Precise Runtime Analysis for Plateaus

Denis Antipov[1(✉)] and Benjamin Doerr[2]

[1] ITMO University, 49 Kronverkskiy prosp., 197101 Saint-Petersburg, Russia
antipovden@yandex.ru
[2] École Polytechnique, CNRS, Laboratoire d'Informatique (LIX), Palaiseau, France
doerr@lix.polytechnique.fr

Abstract. To gain a better theoretical understanding of how evolutionary algorithms cope with plateaus of constant fitness, we analyze how the $(1 + 1)$ EA optimizes the n-dimensional $\textsc{Plateau}_k$ function. This function has a plateau of second-best fitness in a radius of k around the optimum. As optimization algorithm, we regard the $(1 + 1)$ EA using an arbitrary unbiased mutation operator. Denoting by α the random number of bits flipped in an application of this operator and assuming $\Pr[\alpha = 1] = \Omega(1)$, we show the surprising result that for $k \geq 2$ the expected optimization time of this algorithm is

$$\frac{n^k}{k! \Pr[1 \leq \alpha \leq k]}(1 + o(1)),$$

that is, the size of the plateau times the expected waiting time for an iteration flipping between 1 and k bits. Our result implies that the optimal mutation rate for this function is approximately k/en. Our main analysis tool is a combined analysis of the Markov chains on the search point space and on the Hamming level space, an approach that promises to be useful also for other plateau problems.

Keywords: Runtime analysis · Theory · Markov chains · Mutation

1 Introduction

This work aims at making progress on several related subjects—we aim at understanding how evolutionary algorithms optimize non-unimodal[1] fitness functions, what mutation operators to use in such settings, how to analyze the behavior of evolutionary algorithms on large plateaus of constant fitness, and in particular, how to obtain runtime bounds that are precise including the leading constant.

The recent work [14] observed that a large proportion of the theoretical work in the past concentrates on analyzing of how evolutionary algorithms optimize unimodal fitness functions and that this can lead to misleading recommendations how to design evolutionary algorithms. Based on a precise analysis of how

[1] As common in optimization, we reserve the notion *unimodal* for objective functions such that each non-optimal search point has a strictly better neighbor.

A. Auger et al. (Eds.): PPSN 2018, LNCS 11102, pp. 117–128, 2018.
https://doi.org/10.1007/978-3-319-99259-4_10

the $(1+1)$ EA optimizes jump functions, it was observed that the classic recommendation to use standard bit mutation with mutation rate $\frac{1}{n}$ is far from optimal for this function class. For jump size k, a speed-up of order $k^{\Theta(k)}$ can be obtained from using a mutation rate of $\frac{k}{n}$.

Jump functions are difficult to optimize, because the optimum is surrounded by a large set of search points of very low fitness (all search points in Hamming distance 1 to $k-1$ from the optimum). However, this is not the only reason for fitness functions being difficult. Another challenge for most evolutionary algorithms are large plateaus of constant fitness. On such plateaus, the evolutionary algorithm learns little from evaluating search points and consequently performs an unguided random walk. To understand this phenomenon in more detail, we propose a class of fitness function very similar to jump functions. A *plateau function* with plateau parameter k is identical to a jump function with jump size k except that the $k-1$ Hamming levels around the optimum do not have a small fitness, but have the same second-best fitness as the k-th Hamming level. Consequently, these functions do not have true local optima (in which an evolutionary algorithm could get stuck for longer time), but only a plateau of constant fitness. Our hope is that this generic fitness function with a plateau of scalable size may aid the understanding of plateaus in evolutionary computation in a similar manner as the jump functions have led to many useful results about the optimization of functions with true local optima, e.g., [4–6,8,15,18].

When trying to analyze how evolutionary algorithms optimize plateau functions, we observe that the active area of theoretical analyses of evolutionary algorithms has produced many strong tools suitable to analyze how evolutionary algorithms make true progress (e.g., various form of the fitness level method [7,24,25] or drift analysis [13,17,20]), but much less is known on how to analyze plateaus. This is not to mean that plateaus have not been analyzed previously, see, e.g., [3,10,16], but these results appear to be more ad hoc and less suitable to derive generic methods for the analysis of plateaus. In particular, with the exception of [16], we are not aware of any results that determine the runtime of an evolutionary algorithm on a fitness function with non-trivial plateaus precise including the leading constant (whereas a decent number of very precise results have recently appeared for unimodal fitness functions, e.g., [2,9,21,26]).

Such precise results are necessary for our further goal of understanding the influence of the mutation operator on the efficiency of the optimization process. Mutation is one of the most basic building blocks in evolutionary computation and has, consequently, received significant attention also in the runtime analysis literature. We refer to the discussion in [14] for a more extensive treatment of this topic and note here only already small changes of the mutation operator or its parameters can lead to a drastic change of the efficiency of the algorithm [11,12]

Our Results: Our main result is a very general analysis of how the simplest mutation-based evolutionary algorithm, the $(1+1)$ EA, optimizes the n-dimensional plateau function with plateau parameter $k \in \mathbb{N}$. We allow the algorithm to use any unbiased mutation operator (including, e.g., 1-bit flips, standard-bit mutation with an arbitrary mutation rate, or the fast mutation

operator of [14]) as long as the operator flips exactly one bit with probability at least some positive constant. This assumption is natural, but also necessary to ensure that the algorithm can reach all points on the plateau. Denoting the number of bits flipped in an application of this operator by the random variable α, we prove that the expected optimization time (number of fitness evaluations until the optimum is visited) is

$$\frac{n^k}{k! \Pr[1 \leq \alpha \leq k]}(1 + o(1)).$$

This result, tight apart from lower order terms only, is remarkable in several respects. It shows that the performance depends very little on the particular mutation operator, only the probability to flip between 1 and k bits has an influence. The absolute runtime is also surprising—it is the size of the plateau times the waiting time for a one-to-k bit flip.

A similar-looking result was obtained in [16], namely that the expected runtime of the $(1+1)$ EA with 1-bit mutation and with standard-bit mutation with rate $\frac{1}{n}$ on the needle function is (apart from lower order terms) the size of the plateau times the probability to flip a positive number of bits (which is 1 for 1-bit mutation and $(1 - o(1))(1 - \frac{1}{e})$ for standard bit mutation with rate $1/n$). Our result thus complements this result (valid for two specific mutation operators and for the plateau of radius n around the unique optimum) with an analogous result for constrained plateaus of arbitrary (constant) radius $k \geq 2$ around the optimum and for arbitrary unbiased mutation operators.

We note that there is a substantial difference between the case $k = n$ and k constant. Since the needle function consists of a plateau containing the whole search space apart from the optimum, the optimization time in this case is just the hitting time of a particular search point when doing an undirected random walk (via repeated mutation) on the hypercube $\{0, 1\}^n$. For the function considered in this paper, the plateau has a large boundary. More precisely, almost all[2] search points of the plateau lie on its outer boundary and furthermore, all these search points have almost all their neighbors outside the plateau. Hence a large number of iterations (namely almost all) are lost in the sense that the mutation operator generates a search point outside the plateau (and different from the optimum), which is not accepted. Interestingly, as our result shows, the optimization of such restricted plateaus is not necessarily significantly more difficult (relative to the plateau size) than the optimization of the unrestricted needle plateau.

Our precise runtime analysis allows to deduce a number of particular results. For example, when using standard bit mutation, the optimal[3] mutation rate is $\frac{k}{en}$. This is by a constant factor less than the optimal rate of $\frac{k}{n}$ for the jump function with jump size k, but again a factor of $\Theta(k)$ larger than the classic recommendation of $\frac{1}{n}$, which is optimal for many unimodal fitness functions.

[2] In the usual asymptotic sense, that is, meaning all but a lower order fraction.

[3] We call a mutation rate optimal when it differs from the truly optimal rate at most by lower order terms, that is, e.g. a factor of $(1 \pm o(1))$.

Hence our result confirms that the optimal mutation rates can be significantly higher for non-unimodal fitness functions. While the optimal mutation rates for jump and plateau functions are similar, the effect of using the optimal rate is very different. For jump functions, an $k^{\Theta(k)}$ factor speed-up (compared to the standard recommendation of $\frac{1}{n}$) was observed, here the influence of the mutation operator is much smaller, namely the factor $\Pr[1 \leq \alpha \leq k]$, which is trivially at most 1, but which was assumed to be at least some positive constant. Interestingly, our results imply that the fast mutation operator described in [14] is not more effective than other unbiased mutation operators, even though it was proven to be significantly more effective for jump functions and it has shown good results in some practical problems [23].

So one structural finding, which we believe to be true for larger classes of problems and which fits to the result [16] for needle functions, is that the mutation rate, and more generally, the particular mutation operator which is used, is less important while the evolutionary algorithm is traversing a plateau of constant fitness.

The main technical novelty in this work is that we model the optimization process via two different Markov chains describing the random walk on the plateau, namely the chain defined on the $\Theta(n^k)$ elements of the plateau (plus the optimum) and the chain obtained from aggregating these into the total mass on the Hamming levels. Due to the symmetry of the process, one could believe that it suffices to regard only the level chain. The chain defined on the elements, however, has some nice features which the level chain is missing, among others, a symmetric transition matrix (because for any two search points x and y on the plateau, the probability of going from x to y is the same as the probability of going from y to x). For this reason, we find it fruitful to switch between the two chains. Exploiting the interplay between the two chains and using classic methods from linear algebra, we find the exact expression for the expected runtime.

The abstract idea of switching between the chain of all the elements of the plateau and an aggregated chain exploiting symmetries of the process as well as the linear algebra arguments we use are not specific to our particular problem. For this reason, we are optimistic that our techniques may be applied as well to other optimization processes involving plateaus[4].

2 Problem Statement

We consider the maximization of a function that resembles the OneMax function, but has a plateau of second-highest fitness of radius k around the optimum. We call this function $\textsc{Plateau}_k$ and define it as follows.

[4] For reasons of space, not all mathematical proofs could fit into this extended abstract. The proofs can be found in [1].

$$\textsc{Plateau}_k(x) := \begin{cases} \textsc{OneMax}(x), & \text{if } \textsc{OneMax}(x) \leq n-k, \\ n-k, & \text{if } n-k < \textsc{OneMax}(x) < n, \\ n, & \text{if } \textsc{OneMax}(x) = n, \end{cases}$$

where $\textsc{OneMax}(x)$ is the number of one-bits in x.

Notice that the plateau of the function $\textsc{Plateau}_k(x)$ consists of all bit-strings that have at least $n-k$ one-bits, except the optimal bit-string $x^* = (1, \ldots, 1)$. See Fig. 1 for an illustration of $\textsc{Plateau}_k$. Since a reviewer asked for it, we note that the unary unbiased black-box complexity (see [19] for the definition) of $\textsc{Plateau}_k$ is $\Theta(n \log n)$. Here the lower bound follows from the $\Omega(n \log n)$ lower bound for the unary unbiased black-box complexity of $\textsc{OneMax}$, see again [19], and the fact that $\textsc{Plateau}_k = f \circ \textsc{OneMax}$ for a suitable function f, hence any algorithm solving $\textsc{Plateau}_k$ can also solve $\textsc{OneMax}$. The upper bound follows along the same lines as the $O(n \log n)$ upper bound for the unary unbiased black-box complexity of $\textsc{Jump}_k$, see [8].

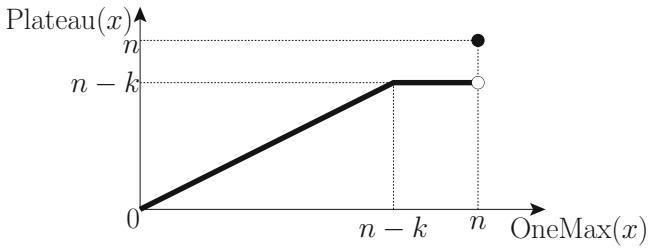

Fig. 1. Graph of the $\textsc{Plateau}$ function. As a function of unitation, the function value of a search point x depends only on the value $\|x\|_1$ of the $\textsc{OneMax}$ function.

As the optimization algorithm we consider the $(1+1)$ EA, shown in Algorithm 1, using an unbiased mutation operator. A mutation operator $\textsc{Mutate}$ for bit-string representations is called *unbiased* if it is symmetric in the bit-positions $[1..n]$ and in the bit-values 0 and 1. This is equivalent to saying that for all $x \in \{0,1\}^n$ and all automorphisms σ of the hypercube $\{0,1\}^n$ (respecting Hamming neighbors) we have $\sigma^{-1}(\textsc{Mutate}(\sigma(x)) = \textsc{Mutate}(x)$ (and this is an equality of distributions). The notation of unbiasedness was introduced (also for higher-arity operators) in the seminal paper [19].

For our purposes, it suffices to know that the set of unbiased mutation operators consists of all operators which can be described as follows. First, we choose a number $\alpha \in [0..n]$ according to some probability distribution and then we flip exactly α bits chosen uniformly at random. Examples for unbiased operators are the operator of Random Local Search, which flips a random bit, or standard bit mutation, which flips each bit independently with probability $\frac{1}{n}$. Note that in the first case α is always equal to one, whereas in the latter α follows a binomial distribution with parameters n and $\frac{1}{n}$.

Algorithm 1. The (1+1) EA with unary unbiased mutation operator MUTATE maximizing the function f

```
1: x ← random bit string of length n
2: repeat
3:     y ← Mutate(x)
4:     if f(y) ≥ f(x) then
5:         x ← y
6:     end if
7: until forever.
```

Additional Assumptions: The class of unbiased mutation operators contains a few operators which are unable to solve even very simple problems. For example, operators that always flips exactly two bits never finds the optimum of any function with unique optimum if the initial individual has an odd Hamming distance from the optimum. To avoid such difficulties, we only consider unbiased operators that have at least a constant (positive) probability to flip exactly one bit.

As usual in runtime analysis, we are interested in the optimization behavior for large problem size n. Formally, this means that we view the runtime $T = T(n)$ as a function of n and aim at understanding its asymptotic behavior for n tending to infinity. We aim at sharp results (including finding the leading constant), that is, we try to find a simple function $\tau : \mathbb{N} \to \mathbb{N}$ such that $T(n) = (1 + o(1))\tau(n)$, which is equivalent to saying that $\lim_{n\to\infty} T(n)/\tau(n) = 1$. In this limit sense, however, we treat k as a constant, that is, k is a given positive integer and not also a function of n. Since the case $k = 1$ is well-understood (PLATEAU_1 is the well-known ONEMAX function), we always assume $k \geq 2$.

3 Preliminaries and Notation

As long as the unbiased operator with constant probability flips exactly one bit, the expected time to reach the plateau is $O(n \log n)$. Since the time for leaving the plateau (as shown in this paper) is $\Omega(n^k)$, we only consider the runtime of the algorithm after it has reached the plateau.

For our precise runtime analysis on the plateau we consider the plateau in two different ways. The first way is to regard a Markov chain that contains $N = \sum_{i=0}^{k-1} \binom{n}{k-i}$ states, where each state represents one element of the plateau. Note that $N = \frac{n^k}{k!} + o(n^k)$, since $\binom{n}{j} = \frac{n^j}{j!}(1 + o(1))$ for all $j \in [1..k]$. Since we only regard unbiased mutation operators, the transition probability between two elements depends only on the Hamming distance between these elements. The transition matrix P_{ind} for this chain is large and inconvenient to work with. However this matrix is symmetric due to unbiasedness of the operator that implies for mutating from individual x to individual y we need to flip exactly the same set of bits as for mutating from individual y to individual x. The symmetry of this matrix will give us some simplifications in our analysis. For

example we will use the fact that the eigenvectors of this matrix are orthogonal. We intentionally do not include optimum into this chain to understand, how the algorithm behaves before escaping the plateau. As a result, the sum of each row in P_{ind} may be less than one. We call this chain the *individual chain* and call the space of real vectors of dimension N the *individual space*, since it is the space of all possible current individuals, when the algorithm is on the plateau.

To define the second Markov chain we shall use, let us first define the *i-th level* as the set of all the search points that have exactly $n-k+i$ one-bits. Then the plateau is the union of levels 0 to $k-1$ and the optimum is level k. Notice that the i-th level contains exactly $\binom{n}{k-i}$ elements. For every i and $j \in [0..k]$ and any element of the i-th level the probability to mutate to the j-th level is the same due to the unbiasedness of the operator. Therefore we can regard a Markov chain of k states, where i-th state represents all the elements of the i-th level. We call the transition matrix of this chain P. This matrix has a size of $k \times k$. Matrix P (unlike P_{ind}) is not symmetric. Like in the individual chain, since the optimum is not represented by any state, the sum of all the outgoing transition probabilities for each state may be less than one. We call this chain the *level chain* and we call the space of real vectors of length k the *level space*. The level chain is illustrated in Fig. 2.

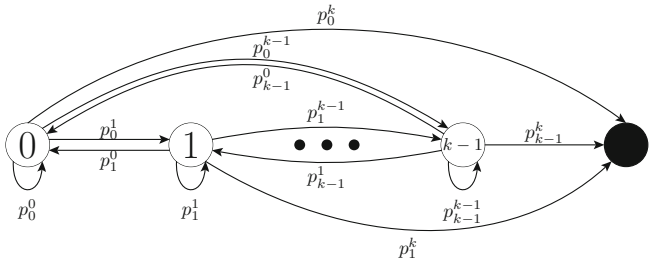

Fig. 2. Illustration of the level chain. The black circle represents the optimum that is not considered as a part of the chain, states $[0..k-1]$ represent the levels of the plateau surrounding the optimum.

There is a natural mapping from the level space to the individual space. Every vector $x = (x_0, \ldots, x_{k-1})$ can be mapped to vector $\phi(x) = (y_0, \ldots, y_{N-1})$, where $y_i = x_j / \binom{n}{k-j}$, if i-th element belongs to the j-th level. If x is a distribution over the levels, that is, $x \in [0,1]^k$ and $\|x\|_1 = 1$, then $\phi(x)$ is the distribution over the elements of the plateau which is uniform on the levels and which has the same total mass on each level as x.

This mapping has several useful properties.

1. This mapping is linear, that is, we have $\phi(\alpha x + \beta y) = \alpha\phi(x) + \beta\phi(y)$ for all $x, y \in \mathbb{R}^k$ and all $\alpha, \beta \in \mathbb{R}$. This property follows right from the definition of ϕ.

2. For all $x \in \mathbb{R}^k$ we have $\phi(xP) = \phi(x)P_{\text{ind}}$. To justify this property, let us notice two facts. First, if some mass vector y from the individual space has a uniform distribution of the mass inside each level, then after applying matrix P_{ind} to this vector, this property will remain true due to symmetry. Second, the transition of mass between levels in the level chain is the same as in the individual chain. Therefore, if we regard $\phi(xP)$ as the mass x that firstly was transferred over the level chain by matrix P and then distributed uniformly inside each level and we regard $\phi(x)P_{\text{ind}}$ as the mass x that firstly was distributed inside each level and then transferred between levels by matrix P_{ind}, then we see that it is the same vector.
3. From the previous properties we see that if x is an eigenvector of P, then $\phi(x)$ is an eigenvector of P_{ind} with the same eigenvalue. Therefore, the spectrum $\sigma(P)$ of the matrix P is a subset of the spectrum $\sigma(P_{\text{ind}})$ of matrix P_{ind}.
4. For all $x \in \mathbb{R}^k$, the Manhattan norm is invariant under ϕ, that is, $\|x\|_1 = \|\phi(x)\|_1$. This follows from the fact that all components of $\phi(x)$ that are from the same level have the same sign. Notice that an analogous property does not hold for the Euclidean norm $\|\cdot\|_2$.

4 The Spectrum of the Transition Matrix

Let p_i^k be the probability to find the optimum in one iteration if the current individual is in level i and let p_i^j for all $i, j \in [0..k-1]$ be the elements of P, that is, the transition matrix for the level chain. For all $i \in [0..k-1]$ and $j \in [0..k]$ we have

$$p_i^j = \begin{cases} \sum_{m=0}^{k-j} \binom{k-i}{j-i+m}\binom{n-k+i}{m}\binom{n}{j-i+2m}^{-1} \Pr[\alpha = j-i+2m], & \text{if } j > i, \\ \sum_{m=0}^{k-i} \binom{k-i}{m}\binom{n-k+i}{i-j+m}\binom{n}{i-j+2m}^{-1} \Pr[\alpha = i-j+2m], & \text{if } j < i, \\ 1 - \sum_{m=0, m\neq i}^{k} p_i^m, & \text{if } j = i. \end{cases}$$

The main result of this section is the following Lemma.

Lemma 1. *All the eigenvalues of matrix P are real. The largest eigenvalue of P is $\lambda_0 = 1 - o(1)$. If we have $\Pr[\alpha = 1] > c$, where c is some constant, then there exists a constant $\varepsilon > 0$ such that any other eigenvalue λ' of P satisfies $|\lambda'| < 1 - \varepsilon$.*

The fact that all the eigenvalues are real follows from the third property of ϕ and the fact that all the eigenvalues of the symmetric matrix P_{ind} are real. λ_0, that is, largest eigenvalue of P, is bounded by the minimal and the maximal row sum of P (see Perron-Frobenius Theorem [22]), which are both $1 - o(1)$. However the fact that all other eigenvalues are less than $1 - \varepsilon$ for some constant ε requires a precise analysis of the characteristic polynomial of P. The details are omitted for reasons of space.

5 Runtime Analysis

The Perron-Frobenius Theorem [22] states that for positive matrices the largest eigenvalue has a one-dimensional eigenspace. Also this theorem asserts that both left and right eigenvectors that correspond to the largest eigenvalue have all components with the same sign and they do not have any zero component. Let π^* be such a left eigenvector with positive components for P and let it be normalized in such way that $\|\pi^*\|_1 = 1$. We view π^* as distribution over the levels of the plateau and call it the *conditional stationary distribution of* P since it does not change in one iteration under the condition that the algorithm does not find the optimum. Also let $u = (u_0, \dots, u_{k-1})$ be the probability distribution in the level space that is uniform on the whole plateau, that is,

$$u_i = \binom{n}{k-i} \Big/ \sum_{j=0}^{k-1} \binom{n}{k-j} = \binom{n}{k-i} N^{-1}.$$

In the remainder, we need the following basis of the level space.

Lemma 2. *There exists a basis of the level space* $\{e^i\}_{i=0}^{k-1}$ *with the following properties.*

1. $\pi^* = e^0$;
2. e^i *is an eigenvector of* P *for all* $i \in [0..k-1]$;
3. *all* $\phi(e^i)$ *are orthogonal in the individual space.*

The first two statements are satisfied by any basis of eigenvectors of P. The third statement is not automatically satisfied if some eigenvalues have an eigenspace of dimension greater than one, however the orthogonality can be ensured via standard means from linear algebra. The details are omitted for reasons of space.

Having the basis from Lemma 2 we can prove the following relation between the vectors π^* and u.

Lemma 3. *For all* $j \in [0..k-1]$, *we have* $\pi^*_j = u_j(1 \pm O(1/\sqrt{n}))$.

The proof is again omitted for reasons of space. From Lemmas 2 and 3, we obtain our main result (with again the proof omitted for reasons of space).

Theorem 1. *The expected runtime of the* $(1+1)$ *EA using any unbiased mutation operator with constant probability to flip exactly one bit on the plateau of* $\textsc{Plateau}_k$ *function is* $N\left(\Pr[1 \le \alpha \le k]\right)^{-1}(1 + o(1))$.

6 Corollaries

We now exploit Theorem 1 to see how the choice of the mutation operator influences the runtime. Since, by Theorem 1 the expected runtime depends only on the probability to flip between 1 and k bits, this is an easy task. The proofs

of the theorems formulated in this section are omitted, since they trivially follow from Theorem 1.

We first observe that for all the unbiased operators with constant probability to flip exactly one bit, the expected optimization time is $\Theta(N)$. Hence all these mutation operators lead to asymptotically the same runtime.

The best runtime, obviously, is obtained from mutation operators which flip only between 1 and k bits. It implies that the uniformly most effective algorithm for every plateau function is Random Local Search (RLS), as for this algorithm $\Pr[1 \leq \alpha \leq k] = \Pr[\alpha = 1] = 1$.

We now analyze the runtime resulting from using standard-bit mutation as in the classic $(1+1)$ EA and from using the fast genetic algorithm, that is, the $(1+1)$ EA with a heavy-tailed mutation operator.

Recall that in standard-bit mutation, each bit is flipped independently with probability γ/n, where γ usually is a constant. Recall further that the size of the plateau is $N = \sum_{i=0}^{k-1} \binom{n}{n-k+i} = (1 \pm o(1))n^k/k!$.

Theorem 2. *Let γ be some arbitrary positive constant and $k \geq 2$. Then the $(1+1)$ EA with mutation rate γ/n optimizes* $\textsc{Plateau}_k$ *in an expected number of $N/(e^{-\gamma}\sum_{i=1}^{k} \frac{\gamma^i}{i!})(1+o(1))$ iterations. This time is asymptotically minimal for $\gamma = \sqrt[k]{k!} \approx k/e$.*

The fast genetic algorithm recently proposed in [14] is simply a (1+1) EA that uses the unbiased mutation operator such that $\Pr[\alpha = i] = 0$ for every $i > n/2$ and $i = 0$ and $\Pr[\alpha = i] = i^{-\beta}/H_{n/2,\beta}$ otherwise, where β is a parameter of the algorithm that is greater than one and $H_{n/2,\beta} := \sum_{i=1}^{n/2} i^{-\beta}$ is a generalized harmonic number. The parameter β is considered to be a constant over the problem size.

Theorem 3. *The expected runtime of the fast genetic algorithm on* $\textsc{Plateau}_k$ *is $C_{kn}N$, where C_{kn} can be bounded by constants, namely $C_{kn} \in \left[\frac{1}{\beta-1} - o(1)}{H_{k,\beta}}, \frac{\frac{1}{\beta-1}+1}{H_{k,\beta}}\right]$.*

7 Conclusion

In this paper we introduced a new method to analyze the runtime of evolutionary algorithms on plateaus. This method does not depend on the particular mutation operator used by the EA as long as there is a constant positive probability to flip a single random bit. We performed a very precise analysis on the particular class of plateau functions, but we are optimistic that similar methods can be applied for the analysis of other plateaus. For example, Lemmas 1, 2 and 3 remain true for those plateaus of the function XDIVK (that is defined as $\lfloor \textsc{OneMax}(x)/k \rfloor$ for some parameter k) that are in a constant Hamming distance from the optimum (and these are the plateaus which contribute most to the runtime). That said, the proof of Lemma 1 would need to be adapted to these plateaus different from the one of our plateau function. We are optimistic that this can be done, but leave it as an open problem for now.

The inspiration for our analysis method comes from the observation that the algorithm spends a relatively long time on the plateau. So regardless of the initial distribution on the plateau, the distribution of the individual converges to the conditional stationary distribution long before the algorithm leaves the plateau. This indicates that our method is less suitable to analyze how evolutionary algorithms leave plateaus which are easy to leave, but such plateaus usually present not bigger problems in optimization.

On the positive side, our analysis method can also be used to give runtime estimates for functions having less symmetric plateaus than our PLATEAU functions. For example, assume that $f : \{0,1\}^n \to \mathbb{R}$ is a function that agrees with PLATEAU_k on all search points x with $\text{PLATEAU}(x) = \text{ONEMAX}(x)$, but has only the restriction $n - k \leq f(x) \leq n$ for the other search points. Such functions can have plateaus of arbitrary shape inside the plateau of second-best fitness of PLATEAU_k. It is easy to see that the runtime T of the $(1+1)$ EA with arbitrary unbiased mutation operator satisfies the same asymptotic upper bound $N/\Pr[\alpha \in [1..k]](1 + o(1))$ that we have proven for the PLATEAU_k function.

Overall, we are optimistic that our main analysis method, switching between the level chain and the individual chain, which might be the first attempt to devise a general analysis method for EAs on plateaus, finds further applications.

References

1. Antipov, D., Doerr, B.: Precise runtime analysis for plateaus (2018). http://arxiv.org/abs/1806.01331
2. Böttcher, S., Doerr, B., Neumann, F.: Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In: Schaefer, R., Cotta, C., Kołodziej, J., Rudolph, G. (eds.) PPSN 2010. LNCS, vol. 6238, pp. 1–10. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15844-5_1
3. Brockhoff, D., Friedrich, T., Hebbinghaus, N., Klein, C., Neumann, F., Zitzler, E.: On the effects of adding objectives to plateau functions. IEEE Trans. Evol. Comput. **13**(3), 591–603 (2009)
4. Buzdalov, M., Doerr, B., Kever, M.: The unrestricted black-box complexity of jump functions. Evol. Comput. **24**(4), 719–744 (2016)
5. Dang, D.-C., et al.: Emergence of diversity and its benefits for crossover in genetic algorithms. In: Handl, J., Hart, E., Lewis, P.R., López-Ibáñez, M., Ochoa, G., Paechter, B. (eds.) PPSN 2016. LNCS, vol. 9921, pp. 890–900. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45823-6_83
6. Dang, D.C., et al.: Escaping local optima with diversity mechanisms and crossover. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2016, pp. 645–652. ACM (2016)
7. Dang, D., Lehre, P.K.: Runtime analysis of non-elitist populations: from classical optimisation to partial information. Algorithmica **75**(3), 428–461 (2016)
8. Doerr, B., Doerr, C., Kötzing, T.: Unbiased black-box complexities of jump functions. Evol. Comput. **23**(4), 641–670 (2015)
9. Doerr, B., Fouz, M., Witt, C.: Sharp bounds by probability-generating functions and variable drift. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2011, pp. 2083–2090. ACM (2011)

10. Doerr, B., Hebbinghaus, N., Neumann, F.: Speeding up evolutionary algorithms through asymmetric mutation operators. Evol. Comput. **15**, 401–410 (2007)
11. Doerr, B., Jansen, T., Sudholt, D., Winzen, C., Zarges, C.: Mutation rate matters even when optimizing monotonic functions. Evol. Comput. **21**(1), 1–27 (2013)
12. Doerr, B., Jansen, T., Klein, C.: Comparing global and local mutations on bit strings. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2008, pp. 929–936. ACM (2008)
13. Doerr, B., Johannsen, D., Winzen, C.: Multiplicative drift analysis. Algorithmica **64**(4), 673–697 (2012)
14. Doerr, B., Le, H.P., Makhmara, R., Nguyen, T.D.: Fast genetic algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, pp. 777–784. ACM (2017). http://arxiv.org/abs/1703.03334
15. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. Theor. Comput. Sci. **276**(1–2), 51–81 (2002)
16. Garnier, J., Kallel, L., Schoenauer, M.: Rigorous hitting times for binary mutations. Evol. Comput. **7**(2), 173–203 (1999)
17. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. Artif. Intell. **127**, 51–81 (2001)
18. Jansen, T., Wegener, I.: The analysis of evolutionary algorithms–a proof that crossover really can help. Algorithmica **34**(1), 47–66 (2002)
19. Lehre, P.K., Witt, C.: Black-box search by unbiased variation. Algorithmica **64**(4), 623–642 (2012)
20. Lehre, P.K., Witt, C.: Concentrated hitting times of randomized search heuristics with variable drift. In: Ahn, H.-K., Shin, C.-S. (eds.) ISAAC 2014. LNCS, vol. 8889, pp. 686–697. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13075-0_54
21. Lissovoi, A., Oliveto, P.S., Warwicker, J.A.: On the runtime analysis of generalised selection hyper-heuristics for pseudo-boolean optimisation. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, pp. 849–856. ACM (2017)
22. Meyer, C.D. (ed.): Matrix Analysis and Applied Linear Algebra. Society for Industrial and Applied Mathematics, Philadelphia (2000)
23. Mironovich, V., Buzdalov, M.: Hard test generation for maximum flow algorithms with the fast crossover-based evolutionary algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2015, pp. 1229–1232. ACM (2015)
24. Sudholt, D.: A new method for lower bounds on the running time of evolutionary algorithms. IEEE Trans. Evol. Comput. **17**, 418–435 (2013)
25. Wegener, I.: Theoretical aspects of evolutionary algorithms. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 64–78. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-48224-5_6
26. Witt, C.: Tight bounds on the optimization time of a randomized search heuristic on linear functions. Comb. Probab. Comput. **22**(2), 294–318 (2013)