F^{\mu}(r)$. More precisely, we will condition on the event that an offspring $\bar{x}$ is generated and accepted into the population.

Recall that $F^{\mu}(r)$ corresponds to the population of $\{X^0_{\mu-1}, X^{(1-r)}_1\}$, i.e., $\mu - 1$ copies of x^0 and one copy of $x^{(1-r)}$. Therefore, if the offspring is accepted, one of these search points must be ejected from the population. Let us first consider the case that $x^{(1-r)}$ is ejected from the population. Then the population is dominated by x^0 afterwards, and will degenerate into X^0_{μ} again by Lemma 3. The other case is that one of the x^0 individuals is ejected, which is described by state $S^{\mu}(r,k)$. It is complicated to compute the contribution of this state precisely, but by Lemma 3 we know that this population will degenerate either to copies of x^0 or of $x^{(1-r)}$. For $\mu = 2$, only the second case is possible, since there are no copies of x^0 left in $S^{\mu}(r,k)$. Thus we either get $\Delta(S^{\mu}(r,k),\varepsilon) = 0$ or $\Delta(S^{\mu}(r,k),\varepsilon) = 1 - r$. Since $r \geq 1$, in both cases we can use the pessimistic bound $\Delta(S^{\mu}(r,k),\varepsilon) \geq 1 - r$ for the drift of $S^{\mu}(r,k)$, with equality for $\mu = 2$.[5] Summarizing, once a new offspring is accepted, if a copy of x^0 is discarded we get a drift of at most $1 - r$ and if $x^{(1-r)}$ is discarded we get a drift of 0. It only remains to compute the conditional probabilities with which these cases occur.

Computing the probabilities is not straightforward, but we can use a rather surprising symmetry, using the terminology from Definition 1. Assume that the algorithm is in generation τ. We make the following observation: an offspring is accepted if and only if it is mutated from z^{τ}_2 and $B^{\tau}_0 > B_{\min} := \min\{B_{x^0}, B_{x^{(1-r)}}\}$. Hence, we need to compute the probability:

$$\hat{p} := \Pr\left(f^{\tau}(x^{(1-r)}) \geq f^{\tau}(x^0) \mid \{\text{mutated } z^{\tau}_2\} \wedge \{B^{\tau}_0 > B_{\min}\}\right),$$

since then we can bound $\Delta(F^{\mu}(r),\varepsilon) \geq (1-r)\hat{p} + \mathcal{O}(\varepsilon)$ by Lemma 2. For $\mu = 2$, this lower bound is an equality.

Clearly, the events $\{f^{\tau}(x^{(1-r)}) \geq f^{\tau}(x^0)\}$ and $\{B^{\tau}_0 > B_{\min}\}$ are independent, since the position $B_{\min}$ is independent on whether the one-bit at this position belongs to $x^{(1-r)}$ or to x^0. We emphasize that this is a rather subtle symmetry of the selection process that would not be easily visible without describing the selection process in terms of $B_{\min}$. One way to intuitively phrase it (but perhaps less obvious) is that in the permutation π^{τ}, the *internal ordering* of the $r+1$ bits in which $x^{(1-r)}$ and x^0 differ is independent of the *set of absolute positions* that these $r+1$ bits receive by π^{τ}. The former information determines which of $x^{(1-r)}, x^0$ has larger fitness, while the latter determines whether the offspring is rejected. Using the independence and conditional probability, $\hat{p}$ simplifies to:

$$\hat{p} = \frac{\Pr\left(f(x^{(1-r)}) \geq f(x^0) \wedge \{\text{mutated} \quad z^{\tau}_2\}\right)}{\Pr\left(\{\text{mutated} \quad z^{\tau}_2\}\right)}. \tag{8}$$

To compute the remaining probabilities, we remind the reader that $x^{(1-r)}$ has exactly r more zero-bits and 1 more one-bit than x^0. Hence, to compare them, we only need to look at the relative positions of these $r+1$ bits in which they differ. In particular, $x^{(1-r)} = z^{\tau}_2$ holds if and only if the permutation π^{τ} places the one-bit from $x^{(1-r)}$ before the r one-bits of x^0, and this happens with probability $1/(r+1)$. Moreover, recall that there are $\mu - 1$ copies of x^0 and only one $x^{(1-r)}$, so the probability of picking them as parents is $(\mu-1)/\mu$ and $1/\mu$, respectively. Therefore, by using the law of total probability,

$$\begin{aligned}
&\Pr\left(\{\text{mutated} \quad z^{\tau}_2\}\right) \\
&= \Pr\left(\{\text{mutated} \quad z^{\tau}_2\} \mid x^{(1-r)} = z^{\tau}_2\right) \cdot \Pr\left(x^{(1-r)} = z^{\tau}_2\right) \\
&\quad + \Pr\left(\{\text{mutated} \quad z^{\tau}_2\} \mid x^0 = z^{\tau}_2\right) \cdot \Pr\left(x^0 = z^{\tau}_2\right) \\
&= \frac{1}{\mu} \cdot \frac{1}{r+1} + \frac{\mu-1}{\mu} \cdot \frac{r}{r+1}
\end{aligned}$$

Plugging this into (8) yields

$$\hat{p} = \left(\frac{1}{r+1} \cdot \frac{1}{\mu}\right) \Big/ \left(\frac{1}{\mu} \cdot \frac{1}{r+1} + \frac{\mu-1}{\mu} \cdot \frac{r}{r+1}\right) = \frac{1}{1 + (\mu-1)r}.$$

Together with Lemma 2 and the lower bound $\Delta(F^{\mu}(r),\varepsilon) \geq (1-r)\hat{p} + O(\varepsilon)$, this concludes the proof of the first part.

For $\mu = 2$, the argument is similar. Again, by Lemma 2, we may assume that no further zero-bit is flipped and none of the $i_1 + i_2$ bits in which x_1 and x_2 differ is flipped, since these cases only contribute a term $\mathcal{O}(\varepsilon)$. Then as soon as an offspring is accepted, its parent dominates the population. (Recall that we count it as rejection of the offspring if it is a copy of the parent and one of the copies gets removed.) Afterwards, the population will degenerate into copies of the surviving parent by Lemma 3. Hence the change in Φ^t will either be j_1 or j_2. To compute the probability of the first case, we let

$$\hat{p} := \Pr\left(f^{\tau}(x_1) > f^{\tau}(x_2) \mid \text{offspring accepted}\right).$$

[5] The notation is slightly imprecise here, since we condition on the event that no further zero-bit is flipped, which is not reflected in the notation. But as argued above, this only adds an additive $O(\varepsilon)$ error term to the final result.

As for the general case, we use the surprising symmetry that the event "$f^\tau(x_1) > f^\tau(x_2)$" is independent of the event that the offspring is accepted (under the assumption that none of the bits is flipped in which x_1 and x_2 differ). Thus we again get the analogous formula to (8), except that now both individuals have the same probability to be mutated, since both exist only with one copy in the population. Hence

$$\hat{p} = \Pr\left(f^\tau(x_1) > f^\tau(x_2)\right) = \frac{i_1}{i_1 + i_2},$$

and therefore

$$\begin{aligned}\Delta(S, \varepsilon) &= \hat{p} \cdot j_1 + (1 - \hat{p}) \cdot j_2 + \mathcal{O}(\varepsilon)\\ &= \frac{i_1}{i_1 + i_2} \cdot j_1 + \frac{i_2}{i_1 + i_2} \cdot j_2 + \mathcal{O}(\varepsilon).\end{aligned}$$

□

Now we are ready to bound the degenerate population drift and prove Theorem 1.

Proof of Theorem 1 To prove this theorem, we refer to Fig. 1. By Lemma 2, the contribution of all states that involve flipping more than one zero-bit is $O(\varepsilon^2)$. If we flip no zero-bits at all, then the population degenerates to X^0_μ again, which contributes zero to the drift. Therefore, we only need to consider the case, where we flip exactly one zero-bit in the transition from the tth to the $(t + 1)$st degenerate population. This zero-bit needs to be flipped in the first mutation, since otherwise the population does not change. We denote by p_r the probability to flip exactly one zero-bit and r one-bits in x^0, thus obtaining $x^{(1-r)}$. If $f^\tau(x^{(1-r)}) > f^\tau(x^0)$ then $x^{(1-r)}$ is accepted into the population and we reach state $F^\mu(r)$. This happens if and only if among the $r + 1$ bits in which $x^{(1-r)}$ and x^0 differ, the zero-bit of x^0 is the most relevant one. Therefore, $\Pr[f^\tau(x^{(1-r)}) > f^\tau(x^0)] = 1/(r + 1)$ Finally, by Lemma 4, the drift from $F^\mu(r)$ is at least $-(r - 1)/(1 + (\mu - 1)r) + \mathcal{O}(\varepsilon)$. Summarizing all this into a single formula, we obtain

$$\begin{aligned}\Delta(\varepsilon) &\geq \mathcal{O}(\varepsilon^2) + p_0 + \sum_{r=1}^{(1-\varepsilon)n} p_r \cdot \left[\Pr[f^\tau(x^{(1-r)}) > f^\tau(x^0)] \cdot \Delta(F^\mu(r), \varepsilon)\right]\\ &\geq \mathcal{O}(\varepsilon^2) + p_0 - \sum_{r=1}^{(1-\varepsilon)n} p_r \cdot \frac{1}{r+1} \cdot \left(\frac{r-1}{1 + (\mu - 1)r} + \mathcal{O}(\varepsilon)\right).\end{aligned} \tag{9}$$

For p_r, we use the following standard estimate, which holds for all $r = o(\sqrt{n})$.

$$\begin{aligned}p_r &= \frac{(1-\varepsilon)n \cdot ((1-\varepsilon)n - 1) \cdot \ldots \cdot ((1-\varepsilon)n - r + 1)}{r!} \binom{\varepsilon n}{1} \left(\frac{c}{n}\right)^{r+1} \left(1 - \frac{c}{n}\right)^{n-r}\\ &= (1 + o(1)) \cdot c^{r+1}/r! \cdot e^{-c} \cdot \varepsilon \cdot (1 - \varepsilon)^r,\end{aligned}$$

where $(1 - c/n)^{n-r} = (1 + o(1))e^{-c}$ by [3] and $(1 - \varepsilon)n - i = (1 - \varepsilon)n \cdot (1 - \frac{i}{(1-\varepsilon)n})$, with total error factor $\prod_{i=1}^r (1 - \frac{i}{(1-\varepsilon)n}) \geq 1 - \sum_{i=1}^r \frac{i}{(1-\varepsilon)n} = 1 - O(r^2/n)$. The summands for $r = \Omega(\sqrt{n})$ (or $r = \omega(1)$, actually) in (9) are negligible, since p_r decays exponentially in r. We plug p_0 and p_r into (9), and note that we can absorb $\sum p_r/(r + 1) \cdot O(\varepsilon)$ into the $O(\varepsilon^2)$ error term. We obtain

$$\Delta(\varepsilon) \geq \mathcal{O}(\varepsilon^2) + (1 + o(1))\varepsilon c e^{-c} \left[1 - \sum_{r=1}^{(1-\varepsilon)n} \underbrace{\frac{c^r}{(r+1)!} \cdot \frac{(1-\varepsilon)^r (r - 1)}{(1 + (\mu - 1)r)}}_{=:f(r,c,\mu)}\right]. \tag{10}$$

To bound the inner sum, we use $(r - 1)/(r + 1) \leq 1$ and obtain

$$\begin{aligned}f(r, c, \mu) &\leq \frac{c^r}{(r+1)!} \cdot \frac{r - 1}{(1 + (\mu - 1) \cdot r)}\\ &\leq \frac{c^r}{r!} \cdot \frac{1}{1 + (\mu - 1) \cdot r} \leq \frac{c^r}{r!} \cdot \frac{1}{\mu - 1}.\end{aligned}$$

We plug this bound into (10). Moreover, summing to ∞ instead of $(1 - \varepsilon)n$ only makes the expression in (10) smaller, and allows us to use the identity $\sum_{r=1}^\infty c^r/r! = e^c - 1 \leq e^c$, yielding

$$\Delta(\varepsilon) \geq \mathcal{O}(\varepsilon^2) + (1 + o(1))\varepsilon c e^{-c} \left(1 - \frac{e^c}{\mu - 1}\right).$$

If n is large enough and $\varepsilon \leq \varepsilon_0$ for a sufficiently small constant ε_0 such that the $\mathcal{O}(\varepsilon^2)$ and $o(1)$ error terms together are at most half as large as the main term, then by picking $\mu_0 = 2 + e^c$ we get $\Delta(\varepsilon) \geq \frac{1}{2}\varepsilon c e^{-c}/(e^c + 1) > 0$, and therefore, we can set $\delta = \frac{1}{2} c e^{-c}/(e^c + 1)$, which concludes the proof. □

Runtime of the $(\mu + 1)$-EA Close to the Optimum

In the previous sections, we have shown that the $(\mu + 1)$-EA has positive drift close to the optimum if the population size is chosen accordingly. In this section, we explain briefly what this result implies for the runtime of these algorithms.

Theorem 2 *Assume that the $(\mu + 1)$-EA runs on the* DynBV *function with constant parameters $c > 0$ and $\mu \geq e^c + 2$. Let*

ε_0 be as in Theorem 1 and let $\varepsilon < \varepsilon_0$ for some constant $\varepsilon > 0$. If the $(\mu + 1)$-EA is started with a population in which all individuals have at most εn zero-bits, then whp it finds the optimum in $\mathcal{O}(n \log n)$ steps.

Proof The proof is standard, e.g., [13]. First we note that the number of generations between two degenerate populations satisfies a exponential tail bound by Lemma 1. Therefore, since the number of bit flips in each generation is binomially distributed, the total number of flipped bits between two degenerate populations also satisfies an exponential tail bound, and so does the difference $|\Phi^t - \Phi^{t+1}|$. By Theorem 1 the drift of Φ^t is positive and multiplicative, $\mathbb{E}[\Phi^t - \Phi^{t+1} \mid \Phi^t] \geq \delta\Phi^t/n$ for some $\delta > 0$, as long as $\Phi^t \leq \varepsilon_0 n$. In particular, the drift in the interval $[\varepsilon n, \varepsilon_0 n]$ is at least $\varepsilon\delta > 0$, pointing towards the optimum. Since the interval has length $\varepsilon_0 n - \varepsilon n = \Omega(n)$, by the negative drift theorem [10, Theorem 10+16], whp $\Phi^t < \varepsilon_0 n$ for a super-polynomial number of steps. Hence, the process remains in a region, where the drift bound applies, and we have drift $\mathbb{E}[\Phi^t - \Phi^{t+1} \mid \Phi^t] \geq \delta\Phi^t/n$ for a super-polynomial number of steps. Therefore, by the *multiplicative drift theorem with tail bound* [4, 14] whp the optimum appears among the first $2n/\delta \cdot \log \Phi^0 \leq 2n \log n/\delta$ degenerate populations. By Lemma 1, the number of generations between two degenerate populations is a random variable with a geometric tail bound. Hence, by [3, Theorem 1.10.33], whp the number of generations is $O(n \log n)$. □

Second-Order Analysis of the Drift for $\mu = 2$

In this section, we investigate the $(2+1)$-EA. We will compute a second- order approximation of $\mathbb{E}[\Phi^t - \Phi^{t+1} \mid \Phi^t = \varepsilon n]$, that is we will compute the drift up to $\mathcal{O}(\varepsilon^3)$ error terms. This analysis will allow us to prove the following main result.

Theorem 3 *There are constants $C > 0$, $c^* > 0$ and $\varepsilon^* > 0$ such that the $(2+1)$-EA with mutation parameter c^* has positive drift $\Delta(c^*, \varepsilon) \geq C\varepsilon$ for all $\varepsilon \in (0, \frac{1}{2}\varepsilon^*)$ and has negative drift $\Delta(c^*, \varepsilon) \leq -C$ for all $\varepsilon \in (\frac{3}{2}\varepsilon^*, 2\varepsilon^*)$.*

In a nutshell, Theorem 3 shows that the hardest part for optimization is not around the optimum. In other words, it shows that the range of efficient parameters settings is larger close to the optimum. We remark that we "only" state the result for one concrete parameter c^*, but the same argument could be extended to show that the "range of efficient parameter settings" becomes larger.

All this will follow from a second-order approximation of the drift, and most of the section is devoted to this end. Let us begin by referring to Fig. 2.

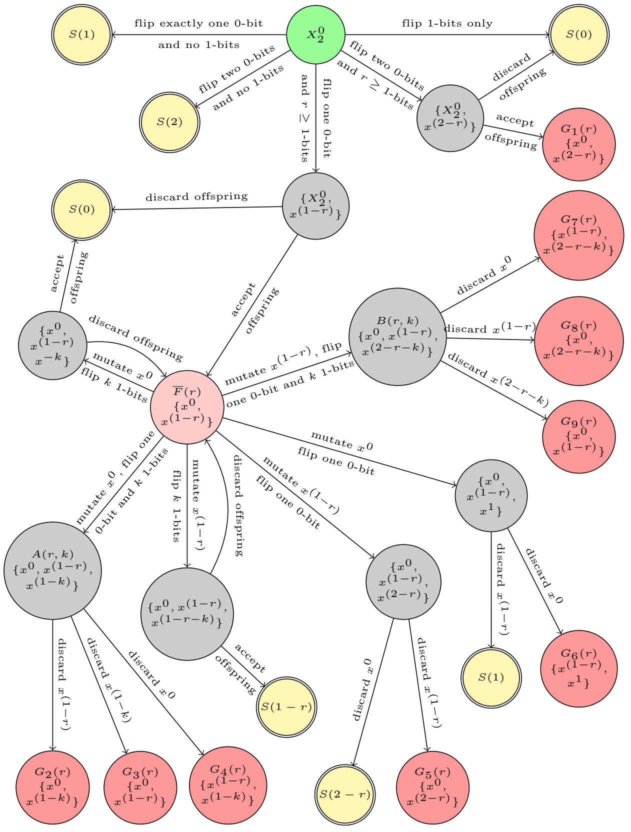

Fig. 2 State diagram for the $(2+1)$-EA up to second-order

From the size of the diagram, one can notice how quickly things get complicated further away from the optimum. On a positive note, we can compute the contribution to the drift from many of the states that the population reaches just using Lemma 4, which is tight for $\mu = 2$. As a reminder, Lemma 4 states that, given a population of two individuals for the $(2+1)$-EA, there is a closed formula for the drift, in case there are no more zero-bit flips. The cases where there happen further zero-bit flips can be summarized by $\mathcal{O}(\varepsilon)$. To get a second- order approximation for the drift, we can only apply this lemma once the population has already flipped two zero-bits (each of which give a factor $O(\varepsilon)$), so that the error term is $\mathcal{O}(\varepsilon^3)$. In particular, in Fig. 2 we have colored the states after two zero-bit flips in red. These are denoted by $G_i(r)$ for $i \in \{1, 2, .., 9\}$.

We begin by giving the intuition on how to compute some of the more challenging transition probabilities. We will often have to compute, given a population of 3 individuals, the probability for each of them to be discarded, or more precisely that it gives the least fitness value according to the DynBV function in that iteration. To compute these probabilities it is helpful to determine if any individual dominates another one, since then it will not be discarded. To compare the remaining ones, one only needs to consider all the bits in which the two or three individuals are different and do a case distinction on which of these will be in the first relative position after the permutation. Sometimes, it is not enough to look at the first position only, as it could happen that two individuals share the same value in that position and only the third is different.

The first goal will be to compute the drift from state $\overline{F}(r)$, depicted in light red in Fig. 2. This state corresponds to $F^{\mu}(r)$ from Fig. 1 for $\mu = 2$, but the continuation is more complicated now. It is reached from a degenerate state if exactly one one-bit and $r \geq 1$ zero-bits are flipped, and the offspring $x^{(1-r)}$ is accepted. From $\overline{F}(r)$, we can reach two states $A(r, k)$ and $B(r, k)$ by mutating x^0 or $x^{(1-r)}$, respectively, and flipping one zero-bit and $k \geq 1$ one-bits. We will start our analysis by computing the contribution to the drift once the population reaches states $A(r, k)$ and $B(r, k)$. For brevity, we denote

$$\Delta_A := \Delta(A(r,k), \varepsilon), \quad \Delta_B := \Delta(B(r,k), \varepsilon),$$
$$\Delta_i := \Delta(G_i(r), \varepsilon) \text{ for } \quad i = 1, \dots, 9$$

To ease reading, we simply write the probability of discarding an individual x as Pr(discard$x$), without specifying the rest of the population. From Fig. 2, it is clear that:

$$\Delta_A = \Pr(\text{discard} \quad x^{(1-r)}) \cdot \Delta_2 + \Pr(\text{discard} \quad x^{(1-k)}) \cdot \Delta_3 + \Pr(\text{discard} \quad x^0) \cdot \Delta_4$$

As discussed at the beginning of this section, we can simply use Lemma 4 to compute:

$$\begin{aligned}
\Delta_2 &= \frac{1-k}{k+1} + \mathcal{O}(\varepsilon) \\
\Delta_3 &= \frac{1-r}{r+1} + \mathcal{O}(\varepsilon) \\
\Delta_4 &= \frac{k+1}{k+r+2} \cdot (1-r) \\
&\quad + \frac{r+1}{k+r+2} \cdot (1-k) + \mathcal{O}(\varepsilon) \\
&= \frac{2-2rk}{k+r+2} + \mathcal{O}(\varepsilon)
\end{aligned} \tag{11}$$

Next up, are the probabilities to discard each individual. For that, we will introduce some notation similar as in Definition 1. We sort the positions descendingly according to the next fitness function f^t. For $i \in \{r, k\}$, the following notation applies to state $A(r, k)$ and with respect to f^t.

F_3 := first among the $r + k$
- $+ 2$ positions in which $x^0, x^{(1-k)}$ and $x^{(1-r)}$ differ.

F_2^i := first among the i
- $+ 1$ positions in which x^0 and $x^{(1-i)}$ differ·

B_i^0 := set of the i positions, where $x^{(1-i)}$
- has additional zero-bits over the others ·

B_i^1 := position where $x^{(1-i)}$
- has the single additional one-bit over the others·

The probability that $x^{(1-r)}$ is discarded can be computed in the same way as in the proof of Lemma 4:

$$\begin{aligned}
\Pr(\text{discard } x^{(1-r)}) &= \Pr(F_3 \in B_r^0) + \Pr(F_3 = B_k^1) \cdot \Pr(F_2^r \in B_r^0 \mid F_3 = B_k^1) \\
&= \frac{r}{r+k+2} + \frac{1}{r+k+2} \cdot \frac{r}{r+1} = \frac{r(r+2)}{(r+k+2)(r+1)}.
\end{aligned} \tag{12}$$

Similarly, we have:

$$\begin{aligned}
\Pr(\text{discard } x^{(1-k)}) &= \Pr(F_3 \in B_k^0) + \Pr(F_3 = B_r^1) \cdot \Pr(F_2^k \in B_k^0 \mid F_3 = B_r^1) \\
&= \frac{k}{r+k+2} + \frac{1}{r+k+2} \cdot \frac{k}{k+1} = \frac{k(k+2)}{(r+k+2)(k+1)}
\end{aligned} \tag{13}$$

and

$$\begin{aligned}
\Pr(\text{discard } x^0) &= \Pr(F_3 = B_k^1) \cdot \Pr(F_2^r = B_r^1 \mid F_3 = B_k^1) \\
&\quad + \Pr(F_3 = B_r^1) \cdot \Pr(F_2^k = B_k^1 \mid F_3 = B_r^1) \\
&= \frac{1}{r+k+2} \cdot \frac{1}{r+1} + \frac{1}{r+k+2} \cdot \frac{1}{k+1} = \frac{1}{(r+1)(k+1)}.
\end{aligned} \tag{14}$$

Putting (11), (12), (13) and (14) together yields the drift Δ_A:

$$\begin{aligned}\Delta_A &= \Pr(\text{discard}\quad x^{(1-r)}) \cdot \Delta_2 + \Pr(\text{discard}\quad x^{(1-k)}) \cdot \Delta_3 + \Pr(\text{discard}\quad x^{0}) \cdot \Delta_4 \\ &= \mathcal{O}(\varepsilon) + \frac{r(r+2)(1-k) + k(k+2)(1-r) + 2 - 2rk}{(r+k+2)(r+1)(k+1)}.\end{aligned}$$

Following the same exact procedures, we can compute Δ_B.
In particular, we again have:

$$\Delta_B = \Pr(\text{discard}\quad x^{0}) \cdot \Delta_7 + \Pr(\text{discard}\quad x^{(1-r)}) \cdot \Delta_8 + \Pr(\text{discard}\quad x^{(2-r-k)}) \cdot \Delta_9$$

Note the abuse of notation, where we omitted the rest of the population. In particular, the above probabilities are not the same as in the previous part of the proof, since the underlying population is different. We begin by applying Lemma 4, which yields:

$$\begin{aligned}\Delta_7 &= \frac{k}{k+1} \cdot (1-r) + \frac{1}{k+1} \cdot (2-r-k) + \mathcal{O}(\varepsilon) = \frac{2-r-rk}{k+1} + \mathcal{O}(\varepsilon) \\ \Delta_8 &= \frac{2 \cdot (2-r-k)}{r+k+2} + \mathcal{O}(\varepsilon) \\ \Delta_9 &= \frac{1-r}{r+1} + \mathcal{O}(\varepsilon)\end{aligned} \tag{15}$$

Similarly as before, we sort the positions descendingly according to the current fitness function f^{τ}. In the following, the last three definitions are identical as above and are only restated for convenience:

- $\hat{F}_3$:= first among the $r+k+2$ positions in which $x^0, x^{(1-r)}$ and $x^{(2-r-k)}$ differ.
- $\hat{F}_2^{r+k}$:= first among the $k+1$ positions in which $x^{(1-r)}$ and $x^{(2-r-k)}$ differ.
- $\hat{B}_k^0$:= set of the k positions, where $x^{(2-r-k)}$ has additional zero-bits over the others.
- $\hat{B}_k^1$:= position, where $x^{(2-r-k)}$ has the single additional one-bit over the others.
- F_2^r := first among the $r+1$ positions in which x^0 and $x^{(1-r)}$ differ.
- B_r^0 := set of the r positions, where $x^{(1-r)}$ has additional zero-bits over x^0.
- B_r^1 := position, where $x^{(1-r)}$ has the single additional one-bit over x^0.

We can follow the same reasoning as before and compute:

$$\begin{aligned}\Pr(\text{discard}\quad x^{0}) &= \Pr(\hat{F}_3 = B_r^1) + \Pr(\hat{F}_3 = \hat{B}_k^1) \cdot \Pr(F_2^r = B_r^1 \mid \hat{F}_3 = \hat{B}_k^1) \\ &= \frac{1}{r+k+2} + \frac{1}{r+k+2} \cdot \frac{1}{r+1} = \frac{r+2}{(r+1)(r+k+2)}.\end{aligned} \tag{16}$$

$$\begin{aligned}\Pr(\text{discard}\quad x^{(1-r)}) &= \Pr(\hat{F}_3 \in B_r^0) \cdot \Pr(\hat{F}_2^{r+k} = \hat{B}_k^1 \mid \hat{F}_3 \in B_r^0) \\ &\quad + \Pr(\hat{F}_3 = \hat{B}_k^1) \cdot \Pr(F_2^r \in B_r^0 \mid \hat{F}_3 = \hat{B}_k^1) \\ &= \frac{r}{r+k+2} \cdot \frac{1}{k+1} + \frac{1}{r+k+2} \cdot \frac{r}{r+1} = \frac{r}{(r+1)(k+1)}.\end{aligned} \tag{17}$$

$$\begin{aligned}\Pr(\text{discard}\quad x^{(2-r-k)}) &= \Pr(\hat{F}_3 \in \hat{B}_k^0) + \Pr(\hat{F}_3 \in B_r^0) \cdot \Pr(\hat{F}_2^{r+k} \in \hat{B}_k^0 \mid \hat{F}_3 \in B_r^0) \\ &= \frac{k}{r+k+2} + \frac{r}{r+k+2} \cdot \frac{k}{k+1} = \frac{k(r+k+1)}{(k+1)(r+k+2)}.\end{aligned} \tag{18}$$

Combining (15), (16), (17) and (18), we obtain:

$$\begin{aligned}\Delta_B &= \Pr(\text{discard } x^0)\cdot\Delta_7 + \Pr(\text{discard } x^{(1-r)})\cdot\Delta_8 + \Pr(\text{discard } x^{(2-r-k)})\cdot\Delta_9\\ &= \mathcal{O}(\varepsilon) + \frac{(r+2)(2-r-rk)+2r(2-r-k)+k(1-r)(r+k+1)}{(k+1)(r+1)(k+r+2)}\\ &= \mathcal{O}(\varepsilon) + \frac{-2r^2k-rk^2-3r^2-4rk+k^2+4r+k+4}{(k+1)(r+1)(k+r+2)}.\end{aligned}$$

Next up, we compute the contribution to the drift $\Delta_F := \Delta(F(r),\varepsilon)$ from state $F(r)$. Using Lemma 4, we get:

$$\begin{aligned}\Delta_F = \mathcal{O}(\varepsilon^2) &+ \frac{1}{2}p_0\left[\tfrac{1}{r+1}\cdot\Delta_6 + \tfrac{r}{r+1}\cdot 1\right] + \frac{1}{2}p^{k\cdot 1}\left[\tfrac{r}{r+1}\cdot\Delta_5 + \tfrac{1}{r+1}\cdot(2-r)\right]\\ &+ \sum_{k=0}^{(1-\varepsilon)n}\Bigg(\frac{1}{2}p^{k\cdot 1}\left[\tfrac{r}{r+k+1}\cdot 0 + \tfrac{k+1}{r+k+1}\cdot\Delta_F\right]\\ &+ \frac{1}{2}p^{k\cdot 1}\left[\tfrac{1}{r+k+1}\cdot(1-r) + \tfrac{k+r}{r+k+1}\cdot\Delta_F\right] + \frac{1}{2}p_0^{k\cdot 1}\Delta_A(k) + \frac{1}{2}p_0^{k\cdot 1}\Delta_B(k)\Bigg).\end{aligned}$$

$$\Delta_5 = \frac{2\cdot(2-r)}{2+r} + \mathcal{O}(\varepsilon).$$

$$\Delta_6 = \frac{r+1}{r+2}\cdot(1-r) + \frac{1}{r+2}\cdot 1 + \mathcal{O}(\varepsilon) = \frac{2-r^2}{r+2} + \mathcal{O}(\varepsilon). \tag{19}$$

To compute Δ_F, we first name and compute some probabilities for the outcome of a mutation. In general, the probability to flip i zero-bits and j one-bits for constant i and j is $\binom{\varepsilon n}{i}\binom{(1-\varepsilon)n}{j}(c/n)^{i+j}(1-c/n)^{n-i-j} = (1+o(1))\varepsilon^i(1-\varepsilon)^j/(i!j!)e^{-c}$ by [3]. In particular

- $p_0 := \Pr(\text{flip exactly one zero-bit}) = (1+o(1))c\varepsilon e^{-c}$.
- $p^{k\cdot 1} := \Pr(\text{flip}k\text{ one-bits}) = (1+o(1))\frac{c^k}{k!}e^{-c}$.
- $p_0^{k\cdot 1} := \Pr(\text{flip one zero-bit and } k \text{ one-bits}) = (1+o(1))\varepsilon\frac{c^{k+1}}{k!}e^{-c}$.

We are finally ready to compute Δ_F from Fig. 2. As usual, Lemma 2 allows us to summarize the contribution of all states that are not shown in Fig. 2 by $\mathcal{O}(\varepsilon^2)$. The factor 1/2 comes from the choice of the parent x^0 or $x^{(1-r)}$, and the inner factors $1/(r+1)$, $r/(r+1)$ etc. correspond the probabilities of following the subsequent arrows as depicted in Fig. 2. The six summands correspond to the cases of flipping exactly one zero-bit (in x^0 or $x^{(1-r)}$), flipping k one-bits (in x^0 or $x^{(1-r)}$), and flipping one zero-bit and k one-bits (in x^0 or $x^{(1-r)}$), in this order.

We simplify and sort the expression:

$$\begin{aligned}\Delta_F = \mathcal{O}(\varepsilon^2) &+ \frac{1}{2}p_0\left[\frac{\Delta_6+2-r}{r+1} + \frac{r(1+\Delta_5)}{r+1}\right] + \Delta_F\left(\sum_{k=0}^{(1-\varepsilon)n}\frac{1}{2}p^{k\cdot 1}\frac{2k+r+1}{r+k+1}\right)\\ &+ \sum_{k=0}^{(1-\varepsilon)n}\left(\frac{1}{2}p^{k\cdot 1}\frac{(1-r)}{r+k+1} + \frac{1}{2}p_0^{k\cdot 1}\big(\Delta_A(k)+\Delta_B(k)\big)\right).\end{aligned}$$

Now we solve for Δ_F by bringing all Δ_F-terms on the left hand side and dividing by its prefactor. We obtain:

$$\Delta_F = \mathcal{O}(\varepsilon^2) + \frac{p_0\frac{2+\Delta_6+r\cdot\Delta_5}{r+1} + \sum_{k=0}^{(1-\varepsilon)n}p^{k\cdot 1}\frac{(1-r)}{r+k+1} + p_0^{k\cdot 1}\big(\Delta_A(k)+\Delta_B(k)\big)}{2-\sum_{k=0}^{(1-\varepsilon)n}p^{k\cdot 1}\frac{2k+r+1}{r+k+1}}.$$

For later reference we note that the $p^{k\cdot 1}$ sum up to one, i.e., $\sum_{k=0}^{(1-\varepsilon)n}p^{k\cdot 1} = 1$. Thus we can rewrite $2 = \sum_k p^{k\cdot 1}\frac{2r+2k+2}{r+k+1}$, which allows us to rewrite the denominator as $\sum_{k=0}^{(1-\varepsilon)n}p^{k\cdot 1}\frac{r+1}{r+k+1}$. This trick will allow us some cancellations later. In particular, note that the probabilities p_0 and $p_0^{k\cdot 1}$ are in $\mathcal{O}(\varepsilon)$, so when we ignore those terms, then the complicated sums cancel out and we recover the formula $\Delta_F = (1-r)/(r+1) + \mathcal{O}(\varepsilon)$ from Lemma 8.

Finally, we can find the drift $\Delta(c,\varepsilon)$ from the starting population X_2^0. To this end, we need two more probabilities:

- $p_{2\cdot 0} := \Pr(\text{flip two zero-bits}) = (1+o(1))\frac{1}{2}\varepsilon^2c^2e^{-c}$
- $p_{2\cdot 0}^{r\cdot 1} := \Pr(\text{flip two zero-bits and r one-bits}) = (1+o(1))\frac{1}{2}\varepsilon^2e^{-c}\frac{c^{r+2}}{r!}$

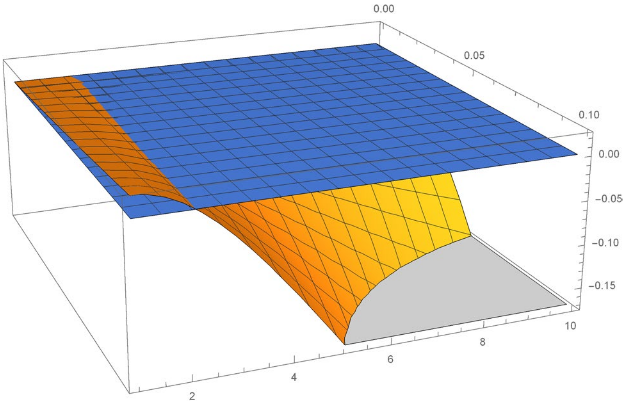

Fig. 3 Plot of the second-order approximation (orange) for the drift of the $(2+1)$-EA. The x-axis is the mutation parameter c, the y-axis is the distance ε from the optimum. The blue plane is the 0 plane. The interesting part is the line of intersection between the blue and orange surface, as this is boundary between positive and negative drift. Looking closely, the intersection moves to the left (smaller c) if we move to the front (larger ε). Thus the problem becomes harder (smaller threshold for c) as we increase ε. Hence, the hardest part is not around the optimum. In particular, for some choices of the mutation parameter c (e.g., $c = 2.2$) the drift of the $(2+1)$-EA is positive in a region around the optimum, but is negative further away from the optimum. We prove this surprising result below, and it is in line with the experimental results found in [16]

Again, Fig. 2, we can calculate $\Delta(c,\varepsilon)$. In the following calculation, we use the full notation $\Delta_{F(r)} = \Delta_F$ to make the dependency on r explicit. Moreover, note that we already have computed the drift from state $G_1(r)$, since this is identical with $G_5(r)$, so the drift is Δ_5.

$$\begin{aligned}\Delta(c,\varepsilon) &= \mathcal{O}(\varepsilon^3) + p_0 \cdot 1 + p_{2\cdot 0} \cdot 2 + \sum_{r=0}^{(1-\varepsilon)n} p^{r\cdot 1} \cdot 0 \\ &\quad + \sum_{r=1}^{(1-\varepsilon)n} \left(p_0^{r\cdot 1}\Big(\tfrac{1}{r+1} \cdot \Delta_{F(r)} + \tfrac{r}{r+1} \cdot 0\Big) + p_{2\cdot 0}^{r\cdot 1}\Big(\tfrac{2}{2+r} \cdot \Delta_5 + \tfrac{r}{r+2} \cdot 0\Big)\right) \\ &= \mathcal{O}(\varepsilon^3) + p_0 + 2p_{2\cdot 0} + \sum_{r=1}^{(1-\varepsilon)n} \frac{p_0^{r\cdot 1}}{r+1}\Delta_{F(r)} + \frac{2p_{2\cdot 0}^{r\cdot 1}}{2+r}\Delta_5.\end{aligned}$$

The last step is to plug in the formulas for the probabilities and sort terms. In addition, letting the sums go to ∞ instead of $(1-\varepsilon)n$ will only add another factor of $(1+o(1))$. Thus we can rewrite the drift as:

$$\Delta(c,\varepsilon) = \varepsilon(1+o(1))f_0(c) + \varepsilon^2(1+o(1))f_1(c) + \mathcal{O}(\varepsilon^3), \quad (20)$$

where

$$\begin{aligned} f_0(c) &= ce^{-c} + \sum_{r=1}^{\infty} \frac{c^{r+1}}{r!}e^{-c} \cdot \frac{1}{r+1} \cdot \frac{\sum_{k=0}^{\infty} p^{k\cdot 1}\frac{(1-r)}{r+k+1}}{\sum_{k=0}^{\infty} p^{k\cdot 1}\frac{r+1}{r+k+1}} \\ &= ce^{-c} \cdot \left(1 + \sum_{r=1}^{\infty} \frac{c^r}{(r+1)!} \cdot \frac{1-r}{r+1}\right) \end{aligned} \quad (21)$$

and

$$\begin{aligned} f_1(c) &= c^2e^{-c} + \sum_{r=1}^{\infty}(r+1)e^{-c}\frac{c^{r+2}}{(r+2)!}\Delta_5 \\ &\quad + \frac{e^{-2\cdot c}}{2}\sum_{r=1}^{\infty}\frac{c^{r+2}}{(r+1)!} \cdot \frac{\frac{2+\Delta_6+r\Delta_5}{r+1} + \sum_{k=0}^{\infty}\frac{c^k}{k!}\big(\Delta_A(k)+\Delta_B(k)\big)}{\sum_{k=0}^{\infty}\frac{c^k}{k!}e^{-c}\frac{r+1}{r+k+1}}. \end{aligned} \quad (22)$$

After all this preliminary calculations, we are now ready to prove Theorem 3. Moreover, we plot the second-order approximation of the drift numerically with Wolfram Mathematica in Fig. 3.

Proof of Theorem 3 Recall that the second-order approximation of the drift is given by (20), (21) and (22). Inspecting (21), we see that the sum goes over negative terms, except for the term for $r = 1$ which is zero. Thus the factor in the bracket is strictly decreasing in c, ranging from 1 (for $c = 0$) to $-\infty$ (for $c \to \infty$). In particular, there is exactly one $c_0 > 0$ such that $f_0(c_0) = 0$. Numerically we find $c_0 = 2.4931\ldots$ and $f_1(c_0) = -0.4845\ldots < 0$.

In the following, we will fix some $c^* < c_0$ and set $\varepsilon^* := -f_0(c^*)/f_1(c^*)$. Note that by choosing c^* sufficiently close to c_0 we can assume that $f_1(c^*) < 0$, since f_1 is a continuous function. Due to the discussion of f_0 above, the choice $c^* < c_0$ also implies $f_0(c^*) > 0$. Thus $\varepsilon^* > 0$. Moreover, since $f_0(c) \to 0$ for $c \to c^*$, if we choose c^* close enough to c_0 then we can make ε^* as close to zero as we wish.

To add some intuition to these definitions, note that $\Delta(c,\varepsilon) = \varepsilon(f_0(c) + \varepsilon f_1(c) + \mathcal{O}(\varepsilon^2))$, so the condition $\varepsilon = -f_0(c)/f_1(c)$ is a choice for ε for which the drift is approximately zero, up to the error term. We will indeed prove that for fixed c^*, the sign of the drift switches around $\varepsilon \approx \varepsilon^*$. More precisely, we will show that the sign switches from positive to negative as we go from $\Delta(c^*, \varepsilon^* - \varepsilon')$ to $\Delta(c^*, \varepsilon^* + \varepsilon')$, for $\varepsilon' \in (0, \varepsilon^*)$. Actually, we will constrict to $\varepsilon' \in (\varepsilon^*/2, \varepsilon^*)$ so that we can handle the error terms. This implies that the value c^* yields positive drift close to the optimum (in the range $\varepsilon \in (0, \frac{1}{2}\varepsilon^*)$), but yields negative drift further away from the optimum (in the range $\varepsilon \in (\frac{3}{2}\varepsilon^*, 2\varepsilon^*)$). This implies Theorem 3.

To study the sign of the drift, we define

$$\Delta^*(c, \varepsilon) := \frac{\Delta(c, \varepsilon)}{\varepsilon} = (1 + o(1)) \cdot \big(f_0(c) + \varepsilon \cdot f_1(c) + O(\varepsilon^2)\big).$$

It is slightly more convenient to consider Δ^* instead of Δ, but note that both terms have the same sign. Therefore, it remains to investigate the sign of $\Delta^*(c^*, \varepsilon^* - \varepsilon')$ and $\Delta^*(c^*, \varepsilon^* + \varepsilon')$ for $\varepsilon' \in (\varepsilon^*/2, \varepsilon^*)$. We will show that $\Delta^*(c^*, \varepsilon^* - \varepsilon') > 0$, and the inequality $\Delta^*(c^*, \varepsilon^* - \varepsilon') < 0$ follows analogously. Recalling the definition of ε^* and that $f_1(c^*) < 0$, we have

$$\begin{aligned}\Delta^*(c^*, \varepsilon^* - \varepsilon') &= (1 + o(1))\big(f_0(c^*) + (\varepsilon^* - \varepsilon')f_1(c^*)\big) + \mathcal{O}((\varepsilon^*)^2)\\ &= (1 + o(1))\underbrace{\big(f_0(c^*) + \varepsilon^* f_1(c^*)\big)}_{=0} - (1 + o(1))\underbrace{\big(\varepsilon' f_1(c^*)\big)}_{<\varepsilon^* f_1(c^*)/2} + \mathcal{O}((\varepsilon^*)^2)\\ &> -(1 + o(1))\tfrac{1}{2}\varepsilon^* f_1(c^*) + \mathcal{O}((\varepsilon^*)^2).\end{aligned}$$

Recall that we may choose the constant ε^* as small as we want. In particular, we can choose it so small that the above term has the same sign as the main term, which is positive due to $f_1(c^*) < 0$. Hence $\Delta^*(c^*, \varepsilon^* - \varepsilon') > 0$, as desired. In addition, note that the lower bound is independent of ε', i.e., it holds uniformly for all $\varepsilon' \in (\varepsilon^*/2, \varepsilon^*)$, which corresponds to the argument $\varepsilon^* - \varepsilon'$ of Δ^* to be in the interval $(0, \varepsilon^*/2)$. The inequality $\Delta^*(c^*, \varepsilon^* + \varepsilon') < 0$ follows analogously. This concludes the proof. □

We conclude the section by a theorem stating that the drift translates immediately into runtimes.

Theorem 4 *Let c^*, ε^* be the constants from Theorem 3. Then whp the $(2 + 1)$-EA with mutation parameter c^* finds the optimum in $O(n \log n)$ if it is started in distance at most $\varepsilon^* n/4$ zero-bits, but does not find the optimum in polynomial time if it is started in distance at least $2\varepsilon^* n$ from the optimum.*

Proof The proof is almost identical to the proof of Theorem 2. We again observe that the difference $|\Phi^t - \Phi^{t+1}|$ satisfies exponential tail bounds so that the negative drift theorem [10, Theorem 16] is applicable by [10, Theorem 10]. For the negative result, it suffices to observe that the drift of Φ^t is negative by Theorem 3 and apply [10, Theorem 16]. For the positive result, by Theorem 3 the drift of Φ^t is positive as long as $\Phi^t \le \varepsilon^* n/2$. As in the proof of Theorem 2, starting from below $\varepsilon^* n/4$, by the negative drift theorem whp Φ^t stays below $\varepsilon^* n/2$ for a superpolynomial number of steps. Hence the algorithm stays in a region, where we have the drift bound $\mathbb{E}[\Phi^t - \Phi^{t+1} \mid \Phi^t] \ge C\Phi^t/n$. By the multiplicative drift theorem with tail bound [4, 14], whp the optimum appears among the first $2n/C \cdot \log \Phi^0 \le 2n \log n/C$ degenerate populations. By Lemma 1, the number of generations between two degenerate populations is a random variable with a geometric tail bound. Hence, by [3, Theorem 1.10.33], whp the number of generations is $O(n \log n)$. □

Conclusions

We have explored the DynBV function, and we have found that the $(\mu + 1)$-EA profits from large population size, close to the optimum. In particular, for all choices of the mutation parameter c, the $(\mu + 1)$-EA is efficient around the optimum if μ is large enough. However, surprisingly the region around the optimum may not be the most difficult region. For $\mu = 2$, we have proven that it is not.

This surprising result, in line with the experiments in [16], raises much more questions than it answers. Does the $(\mu + 1)$-EA with increasing μ turn efficient for a larger and larger range of c, as the behavior around the optimum suggests? Or is the opposite true, that the range of efficient c shrinks to zero as the population grows, as it is the case for the $(\mu + 1)$-EA on HotTopic functions? Where is the hardest region for larger μ? Around the optimum or elsewhere?

For the $(\mu + 1)$-GA, the picture is even less complete. Experiments in [16] indicated that the hardest region of DynBV for the $(\mu + 1)$-GA is around the optimum, and that the range of efficient c increases with μ. However, the experiments were only run for $\mu \le 5$, and formal proofs are missing. Should we expect that the discrepancy between $(\mu + 1)$-GA (hardest region around optimum) and $(\mu + 1)$-EA (hardest region elsewhere) remains if we increase the population size, and possibly becomes stronger? Or does it disappear? For HotTopic functions, we know that around the optimum, the range of efficient c becomes arbitrarily large as μ grows (similarly as we have shown for the $(\mu + 1)$-EA on DynBV), but we have no idea what the situation away from the optimum is.

The similarities of results between DynBV and HotTopic functions are striking, and we are pretty clueless, where they come from. For example, the analysis of the $(\mu + 1)$-EA on HotTopic away from the optimum in [21] clearly does not generalize to DynBV, since the very heart of the proof is that the weights do not change over long periods. In DynBV, they change every round. Nevertheless, experiments and theoretical results indicate that the outcome is similar in both cases. Perhaps one could gain insight from "interpolating" between DynBV and HotTopic by re-drawing the weights not every round, but only every kth round.

In general, the situation away from the optimum is governed by complex population dynamics, which is why the $(\mu + 1)$-EA and the $(\mu + 1)$-GA might behave very differently. Currently, we lack the theoretic means to understand population dynamics in which the internal population structure is complex and essential. The authors believe that developing tools for understanding such dynamics is one of the most important projects for improving our understanding of population-based search heuristics.

Funding Open access funding provided by ETH Zürich.

Declarations

Conflict of Interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Colin S, Doerr B, Férey G. Monotonic functions in EC: anything but monotone! In: Genetic and evolutionary computation Conference (GECCO). ACM; 2014. pp. 753–760.
2. Dang-Nhu R, Dardinier T, Doerr B, Izacard G, Nogneng D. A new analysis method for evolutionary optimization of dynamic and noisy objective functions. In: Genetic and Evolutionary Computation Conference (GECCO). ACM; 2018. pp. 1467–1474.
3. Doerr B. Probabilistic tools for the analysis of randomized optimization heuristics. In: Theory of evolutionary computation. Springer; 2020. pp. 1–87.
4. Doerr B, Goldberg LA. Adaptive drift analysis. Algorithmica. 2013;65(1):224–50.
5. Doerr B, Hota A, Kötzing T. Ants easily solve stochastic shortest path problems. In: Genetic and Evolutionary Computation Conference (GECCO). ACM; 2012. pp. 17–24.
6. Doerr B, Jansen T, Sudholt D, Winzen C, Zarges C. Mutation rate matters even when optimizing monotonic functions. Evol Comput. 2013;21(1):1–27.
7. Droste S. Analysis of the $(1 + 1)$ EA for a dynamically changing ONEMAX-variant. In: Congress on Evolutionary Computation (CEC), vol. 1, IEEE; 2002. pp. 55–60.
8. Horoba C, Sudholt D. Ant colony optimization for stochastic shortest path problems. In: Genetic and Evolutionary Computation Conference (GECCO). ACM; 2010. pp. 1465–1472.
9. Jansen T. On the brittleness of evolutionary algorithms. In: Foundations of genetic algorithms (FOGA). Springer; 2007. pp. 54–69.
10. Kötzing T. Concentration of first hitting times under additive drift. Algorithmica. 2016;75(3):490–506.
11. Kötzing T, Lissovoi A, Witt C. (1+1) EA on generalized dynamic OneMax. In: Foundations of genetic algorithms (FOGA). Springer; 2015. pp. 40–51.
12. Kötzing T, Molter H. ACO beats EA on a dynamic pseudo-boolean function. In: Parallel problem solving from nature (PPSN). Springer; 2012. pp. 113–122.
13. Lengler J. A general dichotomy of evolutionary algorithms on monotone functions. IEEE Trans Evol Comput. 2019;24(6):995–1009.
14. Lengler J. Drift analysis. In: Theory of evolutionary computation. Springer; 2020. pp. 89–131.
15. Lengler J, Martinsson A, Steger A. When does hillclimbing fail on monotone functions: an entropy compression argument. In: Analytic algorithmics and combinatorics (ANALCO). SIAM; 2019. pp. 94–102.
16. Lengler J, Meier J. Large population sizes and crossover help in dynamic environments. In: Parallel problem solving from nature (PPSN). Springer; 2020. pp. 610–622.
17. Lengler J, Meier J. Large population sizes and crossover help in dynamic environments, full version. arXiv preprint; 2020. http://arxiv.org/abs/2004.09949
18. Lengler J, Riedi S. Runtime analysis of the $(\mu + 1)$-EA on the Dynamic BinVal function. In: Evolutionary computation in combinatorial optimization (EvoCOP). Springer; 2021 pp. 84–99.
19. Lengler J, Schaller U. The (1+1)-EA on noisy linear functions with random positive weights. In: Symposium Series on Computational Intelligence (SSCI). IEEE; 2018. pp. 712–719.
20. Lengler J, Steger A. Drift analysis and evolutionary algorithms revisited. Combinatorics Probab Comput. 2018;27(4):643–66.
21. Lengler J, Zou X. Exponential slowdown for larger populations: the $(\mu+1)$-EA on monotone functions. In: Foundations of genetic algorithms (FOGA). ACM; 2019. pp. 87–101.
22. Lissovoi A, Witt C. MMAS versus population-based EA on a family of dynamic fitness functions. Algorithmica. 2016;75(3):554–76.
23. Lissovoi A, Witt C. A runtime analysis of parallel evolutionary algorithms in dynamic optimization. Algorithmica. 2017;78(2):641–59.

24. Lissovoi A, Witt C. The impact of a sparse migration topology on the runtime of island models in dynamic optimization. Algorithmica. 2018;80(5):1634–57.
25. Neumann F, Pourhassan M, Roostapour V. Analysis of evolutionary algorithms in dynamic and stochastic environments. In: Theory of evolutionary computation. Springer; 2020. pp. 323–357.
26. Neumann F, Witt C. On the runtime of randomized local search and simple evolutionary algorithms for dynamic makespan scheduling. In: International Joint Conference on Artificial Intelligence (IJCAI). AAAI Press; 2015 pp. 3742–3748.
27. Pourhassan M, Gao W, Neumann F. Maintaining 2-approximations for the dynamic vertex cover problem using evolutionary algorithms. In: Genetic and Evolutionary Computation Conference (GECCO). ACM; 2015. pp. 903–910.
28. Shi F, Schirneck M, Friedrich T, Kötzing T, Neumann F. Reoptimization time analysis of evolutionary algorithms on linear functions under dynamic uniform constraints. Algorithmica. 2019;81(2):828–57.

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.