Self-Adaptation in Nonelitist Evolutionary Algorithms on Discrete Problems With Unknown Structure

Brendan Case and Per Kristian Lehre

Abstract—A key challenge to make effective use of evolutionary algorithms (EAs) is to choose appropriate settings for their parameters. However, the appropriate parameter setting generally depends on the structure of the optimization problem, which is often unknown to the user. Nondeterministic parameter control mechanisms adjust parameters using information obtained from the evolutionary process. Self-adaptation—where parameter settings are encoded in the chromosomes of individuals and evolve through mutation and crossover—is a popular parameter control mechanism in evolutionary strategies. However, there is little theoretical evidence that self-adaptation is effective, and self-adaptation has largely been ignored by the discrete evolutionary computation community. Here, we show through a theoretical runtime analysis that a nonelitist, discrete EA which self-adapts its mutation rate not only outperforms EAs which use static mutation rates on LEADINGONES4, but also improves asymptotically on an EA using a state-of-the-art control mechanism. The structure of this problem depends on a parameter $k$, which is \textit{a priori} unknown to the algorithm, and which is needed to appropriately set a fixed mutation rate. The self-adaptive EA achieves the same asymptotic runtime as if this parameter was known to the algorithm beforehand, which is an asymptotic speedup for this problem compared to all other EAs previously studied. An experimental study of how the mutation-rates evolve shows that they respond adequately to a diverse range of problem structures. These results suggest that self-adaptation should be adopted more broadly as a parameter control mechanism in discrete, nonelitist EAs.

Index Terms—Evolutionary algorithm (EA), level-based analysis, runtime analysis, self-adaptation.

I. INTRODUCTION

Evolutionary algorithms (EAs) have long been heralded for their easy application to a vast array of real-world problems. In their earlier years of study, two of the advantages which were often given were their robustness to different parameter settings, such as mutation rate and population size, and their effectiveness in domains where little is known about the problem structure [10]. However, progress in the empirical and theoretical study of EAs has shown many exceptions to these statements. It is now known that even small changes to the basic parameters of an EA can drastically increase the runtime on some problems [20], [31], and more recently [34], and that hiding some aspects of the problem structure from an EA can decrease performance [5], [15], [16], [26].

A popular solution to overcoming these shortcomings is parameter tuning, where the parameters are adjusted between runs of the algorithm. Since the parameters remain fixed throughout the entire run of the optimization process under this scheme, this parameter scheme is said to be static [25]. While the majority of theoretical works have historically investigated static parameter settings, a weakness of parameter tuning is that effective parameter settings may depend on the current state of the search process.

An alternative approach is a dynamic parameter scheme, which has long been known to be advantageous compared to static parameter choices in certain settings [24], [37], reviewed in [13]. In contrast to parameter tuning, adjusting parameters in this way is referred to as parameter control [25]. Dynamic parameter control changes parameters of the EA during its execution, usually depending on the EA's state in the optimization process or on time. While this can lead to provably better performance, many theoretically studied algorithms are fitness-dependent, meaning they set parameters according to the given optimization function. While important for understanding the limits of parameter control, such control schemes are often ill-suited for more general optimization tasks or on problems where finding an effective fitness-dependent parameter setting is impractical [13]. Thus, practitioners may find it challenging to transfer theoretical results about fitness-dependent algorithms to an applied setting.

A more flexible way to dynamically adjust parameters is to use feedback from the algorithm’s recent performance. This self-correcting, or adaptive approach to parameter control has been present in evolutionary strategies since their beginning with the 1/5th rule; however, results concerning this kind of adjustment have only recently been seen in the theoretical literature for discrete EAs [11], [17], [28]. The advantages of adjusting parameters on the fly in this way include a reduction in design decisions compared to fitness-dependent algorithms, and the ability for the same adaptive scheme to work well for a wider range of optimization problems [27].

The adaptive parameter control scheme we consider employs self-adaptation of the EA’s mutation rates, where
mutation rates are encoded into the genome of individual solutions. With self-adaptation, the mutation rate itself is mutated when an individual undergoes mutation. As far as we know, within the theory of discrete EAs there are only two existing studies of self-adaptation. In [8], a self-adapting population using two mutation rates is shown to have a runtime (expected number of fitness evaluations) of $O(nλ/\log λ + n \log n)$ on a simple peak function, while the same algorithm using any fixed mutation rate took $e^{Ω(n)}$ evaluations with overwhelming probability. Recently, Doerr et al. [21] gave an example of a $(1,λ)$ EA using self-adaptation of mutation rates with expected runtime $O(nλ/\log λ + n \log n)$ on ONE MAX when $λ \geq (ln n)^{1+ε}$, an asymptotic speedup from the classic $(1+λ)$ EA. However, the former work optimistically assumes one of the two available mutation rates are appropriate for the given setting, so that any individual can easily switch to an ideal mutation rate in a single step, and the latter only keeps the mutation rate of the best individual after each generation, which makes tracking the trajectory of mutation rates less difficult than if there were multiple parents with different mutation rates. Therefore, while both these algorithms were effective, these two results offer only a preliminary theoretical understanding of the full range of self-adaptive mechanisms. Further, the use of limited mutation rates or a single parent is unrealistic in many real-world settings (i.e., where cross-over is frequently used). Thus, it remains an open question whether a population-based EA can effectively adapt mutation rate without these assumptions.

We answer this question in the affirmative, introducing an extension of the $(μ,λ)$ EA which uses self-adaptation of mutation rates over a continuous interval (Algorithm 1). In each generation, a new population of $λ$ individuals is created by selecting among the $μ$ individuals with highest fitness, ties broken according to higher mutation rate. Each selected individual then multiplies its current mutation rate by a factor of $b$ in $(0,1)$, effectively increasing or decreasing its mutation rate, before undergoing bitwise mutation. To evaluate the capability of the EA to adapt its mutation rate, we choose a problem where selecting the right mutation rate is critical, and where the correct setting can be anywhere between a small constant to $n/2$, where $n$ is the problem instance size. We show that when optimizing LEADING ONES$_k$ the self-adaptive algorithm has an expected runtime of $O(k^2)$ so long as $λ = O(\log n)$ and $k \geq (\log n)^2$, which is the same runtime as if $k$ were known. As discussed in more detail in Section I-B, this is a significant speedup compared to an EA using a static choice of mutation rate, which can only achieve $\Theta(nk)$ on LEADING ONES$_k$. This is also an asymptotic speedup from the best-known runtime shown in [16], and indeed is asymptotically optimal among all unary unbiased black-box algorithms [3].

A. Theory of Adaptive Parameter Control

In the following summary of recent results in the theory of parameter control in EAs, we use the language of [25] to distinguish between types of parameter control. A parameter control scheme is called deterministic if it uses time or other predefined, fitness-independent factors to adjust parameters, and adaptive if it changes parameters according to feedback from the optimization process. As further distinguished in [11], a notable distinction among adaptive algorithms is whether or not they are fitness-dependent, i.e., whether they directly use a particular fitness function when choosing parameter settings. Adaptive algorithms which are fitness-independent are either self-adjusting, where a global parameter is modified according to a simple rule, or self-adaptive, where the parameter is encoded into the genome of an individual and modified through mutation.

For a comprehensive survey of the theory of parameter control in discrete settings, we refer the reader to Doerr and Doerr’s recent review [13]. We now highlight some themes from the theory of parameter control relevant to this article.

Comparison of Fitness-Independent Mechanisms to Fitness-Dependent Ones: Often, the best parameter settings have a fitness-dependent expression which depends on the precise fitness value of the search point at that time. While these settings are typically problem-specific, there is an increasing number of self-adjusting algorithms which are nearly as efficient, despite not being tailored to a particular fitness landscape. A common strategy is to first analyze the fitness-dependent case, followed by a self-adjusting scheme which attempts to approximate the behavior of the fitness-dependent one. For example, in [14] a novel $(1+(λ,λ))$ GA is shown to need only $Θ(n)$ fitness evaluations on ONE MAX when using a fitness-dependent offspring size of $λ = Θ(\sqrt{n/(n−OM(x)))}$. This result is then extended using an adaptive mechanism based on the $1/5$th rule, where a key element to proving the algorithm’s effectiveness is in demonstrating the adaptive GA’s offspring size $λ$ is quickly attracted to within a constant factor of the fitness-dependent value [12]. A similar pattern of discovery occurred for the $(1+λ)$ EA on ONE MAX. Badkobeh et al. [3] first showed that a fitness-dependent mutation rate led to an expected runtime of $O([nλ/\log λ]+n \log n)$, which is asymptotically tight among all $λ$-parallel mutation-based unbiased black-box algorithms and a speedup from the static-mutation case. This was followed by [19], where a self-adjusting $(1+λ)$ EA is shown to have the same asymptotic runtime when $λ = n^{O(1)}$, and the aforementioned result for the self-adaptive $(1,λ)$ EA [21]. Again, it is shown the algorithm is able to quickly find mutation rates close to the fitness-dependent values. The mutation rates are shown to stay within this optimal range using occupation bounds.

For LEADING ONES, it was first demonstrated by Böttcher et al. [4] that the bitflip probability $1/(LO(x)+1)$ led to an improved runtime of roughly $0.68n^2$ for the $(1+1)$ EA on LEADING ONES. Since then, experimental results for the self-adjusting $(1+1)_a$ EA suggest the algorithm is able to closely approximate this value [23, Fig. 3], and hyper-parameters for the algorithm have been found to yield the asymptotically optimal bound $0.68n^2(1+o(1))$ [17].

Interplay Between the Mutation Rate and Selective Pressure in Nonelitist EAs: While adaptive parameter control has been studied considerably less in nonelitist EAs, the critical balance between a nonelitist EA’s mutation rate and selective pressure
(how much the algorithm tends to select the top individuals in the population) takes on new importance when using self-adaptation of mutation rates. In [33], the linear-ranking EA is shown to optimize a class of functions in a subexponential number of fitness evaluations only when the selective pressure is in a narrow interval, proportional to the mutation rate. A more general result for nonelitist EAs using mutation rate $\chi/n$ is found in [29, Corollary 1], where for a variety of selective mechanisms the lower bound $\chi > \ln(\alpha_0) + \delta$ is given, where $\alpha_0$ is the reproductive rate and $\delta \in (0, 1)$ is a constant (the reproductive rate is one measure of the selective pressure on an EA, see Definition 2). If $\chi$ exceeds this bound, with overwhelming probability any algorithm using this rate will have exponential runtime on any function with a polynomial number of global optima. This negative result is extended in [8, Th. 2] to include nonelitist EAs which choose from a range of $m$ different mutation rates by selecting mutation rate $\chi_i/n$ with probability $q_i$. Roughly, if $\sum_{i=1}^{m} q_i e^{-x_i} \leq (1-\delta)/\alpha_0$, the algorithm will be ineffective.

B. Optimization Against an Adversary

We will analyze the performance of our algorithm on the \textsc{LeadingOnes}_k(x) function, which counts the number of leading 1-bits, but only through the first $k$ bits and ignores the rest of the bitstring.

\begin{definition}
For $x \in \{0, 1\}^n$, and $1 \leq k \leq n$
\end{definition}

\begin{definition}
\textsc{LeadingOnes}_k(x) := \sum_{i=1}^{k} \prod_{j=1}^{i} x_j.
\end{definition}

The setting in which we consider this function is referred to as optimization against an adversary. This can be viewed as an extension of the traditional black-box optimization setting, in which the algorithm does not have access to the problem data or structure and must learn only through evaluating candidate solutions. Framing the study of EAs within the context of black-box optimization, and its corresponding black-box complexity theory, is of growing interest to the theoretical community [22]. Optimization against an adversary adds the additional constraint that the value $k$ is also unavailable to the algorithm. That is, prior to each run of the optimization algorithm, an adversary chooses an integer $k \leq n$ and the algorithm must optimize the resulting problem $f_k$. Effectively, the adversary is able to choose some $f_k$ from a class of functions parameterized by $k$, and the algorithm could have to solve any problem from this class. Note that the adversary is not able to actually permute any bits during optimization, they only influence the optimization task through their choice of $k$. A similar problem was first analyzed by Cathabard et al. [5], along with an analogous \textsc{OneMax}_k function, though here $k$ was sampled from a known distribution. The setting where $k$ corresponds to an unknown initial number of bits which impact fitness has become known as the initial segment uncertainty model. The closely related hidden subset problem, which is analogous to the initial segment model except the $k$ meaningful bits can be anywhere in the bitstring, has also been studied for \textsc{LeadingOnes}_k and \textsc{OneMax}_k [15], [16], [26].

Since our algorithm always flips all bits with equal probability during mutation, our results immediately extend to this class of problems. Optimization against an adversary further generalizes this terminology to contain any problem in which an adversary can control the hidden problem structure through their choice of $k$. For example, it includes the \textsc{Substring}_k function introduced in Section IV.

The addition of an adversary can be difficult for EAs with static mutation rate due to the following phenomenon: consider a $(1 + 1)$ EA using constant mutation probability $p$, and suppose we are attempting to optimize \textsc{LeadingOnes}_k against an adversary. If $k$ is far less than $n$, the traditional choice of $p = 1/n$ will be far too conservative, and the expected number of function evaluations until the optimum is found will be $\Theta(nk)$. On the other hand, choosing a higher value of $p$ such that $p = \omega(1/n)$ will not work if the adversary chooses a value of $k$ quite close to $n$, since in this case the EA will flip the leading 1-bits with too high probability and have exponential runtime. However, several extensions of the $(1 + 1)$ EA have been proposed which are more effective for optimization in this uncertain environment. Doerr et al. [16] considered two different variants of the $(1 + 1)$ EA, one which assigns different flip probabilities to each bit, and one which samples a new bitflip probability from a distribution $Q$ in each generation, both of which they show to have an expected runtime of $O(k^2 (\log k)^{1+\epsilon})$ on \textsc{LeadingOnes}_k. They also show the log$^{1+\epsilon} k$ term can be further reduced by more carefully choosing the positional bitflip probabilities or the distribution $Q$; however, in a follow-up work, it is shown that the upper bound for both of these algorithms is nearly tight, that is, the expected runtime is $\omega(k^2 \log k)$ [15]. In [26], a different sort of self-adjusting $(1 + 1)$ EA is introduced for the hidden subset problem on the class of linear functions. Rather than adjusting the mutation rate in each generation during the actual search process, the algorithm instead spends $O(k)$ generations approximating the hidden value $k$, and then $O(k \log k)$ generations actually optimizing $f_k$ now that $k$ is approximately known. This algorithm not only improves the bound from $O((k \log k)^{2+\epsilon})$ in [16] to $\Theta(k \log k)$ for \textsc{OneMax}_k under the hidden subset model but also the implicit constants of $(1 \pm o(1)) n \ln n$ are found as well, matching the performance of a $(1 + 1)$ EA which knows $k$ in advance. However, it remained to be demonstrated whether an EA could similarly solve the \textsc{LeadingOnes}_k problem at no extra cost when $k$ was unknown.

C. Structure of This Article

Section II introduces notation, a formal description of the self-adaptive algorithm (Algorithm 1), and the analytical tools we used. Section III provides our main result, that Algorithm 1 optimizes \textsc{LeadingOnes}_k against an adversary in expected time $O(k^2)$. Section IV is an experimental study on theoretical benchmark functions, first illustrating the evolution of the mutation rate throughout the optimization process, then comparing the average runtime during optimization against an adversary of the algorithm to some classic EAs and to the adaptive $(1 + 1)_a$ EA [23]. Section V concludes this article.
Algorithm 1 (μ, λ) Self-Adaptive EA

Require: Fitness function \( f : \mathcal{X} \to \mathbb{R} \).

Require: Population sizes \( \mu, \lambda \in \mathbb{N} \), where \( 1 \leq \mu \leq \lambda \).

Require: Adaptation parameters \( A > 1 \), and \( b, p_{\text{inc}} \in (0, 1) \).

Require: Initial population \( P_0 \in \mathcal{X}^\lambda \).

1: for \( t \in 0, 1, 2, \ldots \) until termination condition met do
2: Sort \( P_t \), st. \( P_t(1) \geq \cdots \geq P_t(\lambda) \), according to (2).
3: for \( i \in 1, \ldots, \lambda \) do
4: Set \( (x, \chi/n) := P_t(i), I_t(i) \sim \text{Unif}(\{\mu\}) \).
5: Set \( \chi' := \min\{A\chi/n, 2\} \) with probability \( p_{\text{inc}} \)
6: Create \( x' \) by independently flipping each bit of \( x \) with probability \( 1/n \).
7: Set \( P_{t+1}(i) := (x', \chi'/n) \).
8: end for
9: end for

II. PRELIMINARIES

A. General Notation

For any \( n \in \mathbb{N} \), let \( [n] := \{1, \ldots, n\} \) and \( [0..n] := \{0\} \cup [n] \).

The natural logarithm is denoted by \( \ln(\cdot) \), and the logarithm base \( 2 \) by \( \log(\cdot) \).

The Iverson bracket is denoted by \( [\cdot] \), which is equal to \( 1 \) if the statement in the brackets is true, and \( 0 \) otherwise.

The search space throughout this article is \( \mathcal{X} := [0, 1]^n \), and we refer to \( x = (x_1, \ldots, x_n) \) in \( \mathcal{X} \) as a bitstring of length \( n \).

Since we are interested in searching the space of mutation rates along with the set of bitstrings, it will be convenient to define an extended search space of

\[ \mathcal{Y} := \mathcal{X} \times [\varepsilon, 1/2]. \]

The parameter \( \varepsilon = c/n \), where \( c < 1 \) is a small constant with respect to \( n \), is necessary only for technical reasons in our analysis. The Hamming distance between two bitstrings \( x, x' \) is denoted by \( H(x, x') \).

All asymptotic notation throughout this article is with respect to \( n \), the size of the problem space. The runtime of a search process is defined as the number of fitness evaluations before an optimal search point is found, denoted by \( \tau \).

B. Self-Adaptive (μ, λ) EA

We consider a nonelitist EA using self-adaptation of mutation rates, outlined in Algorithm 1. We refer to a population as a vector \( P \in \mathcal{Y}^\lambda \), where \( \lambda \in \mathbb{N} \) is the population size, and where the \( i \)th element \( P(i) \) is called the \( i \)th individual. For an individual \( x, \chi/n \in \mathcal{Y} \), we refer to \( \chi/n \in [\varepsilon, 1/2] \) as the mutation rate, and \( \chi \) as the mutation parameter. In each generation \( t \in \mathbb{N}_0 \), Algorithm 1 creates the next population \( P_{t+1} \) by independently creating \( \lambda \) new individuals according to a sequence of operations selection, adaptation, and mutation.

1) Selection: We consider a variant of the standard (μ, λ) selection scheme, where the \( \mu \leq \lambda \) best individuals are chosen according to fitness, with ties broken according to the individual with higher mutation rate. More precisely, the population is first sorted according to the ordering

\[ (x, \chi) \geq (x', \chi') \Leftrightarrow f(x) > f(x') \lor (f(x) = f(x') \land \chi \geq \chi') \]

(2)

where ties of \( f(x) = f(x') \) and \( \chi = \chi' \) are broken arbitrarily. Then, each parent is chosen uniformly from the \( \mu \) top individuals \( P_t(1), \ldots, P_t(\mu) \).

We quantify the selective pressure of the selection mechanism using the reproductive rate.

Definition 2 [29]: The reproductive rate of Algorithm 1 is

\[ a_0 := \max_{1 \leq i \leq 2} E[R_t(i)], \]

where \( R_t(i) := \sum_{j=1}^n [I_t(j) = i] \).

That is, \( a_0 \) is the expected number of times per generation an individual with the highest selection probability is chosen in step 4 of Algorithm 1. A well-known fact is that the reproductive rate of the standard (μ, λ) EA is \( \lambda/\mu \) ([30, Lemma 7]). This is also the case for Algorithm 1.

2) Adaptation: Each chromosome \((x, \chi/n)\) carries both a search point \( x \) and a mutation parameter \( \chi \). In order for the population to explore different mutation rates, it must be possible for the offspring to inherit a “mutated” mutation parameter \( \chi' \) different from its parent. For the purpose of the theoretical analysis, we are looking for the simplest possible update mechanism, which is still capable of adapting the mutation rates in the population.

We will prove that the following simple multiplicative update scheme suffices: given a parent with mutation parameter \( \chi \), the offspring inherits an increased mutation parameter \( A\chi \) with probability \( p_{\text{inc}} \) and a reduced mutation parameter \( b\chi \) with probability \( 1-p_{\text{inc}} \), where \( A \) and \( b \) are two parameters satisfying \( 0 < b < 1 < A \). We choose the parameter names \( A \) and \( b \) for consistency with the adaptive \((1+1)\_a\) EA already introduced in [23], which similarly changes the mutation rate in this step-wise fashion. However, unlike Algorithm 1, the \((1+1)\_a\) EA changes the mutation rate based on whether the offspring is fitter than the parent.

Our goal is that the EA adapts the mutation parameter \( \chi \) to the problem at hand, so that it is no longer necessary to set the parameter \( \chi \) manually. It may seem counter-productive to replace one mutation parameter by introducing three new adaptation mechanism parameters \( A, b, \) and \( p_{\text{inc}} \). However, we will show that while the mutation parameter \( \chi \) must be tuned for each problem, the same fixed setting of the parameters \( A, b, \) and \( p_{\text{inc}} \) is effective across many problems. We conjecture that the self-adaptive EA will be effective with other adaptation mechanisms. For example, rather than multiplying by the constants \( A \) and \( b \), we could multiply the mutation parameter by a factor with log-normal distribution as was originally done in [2]. We suspect that many adjustment mechanisms which favor taking small steps from the current mutation rate could be analyzed similarly to the analysis presented in this article.

3) Mutation: The mutation step is when a new candidate solution is actually created by our algorithm. We consider standard bit-wise mutation, where a parent \( x \in \{0, 1\}^n \) with mutation rate \( \chi/n \) produces an offspring \( x' \in \{0, 1\}^n \) by flipping each bit of \( x \) independently with probability \( \chi/n \). We adopt the notation from [29], and consider the offspring \( x' \) a...
random variable $x' \sim p_{\text{mut}}(x, \chi)$, with distribution

$$\Pr(x' = p_{\text{mut}}(x, \chi)) := \left(\frac{x}{n}\right)^{H(x, x')} \left(1 - \frac{x}{n}\right)^{n-H(x, x')}.$$  

C. Level-Based Analysis

We analyze the runtime of Algorithm 1 using level-based analysis. Introduced by Corus et al. [7], the level-based theorem is a general tool for deriving upper bounds on the expected runtime for nonlocal population-based EAs, and has been applied to a wide range of algorithms, including to GAs [7], and EDAs [9].

The theorem can be applied to any population-based stochastic process $(P_t)_{t \in \mathbb{N}}$, where individuals in $P_{t+1}$ are sampled independently from a distribution $D(P_t)$, where $D$ maps populations to distributions over the search space. In the case of our algorithm, $D$ is the composition of selection, adaptation, and mutation. The theorem also assumes a partition $(A_1, \ldots, A_m)$ of the finite search space into $m$ subsets, also called levels. Usually, this partition is over the function domain $\mathcal{X}$, but since we are concerned with tracking the evolution of the population over the 2-D space of bitstrings and mutation rates, we will rather work with subsets of $\mathcal{Y}$.

Given any subset $A \subseteq \mathcal{Y}$, we slightly abuse notation and let $|P \cap A| := |\{i \in [\mathcal{Y}] \mid P(i) \in A\}|$ denote the number of individuals in a population $P \subseteq \mathcal{Y}$ that belong to the subset $A$. Given a partition of the search space $\mathcal{X}$ into levels $(A_1, \ldots, A_m)$, we define for notational convenience $A_{\geq j} := \bigcup_{j=0}^m A_j$ and $A_{j+1} := \bigcup_{j=1}^m A_j$.

**Theorem 1 [7]:** Given a partition $(A_1, \ldots, A_m)$ of $\mathcal{X}$, define $T := \min\{t \mid P_t \cap A_m \neq \emptyset\}$, where for all $t \in \mathbb{N}$, $P_t \subseteq \mathcal{X}$ is the population of [7, Algorithm 1] in generation $t$. If there exist $z_1, \ldots, z_m$, $\delta \in (0, 1)$, and $\gamma_0 \in (0, 1)$ such that for any population $P \subseteq \mathcal{X}$.

(G1): For each $j \in [m-1]$, if $|P \cap A_{j+1}| \geq \gamma_0 \lambda$, then

$$\Pr_{y \sim D(P)}(y \in A_{j+1}) \geq \gamma_j.$$  

(G2): For each $j \in [m-2]$, and all $\gamma \in (0, \gamma_0]$ if $|P \cap A_{j+1}| \geq \gamma_0 \lambda$ and $|P \cap A_{j+1}| \geq \gamma \lambda$, then

$$\Pr_{y \sim D(P)}(y \in A_{j+1}) \geq (1 + \delta)^\gamma.$$  

(G3): The population size $\lambda \in \mathbb{N}$ satisfies

$$\lambda \geq \left(\frac{4}{\gamma_0 \delta} \right) \ln \left(\frac{128 m}{z \delta^2 \lambda}\right),$$

then $E[T] \leq (8/\delta^2) \sum_{j=1}^{m-1} (\lambda \ln [(6 \delta \lambda)/(4 + z_j \delta \lambda)] + [1/j])$.

III. Runtime Analysis of LeadingOnes

We now introduce our main result, which is an upper bound on the optimization time of Algorithm 1 on the LeadingOnes problem. Note that for population size $\lambda = c \ln(n)$ and problem parameter $k \geq (\log(n))^2$, the bound in the theorem simplifies to $O(k^2)$ which is asymptotically optimal among all unary unbiased black-box algorithms, regardless of whether the parameter $k$ is fixed [32].

**Theorem 2:** Algorithm 1 with $(\lambda/\mu) = a_0 \geq 4$, constant parameters $A, b, p_{\text{inc}} \in \mathbb{R}$ satisfying $A > 1$, $(1 + \delta)/\alpha_0 < p_{\text{inc}} < 2/5$, and $0 < b < 1/(1 + \sqrt{1/(\alpha_0 (1 - p_{\text{inc}}))})$ for some $\delta \in (0, 1)$, parent population size $\mu = \Omega(\log(n))$, and $\lambda \geq c \ln(n)$ for a large enough constant $c$, for any $k \in \mathbb{N}$, has expected runtime $O(k \log(\alpha_0) + k^2)$ on LeadingOnes.

The proof of Theorem 2 is structured as follows. In Section III-A, in order to apply the level-based analysis and track the population’s progress over a 2-D landscape, we begin by defining a partition of $\mathcal{Y}$. Since our partition is more involved than those usually applied to the level-based theorem, we also verify it is truly a partition. In Section III-C, we identify a region of the search space where individuals have a mutation rate which is too high with respect to their fitness, then show that with overwhelming probability, individuals in this region will not dominate the population. In Section III-B, the main technical section, we calculate the probabilities of a parent individual producing an offspring in a level at least as good as its own, and of producing an offspring in a strictly better level. Finally in Section III-D we put everything together and apply Theorem 1 to our partition to obtain an upper bound on the expected runtime.

A. Partitioning the Search Space into Levels

We now partition the 2-D search space $\mathcal{Y} = \mathcal{X} \times [\epsilon, 1/2]$ into “levels,” which is required to apply Theorem 1. The proof of Theorem 1 uses the levels to measure the progress of the population through the search space. The progress of Algorithm 1 depends both on the fitness of its individuals, as well as on their mutation rates. We start by defining a partition on the search space $\mathcal{X}$, into $k + 1$ canonical fitness levels, for $j \in [0..k]$

$$A_j := \{x \in \mathcal{X} \mid \text{LEADINGONES}_k(x) = j\}.$$

These fitness levels will be used later to define a partition on the extended search space $\mathcal{Y}$.

The probability of a “fitness upgrade,” i.e., that a parent produces an offspring which is fitter than itself depends on the mutation rate of the parent. If the fitness of the parent is $j$, but its mutation rate is significantly lower than $1/j$, then the algorithm will lose too much time waiting for a fitness upgrade, and should rather produce offspring with increased mutation rates. Conversely, if the mutation rate is significantly higher than $1/j$, then the mutation operator is too likely to destroy the valuable bits of the parent.

To make this intuition precise, we will define for each fitness level $j \in [0..k-1]$, two threshold values $\theta_1(j)$ and $\theta_2(j)$. These values will be defined such that when the mutation rate satisfies $\mu / \lambda \in [\epsilon, \theta_1(j)]$, then the mutation rate is too low for a speedy fitness upgrade, when the mutation rate satisfies $\mu / \lambda \in (\theta_1(j), \theta_2(j))$, then the mutation rate is ideal for a fitness upgrade, and when the mutation rate satisfies $\mu / \lambda \in (\theta_2(j), 1/2]$, then the mutation rate is too high. To not distract from our introduction of the levels, we postpone the detailed derivation of the expressions of the threshold values to Section III-B, and simply assert that they satisfy the following conditions for all $j \in [0..k-1]$.

1) $\epsilon < \theta_1(j) < \min(1/2, \theta_2(j))$.
2) $\theta_1(j) > \theta_1(j+1)$.
3) $\theta_2(j) > \theta_2(j+1)$.
Condition (1) states that \([\theta_1(j), \theta_2(j)]\) forms an interval which always overlaps with the range of mutation rates reachable by Algorithm 1, while (2) and (3) state that both \(\theta_1\) and \(\theta_2\) are monotonically decreasing functions.

To reflect the progress of the population in terms of increasing the mutation rate toward the “ideal” interval \([\theta_1(j), \theta_2(j)]\) within a fitness level \(j\), we partition the extended search space \(\mathcal{Y}\) into sublevels \(A_{j}(\ell)\). The lowest sublevel \(A_{j}(0)\) corresponds to individuals with fitness \(j\) and mutation rate in the interval from \(\varepsilon\) to \(A\varepsilon\). If the mutation rate is increased by a factor of \(A\) from this level, one reaches the next sublevel \(A_{j}(1)\). In general, sublevel \(A_{j}(\ell)\) corresponds to individuals with fitness \(j\) and mutation rate in the interval from \(A^{\ell-1} \varepsilon\) to \(A^{\ell} \varepsilon\), etc. After the mutation rate has been increased a certain number of times, which we call the “depth” of fitness level \(j\), one reaches the ideal interval \([\theta_1(j), \theta_2(j)]\).

**Definition 3:** For each \(j \in [k-1]\), the depth of level \(j\) is the unique positive integer

\[
d_j := \min \left\{ \ell \in \mathbb{N} \mid \varepsilon A^\ell \geq \theta_1(j) \right\}
\]

where \(A\) is the step-size parameter from Algorithm 1.

Our next step in introducing the levels which build our partition of \(\mathcal{Y}\) is to distinguish between two conceptual types of levels, namely, between low levels and edge levels. The low levels represent regions of \(\mathcal{Y}\) where individuals have mutation rate below \(\theta_1(j)\), i.e., can still raise mutation rate while maintaining fitness with good probability. For each fitness value \(j \in [0..k-1]\), there are \(d_j - 1\) low levels.

Edge levels form a region of search points where the mutation rate is neither too low nor too high with respect to \(j\), i.e., in the ideal interval from \(\theta_1(j)\) to \(\theta_2(j)\). It is these search points which are best equipped for upgrading from fitness level \(j\) to a strictly better level. At the same time, increasing mutation further would put these individuals in danger of ruining their fitness. To progress from an edge level, an individual must strictly increase its fitness.

The final technicality to discuss before defining our partition is where to place an individual with fitness \(j\) and mutation rate \(\chi/n > \theta_2(j)\). We avoid placing such individuals into any of the low or edge levels corresponding to fitness level \(j\). However, due to the conditions (1) through (3) we imposed on \(\theta_2\), there exists some lower fitness value \(j' < j\) such that \(\theta_1(j') \leq \chi/n \leq \theta_2(j')\). This means that \(j'\) is the largest number of bits the individual will be able to maintain with good enough probability. We will, therefore, add such an individual to a level corresponding to fitness level \(j'\).

We now define the “low levels” and the “edge levels” on the extended search space \(\mathcal{Y} = X \times [\varepsilon, 1/2]\).

**Definition 4:** For \(j \in [0..k-1]\) and \(\ell \in [d_j-1]\), we define the low levels as

\[
A_{j}(\ell) := A_j \times \left[ A^{\ell-1} \varepsilon, \min( A^{\ell} \varepsilon, \theta_1(j)) \right]
\]

and for \(j \in [0..k-1]\), we define the edge levels as

\[
A_{j}(d_j) := A_j \times \left[ \theta_1(j), \min( \frac{1}{2}, \theta_2(j)) \right]
\]

\[
\cup A_{j} > j \times \left[ \min( \frac{1}{2}, \theta_2(j+1)), \min( \frac{1}{2}, \theta_2(j)) \right].
\]

Additionally, since all search points with LO\(_k(x) \geq k\) are globally optimal, we simply define a new set \(A_{k}(\ell) \in \mathcal{Y}\)

\[
A_{k}(\ell) := \{(x, \chi/n) \mid \text{LO}_k(x) = k\}.
\]

Thus, we define our partition of \(\mathcal{Y}\) to consist of all sets \(A_{j}(\ell)\) from Definition 4, where \(j \in [0..k-1]\) and \(\ell \in [d_j]\), and \(A_{k}(\ell)\). Hence, there are \(m := (\sum_{j=0}^{k-1} d_j) + 1\) levels.

We now prove that the levels form a partition of the extended search space \(\mathcal{Y}\). To simplify the proof, we first define some upper bounds for the sublevels. For all \(\ell \leq d_j\), the definition of \(d_j\) and Lemma 3 part 7) imply that

\[
A^{\ell-1} \varepsilon < \theta_1(j) < \theta_2(j + 1)
\]

which leads to the upper bounds

\[
A_{j}(\ell) \subset A_{j} \times \left[ A^{\ell-1} \varepsilon, \min( \frac{1}{2}, \theta_2(j)) \right]
\]

\[
\subset A_{j} \times \left[ A^{\ell-1} \varepsilon, \theta_2(j) \right].
\]

Furthermore, for all \(v \leq d_j\), (4) gives the trivial upper bound

\[
A_{j}(v) \subset A_j \times (\varepsilon, A^v \varepsilon).
\]

Lemmas 1 and 2 imply that we have a partition of the search space.

**Lemma 1:** For all \((u, v) \neq (j, \ell)\), it holds \(A_{u}(u) \cap A_{v}(v) = \emptyset\).

**Proof:** The following three cases use that \((X \times Y) \cap (U \times V) = (X \cap U) \times (Y \cap V)\).

**Case 1:** \(u = j\). Assuming w.l.o.g. that \(v < \ell \leq d_u\), the bounds (8) and (9) give

\[
A_{u,(v)} \cap A_{j}(\ell) \subset A_u \times \left( (\varepsilon, A^v \varepsilon) \cap [A^{\ell-1} \varepsilon, \infty) \right) = \emptyset
\]

where the last equality follows from \(A^v \varepsilon \leq A^{\ell-1} \varepsilon\).

**Case 2:** \(u + 1 \leq j\) and \(v < d_u\). In this case, the bounds (8) and (9) give

\[
A_{u,(v)} \cap A_{j}(\ell) \subset (A_u \cap A_{j}) \times (\varepsilon, \infty) = \emptyset
\]

where the last equality follows from \(A_u \cap A_{j} = \emptyset\).

**Case 3:** \(u + 1 \leq j\) and \(v = d_u\). In this case, the bound (7) and the definition of \(A_{u,d_u}\) in (5) give

\[
A_{u,(v)} \cap A_{j}(\ell) \subset A_{u,d_u} \cap \left( A_{j} \times (\varepsilon, \infty, \min( \frac{1}{2}, \theta_2(j))) \right)
\]

\[
\subset (A_{u} \cap A_{j}) \times \left( (\varepsilon, \min( \frac{1}{2}, \theta_2(u+1))) \infty \right) \varepsilon, \min( \frac{1}{2}, \theta_2(j)) \right) = \emptyset
\]

where the last equality follows from the fact that the function \(\theta_2\) decreases monotonically, thus \(\theta_2(j) \leq \theta_2(u+1)\).

**Lemma 2:** \(\bigcup_{j=0}^{k-1} \bigcup_{\ell=1}^{d_j} A_{j}(\ell) = \mathcal{Y}\).

**Proof:** We prove by induction on \(u \leq k\) that

\[
\bigcup_{j=u}^{k} \bigcup_{\ell=1}^{d_j} A_{j}(\ell) = A_{\geq u} \times \left( \varepsilon, \min( \frac{1}{2}, \theta_2(u)) \right).
\]
For the base step, (10) holds when \( u = k \) because by the level definition
\[
\bigcup_{\ell=1}^{d_j} A_{(k,\ell)} = A_k \times \left[ \varepsilon, \min \left( \frac{1}{2}, \theta_2(k) \right) \right]. \tag{11}
\]

For the inductive step, assume that (10) holds for some \( u \in [k] \). We then have
\[
\bigcup_{j=u-1}^{k} \bigcup_{\ell=1}^{d_j} A_{(j,\ell)} = \left( \bigcup_{\ell=1}^{d_{u-1}} A_{(u-1,\ell)} \right) \cup \left( A_{\geq u} \times \left[ \varepsilon, \min \left( \frac{1}{2}, \theta_2(u) \right) \right] \right) \cup \bigcup_{j=u}^{k-1} \bigcup_{\ell=1}^{d_j} A_{(j,\ell)} = A_{u-1} \times \left[ \varepsilon, \min \left( \frac{1}{2}, \theta_2(u-1) \right) \right] \cup \bigcup_{j=u}^{k-1} \bigcup_{\ell=1}^{d_j} A_{(j,\ell)}.
\]

By induction, (10) holds for \( u = 0 \). The proof is now complete by noting that \( \theta_2(0) > 1/2 \) due to Lemma 3 part 1).

The level-based theorem assumes that the levels are totally ordered, however we have introduced 2-D levels. We will order the levels using the lexicographic order \( \succeq \) defined for \( j, j', \ell, \ell' \) by
\[
A_{(j',\ell')} \succeq A_{(j,\ell)} \iff (j' > j) \lor (j' = j \land \ell' \geq \ell).
\]

Also, it will be convenient to introduce the notation
\[
A_{\geq (j,\ell)} := \bigcup \{ A_{(j',\ell')} \mid A_{(j',\ell')} \succeq A_{(j,\ell)} \}. \tag{12}
\]

### B. Survival and Upgrade Probabilities

Having partitioned the search space into levels, the next steps in applying the level-based theorem are to prove that conditions (G1) and (G2) are satisfied. This amounts to estimating the probability that an offspring does not decrease to a lower level (condition (G2)), and the probability that it upgrades to a strictly better level (condition (G1)). It will be convenient to introduce a measure for the probability of reproducing a bitstring of equal or better fitness by applying a mutation rate \( \chi/n \).

**Definition 5:** For all \( j \in [0..n] \) and \( \chi \in [\varepsilon n, n/2] \), we define the survival probability as
\[
r(j, \chi) := \min_{x \in A_{j}} \Pr \{ x' \in A_{\geq j} \}.
\]

For \textsc{leadingones}_k, it is straightforward to show that \( r(j, \chi) = (1 - \chi/n)^j \).

Fig. 1 illustrates a typical lineage of individuals, from fitness level \( j \) to fitness level \( j + 1 \). Starting from some low mutation rate in fitness level \( j \), the mutation rate is increased by a factor of \( A \) in each generation, until the mutation rate reaches the interval \([\theta_1(j), \theta_2(j)]\), i.e., the edge level. The lineage circulated within the edge level for some generations, crossing an intermediary value \( \eta(j) \), before the fitness improves, and fitness level \( j + 1 \) is reached.

It is critical to show that with sufficiently high probability, the lineage remains in the edge level before upgrading to fitness level \( j + 1 \). This is ensured by the bounds in Lemma 3. Statement 7) implies that we cannot overshoot the edge level by increasing the mutation rate. Statement 4) implies that below the intermediary mutation rate \( \eta(j) \), the mutation rate can still be increased by a factor of \( A \). Conversely, statement 5) means that above the intermediary mutation rate \( \eta(j) \), it is safe to decrease the mutation rate by a factor of \( b \). Statements 8) and 9) ensure that an individual in an edge level can always either increase or decrease mutation rate for there to be a sufficiently high probability of maintaining the individual’s fitness value. Finally, statements 2) and 3) imply that within the edge level, the upgrade probability is \( \Theta(1/j) \).

Before we can prove these statements, recall that we have delayed formally defining the functions \( \theta_1, \eta, \) or \( \theta_2 \). In order to derive the claimed bounds for Lemma 3, we do this now. For \( j \geq 1 \), let
\[
\eta(j) := \frac{1}{2A} \left( 1 - \left( \frac{1 + \delta}{\alpha_0 \mu_{\text{inc}}} \right)^{1/j} \right), \tag{13}
\]
\[
\theta_1(j) := b \eta(j), \tag{14}
\]
\[
\theta_2(j) := 1 - q^{1/j}, \tag{15}
\]

where
\[
q := \frac{1 - \xi}{\alpha_0}, \quad r_0 := \frac{1 + \delta}{\alpha_0 (1 - p_{\text{inc}})}, \quad \text{and} \quad \xi := 1 - \alpha_0 (r_0)^{1+\sqrt{r_0}}. \tag{16}
\]

Furthermore, for the special case \( j = 0 \), define
\[
\eta(0) := \frac{\eta(1)}{A}, \quad \theta_1(0) := b \eta(0), \quad \text{and} \quad \theta_2(0) := \frac{\theta_2(1)}{b}.
\]

Note that these definitions, along with statement 1) of Lemma 3, ensure \( \theta_1 \) and \( \theta_2 \) satisfy the informal conditions from Section III-A for \( \varepsilon \) small enough.

**Lemma 3:** Let \( A > 1, b < 1, \) and \( p_{\text{inc}} \in (0, 1) \) be constants satisfying the constraints in Theorem 2. Then there exists a
constant $\delta \in (0, 1/10)$ such that for all $j \in [0...k - 1]$ and $\chi/n \in [\varepsilon, 1/2]$.
1) $\theta_1(0) < \eta(0) < 1/2 < \theta_2(0)$.
2) $\theta_2(j) = \Omega(1/j)$.
3) $\theta_1(j) = 0(1/j)$.
4) $A\eta(j) \leq \theta_2(j)$.
5) $b\eta(j) \leq \theta_2(j)$.
6) $b\theta_2(j) < \theta_2(j + 1)$.
7) $A\theta_1(j) \leq \theta_2(j + 1)$.
8) if $(\chi/n) \leq \theta_2(j)$, then $r(j, Ax) \geq \lfloor(1 + \delta)/\eta(\alpha_0p_{\text{inc}})\rfloor$.
9) if $(\chi/n) \leq \theta_2(j)$, then $r(j, bx) \geq \lfloor(1 + \delta)/\eta(\alpha_0(1 - p_{\text{inc}}))\rfloor$.

Proof: Before proving statements 1)–9), we derive bounds on the three constants $q$, $\xi$, and $r_0$. By the assumptions $p_{\text{inc}} < 2/5$ and $\alpha_0 \geq 4$ from Theorem 2 and $\delta < 1/10$

$$r_0 < \frac{11}{6\alpha_0} < 1.$$  

Furthermore, since $r_0 < 1$ and $\alpha_0 \geq 4$, we have

$$\xi > 1 - \alpha_0(0)^{\alpha_0} > 1 - \frac{1}{\alpha_0} \left(\frac{11}{6}\right)^2 \geq \frac{23}{144}.$$  

Finally, since $\delta, \xi, p_{\text{inc}} \in (0, 1)$, we have from the definitions of $r_0$ and $q$ that

$$0 < q < r_0.$$  

From the definition of the functions $\theta_1$, $\eta$, and the constant $\delta \in (0, 1/10)$, it follows that:

$$\theta_1(0) < \eta(0) < \eta(1) < \frac{1}{2A} \left(1 - \frac{1}{\alpha_0p_{\text{inc}}}\right) \leq \frac{1}{2}.$$  

Also, we have from the definition of $q$, the constraint $\alpha_0 \geq 4$ from Theorem 2, and the bound $\xi > 23/144$ from (18) that

$$\theta_2(0) > \theta_2(1) > 1 - q = 1 - \frac{1 - \xi}{\alpha_0} > 1 - \frac{23}{4} = 455/576.$$  

Thus, we have proven statement 1).

Statement 2) follows directly from Lemma 7, the definition of $\theta_2$ and the constant $q$

$$\theta_2(j) = 1 - q^{1/j} \geq \ln(1/q)/j = \Omega(1/j).$$  

For statement 3), we define $c := \lfloor(1 + \delta)/\eta(\alpha_0p_{\text{inc}})\rfloor < 1$, and observe that the inequality $e^\delta \geq 1 + x$ implies

$$\theta_1(j) < 1 - c^{1/j} = 1 - e^{(1/j)\ln(c)} \leq -(1/j)\ln(c) = O(1/j).$$  

For statement 4), first note that (19) and the assumption $p_{\text{inc}} < 2/5$ imply

$$0 < q < r_0 < \frac{1}{\alpha_0p_{\text{inc}}}.$$  

For $j \geq 1$, we therefore have $1/j > 0$ and

$$\theta_2(j) = 1 - q^{1/j} \geq 1 - \left(\frac{1 + \delta}{\alpha_0p_{\text{inc}}}\right)^{1/j} \geq A\eta(j).$$  

For $j = 0$, the definition of $\eta(0)$, statement 4) for the case $j = 1$ shown above, and the definition of $\theta_2(1)$ give

$$A\eta(0) = \eta(1) \leq \frac{\theta_2(1)}{A} = \frac{b\theta_2(0)}{A} < \theta_2(0).$$  

Statement 5) follows from the definition of $\theta_1(j)$.

We now show 6). The statement is true by definition for $j = 0$, so assume that $j \geq 1$. We first derive an upper bound on the parameter $b$ in terms of the constant $q$. In particular, the constraint on $b$ from Theorem 2 and the relationship $0 < q < r_0$ from (19) give

$$b \leq \frac{1}{1 + \sqrt{r_0}} < \frac{1}{1 + \sqrt{q}} = \frac{1 - \sqrt{q}}{1 - q}.$$  

The right-hand side of (20) can be further bounded by observing that the function $g(j) := (1 - q^{1/j+1})/(1 - q^{1/j})$ with $q > 0$ increases monotonically with respect to $j$. Thus, for all $j \in \mathbb{N}$, we have

$$b < \frac{1 - \sqrt{q}}{1 - q} \leq \frac{1 - q^{1/j+1}}{1 - q^{1/j}}.$$  

This upper bound on parameter $b$ now immediately leads to the desired result

$$b\theta_2(j) = b \left(1 - q^{1/j}\right) \leq 1 - q^{1/(j+1)} = \theta_2(j + 1).$$  

Statement 7) follows by applying the previous three statements in the order 4)–6)

$$A\theta_1(j) \leq Ab\eta(j) \leq b\theta_2(j) \leq \theta_2(j + 1).$$  

Next we prove statement 8). The statement is trivially true for $j = 0$, because $r(0, Ax) = 1$, so assume that $j \geq 1$. By the assumption $\chi/n \leq \eta(j)$ and the definition of $\eta(j)$

$$r(j, Ax) = \left(1 - A\frac{\chi}{n}\right)^j \geq (1 - A\eta(j))^j \geq \left(1 - \left(1 - \left(1 + \delta \frac{\xi}{\alpha_0p_{\text{inc}}}\right)^{1/j}\right)\right)^j \geq 1 + \delta \frac{\xi}{\alpha_0p_{\text{inc}}}.$$  

Finally, we prove statement 9). Again, the statement is trivially true for $j = 0$, because $r(0, bx) = 1$, so assume that $j \geq 1$. We derive an alternative upper bound on parameter $b$ in terms of $r_0$ and $q$. By the constraint on $b$ in Theorem 2

$$b \leq \frac{1}{1 + \sqrt{r_0}} \leq \frac{\ln(r_0)}{\ln(r_0) + \sqrt{r_0}\ln(r_0)}.$$  

Furthermore, note that the function $h(j) := [(1 - r_0^{1/j})/(1 - q^{1/j})]$ decreases monotonically with respect to $j$ when $r_0 > q > 0$, and has the limit $\lim_{j \to \infty} h(j) = \ln(r_0)/\ln(q)$. Using (23), it therefore holds for all $j \in \mathbb{N}$ that

$$b \leq \frac{\ln(r_0)}{\ln(q)} \leq \frac{1 - r_0^{1/j}}{1 - q^{1/j}}.$$  

The assumption $\chi/n \leq \theta_2(j)$, the definition of $\theta_2(j)$, and (24) now give

$$r(j, bx) = \left(1 - b\frac{\chi}{n}\right)^j \geq (1 - b\theta_2(j))^j \geq \left(1 - \left(1 - r_0^{1/j}\right)\right)^j = r_0$$  

which completes the proof of statement 9).
Using Lemma 3, we are now in a position to prove that the levels satisfy condition (G2). If the mutation rate $\chi/n$ is above the intermediary value $\eta(j)$, there is a sufficiently high probability of reducing the mutation rate while maintaining the fitness. Conversely, if the mutation rate is below the intermediary value $\eta(j)$, there is a sufficiently high probability of increasing the mutation rate while maintaining the fitness.

**Lemma 4:** Assume that the parameters $A$, $b$, and $p_{\text{inc}}$ satisfy the constraints in Theorem 2. Then there exists a constant $\delta \in (0, 1/10)$ such that for all $j \in [0, k-1]$ and all $\ell \in [d_j]$, if Algorithm 1 in step 4 selects a parent $(x, \chi/n) \in A_{j, \ell}$, then the offspring $(x', \chi'/n)$ created in steps 5 and 6 of the algorithm satisfies

$$\Pr((x', \chi'/n) \in A_{\geq j, \ell}) \geq \frac{1 + \delta}{\alpha_0}. \tag{25}$$

**Proof:** We will prove the stronger statement that with probability $(1 + \delta)/\alpha_0$, we have simultaneously

$$x' \in A_{\geq j} \text{ and } \min\left\{ \frac{X_j}{n}, \theta_1(j) \right\} \leq \frac{x'}{n} \leq \theta_2(j). \tag{25}$$

The event (25) is a subset of the event $(x', \chi'/n) \in A_{\geq j, \ell}$, because a lower level $A_{j, \ell}$ may contain search points $(x', \chi'/n)$ with mutation rates $\chi'/n < \min(\chi/n, \theta_1(j))$.

By Definition 4, the parent satisfies $x \in A_{\geq j}$ and $\chi/n \leq \theta_2(j)$. We distinguish between two cases.

**Case 1 ($\chi/n \leq \eta(j)$):** By Lemma 3 1), and the monotonicity of $\eta$, we have $\eta(j) < 1/2$. Note that in this case, it is still “safe” to increase the mutation rate. For a lower bound, we therefore pessimistically only account for offspring where the mutation parameter is increased from $\chi < n/2$ to $\min(A\chi, n/2)$. Note first that since $A > 1$, we have

$$\frac{x'}{n} = \min(A\chi, n/2) > \frac{X_j}{n}. \tag{25}$$

Also, Lemma 3 4) implies the upper bound

$$\frac{x'}{n} \leq \frac{AX_j}{n} \leq A\eta(j) \leq \theta_2(j). \tag{25}$$

To lower bound the probability that $x' \in A_{\geq j}$, we consider the event where the mutation rate is increased, and the event that none of the first $b$ bits in the offspring are mutated with the new mutation parameter $\min(A\chi, n/2)$. By definition of the algorithm and using Lemma 3 8), these two events occur with probability at least

$$p_{\text{inc}}r(j, A\chi) \geq (1 + \delta)/\alpha_0. \tag{26}$$

**Case 2 ($\eta(j) < \chi/n \leq \theta_2(j)$):** Note that in this case, it may be “unsafe” to increase the mutation rate. For a lower bound, we pessimistically only consider mutation events where the mutation parameter is decreased from $\chi$ to $b\chi$. Analogously to above, since $b < 1$, we have

$$\frac{x'}{n} = \frac{b\chi}{n} < \frac{X_j}{n} \leq \theta_2(j). \tag{27}$$

Furthermore, Lemma 3 5) implies the lower bound

$$\frac{x'}{n} = \frac{b\eta(j)}{n} \geq \theta_1(j). \tag{28}$$

To lower bound the probability that $x' \in A_{\geq j}$, we consider the event where the mutation parameter is decreased from $\chi$ to $b\chi$, and the offspring $x'$ is not downgraded to a lower level. By definition of the algorithm, $r(j, b\chi)$, and using Lemma 3 9), these two events occur with probability

$$(1 - p_{\text{inc}})r(j, b\chi) \geq (1 + \delta)/\alpha_0. \tag{29}$$

Hence, we have shown that in both cases, the event in (25) occurs with probability at least $(1 + \delta)/\alpha_0$, which completes the proof. \qed

We now show that the edge levels satisfy condition (G1) of the level-based theorem. As we will show later, the upgrade probability for nonedge levels is constant.

**Lemma 5:** Assume that the parameters $b$ and $p_{\text{inc}}$ satisfy the constraints in Theorem 2. Then for any $j \in [0, k-1]$, and any search point $(x, \chi/n) \in A_{j, d_j}$ selected in step 4 of Algorithm 1 applied to $\text{LEADINGONES}_k$, the offspring $(x', \chi'/n)$ created in steps 5 and 6 satisfies $\Pr((x', \chi'/n) \in A_{\geq j+1, 1}) = \Omega(1/j)$. \smallbreak

**Proof:** By the definition of level $A_{j, d_j}$, we have $\theta_1(j) \leq \chi/n \leq \theta_2(j)$ and so by Lemma 3 6), we have $b\chi/n \leq \theta_2(j+1)$. Given the definition of levels $A_{j+1, 1}$, it suffices for a lower bound to only consider the probability of producing an offspring $(x', \chi'/n)$ with lowered mutation rate $\chi'/n = b\chi/n \leq \theta_2(j+1)$ and fitness $\text{LOQ}_j(x') \geq j + 1$. \smallbreak

We claim that if the mutation rate is lowered, the offspring has fitness $\text{LOQ}_j(x') \geq j + 1$ with probability $\Omega(1/j)$. Since the parent belongs to level $A_{j, d_j}$, it has fitness $\text{LOQ}_j(x) \geq j$, so we need to estimate the probability of not flipping the first $j$ bits, and obtain a 1-bit in position $j + 1$.

We now estimate the probability of obtaining a 1-bit in position $j + 1$, assuming that the parent $x$ already has a 1-bit in this position, for any $j \in [0, k-1]$. Using that $\theta_2(j)$ decreases monotonically in $j$, the definition of $\theta_2(0)$, and the lower bound on the parameter $\xi > 23/144$ from (18), the probability of not mutating bit-position $j+1$ with the lowered mutation rate $b\chi/n$ is

$$1 - \frac{b\chi}{n} \geq 1 - b\theta_2(0) \geq 1 - b\theta_2(0)$$

$$= 1 - \theta_2(1) = 1 - \frac{1 - \xi}{\alpha_0} > 1 - \frac{1 - 23}{\alpha_0} = \Omega(1). \tag{30}$$

If the parent $x$ does not have a 1-bit in position $j + 1$, we need to flip this bit-position. By the definition of $\theta_1(j)$ in (14), the probability of this event is in the case $j \geq 1$

$$\frac{b\chi}{n} \geq b\theta_1(j) = \frac{b^2}{2A} \left(1 - \left(\frac{1 + \delta}{\alpha_0 p_{\text{inc}}}\right)^{1/\ell}\right) \tag{31}$$

where the last inequality follows from Lemma 7. If $j = 0$, we use that $\theta_1(j)$ decreases monotonically in $j$ and (30), (31) to show that the probability of flipping bit $j + 1 = 1$ is

$$\frac{b\chi}{n} \geq b\theta_1(0) > b\theta_1(1) = \Omega(1). \tag{30}$$

The claim that we obtain a 1-bit in position $j + 1$ with probability $\Omega(1/j)$ is therefore true.
Thus, the probability of lowering the mutation rate to \( b \tau / n \), obtaining a 1-bit in position \( j + 1 \), and not flipping the first \( j \) positions is, using the definition of \( \theta_2(j) \) in (15)
\[
(1 - p_{\text{mu}})\Omega(1/j) \left( 1 - \frac{b \tau}{n} \right)^j > \Omega(1/j)(1 - \theta_2(j))^j = \Omega(1/j) \left( \frac{1 - \xi}{\alpha_0} \right) = \Omega(1/j)
\]
which completes the proof.

### C. Individuals With Too High Mutation Rates

Highly fit individuals with incorrect parameter settings can cause problems for self-adaptive EAs. If there are too many such “bad” individuals in the population, they may dominate the population, propagate bad parameter settings, and thus impede progress. In this section, we, therefore, bound the number of such bad individuals. We define a region \( B \subseteq \mathcal{Y} \) containing search points with a mutation rate that is too high relative to their fitness. For the constant \( \xi \in (0, 1) \) defined in (16), let
\[
B = \left\{ (x, \chi/n) \in A_j \times [\varepsilon, 1/2] \mid j \in \mathbb{N}_0 \wedge \forall y \in \mathcal{X}, \Pr_{x' \sim p_{\text{mu}}(y, x)}(x' \in A_{\geq 2j}) < \frac{1 - \xi}{\alpha_0} \right\}.
\]

Note that by the definition of the function \( \theta_2(j) \), it holds for all \( y \in \mathcal{X} \) that the statement \( \Pr_{x' \sim p_{\text{mu}}(y, x)}(x' \in A_{\geq 2j}) = (1 - \chi/n)^j < (1 - \xi)/\alpha_0 \) is analogous to \( \chi/n > \theta_2(j) \). Therefore, the region \( B \) can also be expressed as
\[
B = \bigcup_{j=0}^{k-1} A_{\geq j} \times \left( \min \left( 1, \frac{1}{\alpha_0}, \frac{1}{\tau} \theta_2(j + 1) \right), \min \left( 1, \frac{1}{\alpha_0}, \theta_2(j) \right) \right).
\]

An individual \( (x, \chi/n) \in B \) is said to have too high mutation rate. To see why, recall that the number of offspring of \( (x, \chi/n) \) is never more than \( \alpha_0 \). Since the probability an offspring has fitness as least as good as \( \frac{1}{\alpha_0} \), in expectation less than one offspring maintains the fitness, making it unlikely a lineage of \( (x, \chi/n) \) will be able to make progress toward the optimum. This corresponds to the “error threshold” discussed in [29]; by [29, Corollary 1], the probability that a group of individuals of size \( \text{poly}(k) \) staying in \( B \) will optimize \( \text{LO}_k \) in subexponential time is \( e^{-\Omega(k)} \).

The levels given by Definition 4 are not disjoint from the region \( B \) defined above. This is an important departure from the approach used in [8], where \( B \) is effectively removed from the search space and the level-based theorem is applied to a partition over \( \mathcal{Y} \setminus B \). This is not effective in our setting, since an individual can have a sudden increase in fitness but cannot significantly decrease mutation rate. Such an individual may have too high mutation rate with respect to its new fitness, but we still depend on this individual having correctly tuned mutation with respect to the old fitness value. Therefore, an individual in some \( A_{(j, \ell)} \) may mutate in and out of \( B \) before its mutation rate has been adapted to maintain a fitness higher than \( j \). While individuals may occasionally jump into the bad region, if too many individuals are in \( B \) at a given time this may destroy the progress of the algorithm. In particular, if \(|P_t \cap B| > \mu_\ell \), then assuming all individuals in \( B \) have strictly better fitness and higher mutation rate than those not in \( B \), only individuals from \( B \) will be selected for mutation and the next generation will consist only of individuals with very high mutation rate. Therefore, it is critical that for any generation \( t \in \mathbb{N} \), the number of individuals in \( B \) will be less than \( \mu_\ell \) with overwhelmingly high probability. We prove this with the following lemma, which is similar to Lemma 2 from [8], but with the notable difference that the size of \( B \) can be controlled within a single generation, regardless the configuration of \( P_{t-1} \).

**Lemma 6:** Let \( B \subseteq \mathcal{Y} \) be as defined in (32) for a constant \( \xi \in (0, 1) \). Then for any generation \( t \in \mathbb{N} \) of Algorithm 1 applied to \( \text{LeadingOnes}_k \)
\[
\Pr(|B \cap P_t| \geq (1 - \xi/2)\mu_\ell) \leq e^{-\Omega(\mu_\ell)}.
\]

**Proof:** Consider some parent \( (x, \chi/n) \) selected in generation \( t - 1 \geq 0 \) and step 4 of Algorithm 1. Referring to steps 5 and 6, first a new mutation parameter \( \chi' \) is chosen, then a new bitstring \( x' \) is obtained from \( x \) using bitwise mutation with mutation parameter \( \chi' \). To obtain an upper bound on the probability that \((x', \chi'/n) \in B \), we proceed in cases based on the outcome of sampling \( \chi' \), namely, whether the chromosome \((x, \chi'/n) \) is in \( B \).

If \((x, \chi'/n) \in B \): Then it follows immediately from the definition of \( B \) that independently of the chosen parent \( x \), it holds
\[
\Pr((x', \chi'/n) \in B) < \frac{1 - \xi}{\alpha_0},
\]

If \((x, \chi'/n) \notin B \): Then for \((x', \chi'/n) \) to end up in \( B \), by (33) it is necessary that \( x' \in A_{\geq u} \) for some \( u > j \), where \( x \in A_j \) and \( r(u, \chi') < (1 - \xi)/\alpha_0 \). Since \( \chi'/n < 1/2 \), the probability of obtaining \( x' \in A_{\geq u} \) is no more than
\[
\left( 1 - \frac{\chi'}{n} \right)^{u-1} \left( \frac{\chi'}{n} \right) < \left( 1 - \frac{\chi'}{n} \right)^u < \frac{1 - \xi}{\alpha_0}.
\]

Since each of the \( \lambda \) individuals in population \( P_t \) are sampled independently and identically, (34) and (35) imply \(|B \cap P_t| \) is stochastically dominated by a binomially distributed random variable \( Z \sim \text{Bin}(\lambda, [(1 - \xi)/\alpha_0]) \) which has expectation \( \mu(1 - \xi) \). By a Chernoff bound
\[
\Pr(|B \cap P_t| \geq \mu(1 - \xi/2)) \leq \Pr(Z \geq \mu(1 - \xi/2)) = \Pr(Z \geq E[Z] \left(1 + \frac{1}{2(1 - \xi)}\right)) = e^{-\Omega(\mu_\ell)}.
\]

### D. Applying the Level-Based Theorem

We now combine the results of Sections III-A–III-C to prove Theorem 2 using Theorem 1.

**Proof (Theorem 2):** We partition the search space \( \mathcal{Y} \) into the sets \( A_{(j, \ell)} \) from Definition 4, where \( j \in [0..k-1] \) and \( \ell \in [d_j] \), along with \( A_{(k, 1)} \), and define \( A_{(j, \ell)} \) as in (12).

We say that a generation \( t \) is “failed” if the population \( P_{t} \) contains more than \((1 - \xi/2)\mu_\ell \) individuals in region \( B \).
First, we will optimistically assume that no generations fail. Under this assumption, we will prove that the conditions of Theorem 1 hold, leading to an upper bound on the expected number of function evaluations \( t_0(k) \) until a search point in \( A_{(k,1)} \) is created (i.e., a global optimum is found). Then in the end we will use a restart argument to account for failed generations.

Let \( y_0 := (\zeta/2)(\mu/\lambda) \). In the following arguments, we will make consistent use of the important fact that for \( \gamma \in (0, y_0) \), if there are \( \gamma \lambda \) individuals in levels \( A_{(j,\ell)} \) for some \( j \in [0, \ldots, k-1] \) and \( \ell \in [d_j] \), then the probability of selecting an individual from \( A_{(j,\ell)} \) is \( y_0 \). To see this, we note that individuals in \( A_{(j,\ell)} \) are guaranteed to be ranked above those in \( Y \setminus (A_{(j,\ell)} \cup B) \) in line 2 of Algorithm 1, since individuals in \( Y \setminus (A_{(j,\ell)} \cup B) \) must have fitness either strictly less than \( j \), or equal to \( j \), and hence have mutation rate too low to be contained in \( A_{(j,\ell)} \). Recalling that \( |P| \land |\mathcal{B}| \leq (1 - \zeta/2)\mu \), it follows that all \( \gamma \lambda \leq (\zeta/2)\mu \) individuals of \( A_{(j,\ell)} \) are among the \( \mu \) fittest in the population. Therefore, the probability of selecting an individual from \( A_{(j,\ell)} \) indeed is \( \gamma \lambda / \mu = y_0 \).

We now show that conditions (G1) and (G2) of Theorem 1 hold for each level \( A_{(j,\ell)} \) where \( j \in [0, \ldots, k-1] \) and \( \ell \in [d_j] \). We assume that the current population has at least \( y_0 \lambda \) individuals in levels \( A_{(j,\ell)} \). We distinguish between the case \( \ell < d_j \), i.e., when it suffices to increase the mutation rate to upgrade to the next level, and the case \( \ell = d_j \), i.e., when it may be necessary to increase the fitness to reach the next level.

\( \ell < d_j \): To verify condition (G2), we must estimate the probability of producing an offspring in levels \( A_{(j,\ell+1)} \), assuming that there are at least \( \gamma \lambda \) individuals in levels \( A_{(j,\ell+1)} \), for any \( \gamma \in (0, y_0) \). To produce an offspring in levels \( A_{(j,\ell+1)} \), it suffices to first select a parent \( (x, \chi/n) \) from \( A_{(j,\ell+1)} \), and second create an offspring \( (x', \chi'/n) \) in levels \( A_{(j,\ell+1)} \). The probability of selecting such a parent is at least \( y_0 \). Assuming that the parent is in level \( A_{(j,\ell+1)} \), and applying Lemma 4 to level \( A_{(j,\ell+1)} \), the probability that the offspring \( (x', \chi'/n) \) is in levels \( A_{(j,\ell+1)} \) is \( (1 + \delta)/\alpha_0 \) for some \( \delta \in (0, 1) \). Thus, the probability of selecting a parent in levels \( A_{(j,\ell+1)} \), then producing an offspring in levels \( A_{(j,\ell+1)} \), is at least \( y_0 (1 + \delta)/\alpha_0 \). Since \( \gamma \lambda / (1 + \delta) \alpha_0 \) is condition (G2) is satisfied.

To verify condition (G1), we estimate the probability of producing an offspring in levels \( A_{(j,\ell+1)} \). If the parent is in levels \( A_{(j,\ell+1)} \), then again by Lemma 4, the probability is at least \( (1 + \delta)/\alpha_0 \).

On the other hand, if the parent \( (x, \chi/n) \) is in level \( A_{(j,\ell)} \), then we consider the probability of producing an offspring \( (x', \chi'/n) \in A_{(j,\ell+1)} \) by increasing the mutation rate from \( \chi \) to \( A_{(j,\ell+1)} \) and maintaining the fitness \( x' \in A_{(j,\ell+1)} \). By assumption, \( \ell < d_j \), so the level-definition implies that the parent has mutation rate \( \chi/n < \theta_1(j) < \eta(j) \). Hence, by Lemma 3 8), the probability of increasing the mutation parameter to \( A_X \) and maintaining at least \( j \) leading one-bits is at least \( p_{\text{inc}}^\ell (j, A_X) \geq (1 + \delta)/\alpha_0 \).

Taking into account that the probability of selecting a parent in \( A_{(j,\ell)} \) is at least \( \alpha_0 y_0 \), the probability of producing an offspring in \( A_{(j,\ell+1)} \) is at least

\[
y_0 \alpha_0 (1 + \delta/\alpha_0) = (1 + \delta) y_0 = z(j,\ell). \tag{36}
\]

\( \ell = d_j \): To show (G2) we assume that there are at least \( \gamma \lambda \) individuals in levels \( A_{(j+1,\ell)} \), for \( \gamma \in (0, y_0) \). We again apply Lemma 4 to show the probability of selecting an individual from \( A_{(j+1,\ell)} \) and producing a new individual also in \( A_{(j+1,\ell)} \) is at least \( \gamma (1 + \delta) \), showing (G2) is satisfied.

For condition (G1), we only consider parents selected from levels \( A_{(j+1,\ell)} \). If the parent \( (x, \chi/n) \) is in \( A_{(j,d_j)} \), then by Lemma 5 the offspring \( (x', \chi'/n) \) is in levels \( A_{(j+1,\ell)} \) with probability at least \( \Omega(1/j) \). Otherwise, if the parent is already in levels \( A_{(j+1,\ell)} \), then by Lemma 4, the offspring is in levels \( A_{(j+1,\ell)} \) with probability at least \( (1 + \delta)/\alpha_0 = \Omega(1) \). In both cases, the probability of selecting a parent from \( A_{(j,d_j)} \) and producing an offspring in levels \( A_{(j+1,\ell)} \) is at least

\[
y_0 \alpha_0 \Omega(1/j) = \Omega(1/j) = z(j,d_j). \tag{37}
\]

To verify that \( \lambda \geq c \ln(n) \) is large enough to satisfy condition (G3), we must first calculate \( m \), the total number of sublevels. Referring to Definition 3 and using \( \theta_1(j) \leq 1/2 \), the depth of each level \( j \) is no more than \( \log_A(1/2\epsilon) = O(\log(n)) \) for all \( j \in [0, \ldots, k-1] \). Therefore, \( m = O(k \log(n)) \) and so \( \lambda \geq c \ln(n) \) satisfies (G3) for \( c > 1 \) large enough. Thus, we have found parameters \( z_0(0), z_0(1), \ldots, z_0(k-d_k-1), \delta \) and \( y_0 \) such that all three conditions of Theorem 1 are satisfied. Assuming no failure, the expected time to reach the last level is no more than

\[
t_0(k) \leq \left(\frac{8}{\delta^2}\right) \sum_{j=0}^{k-1} \sum_{\ell=1}^{d_j} \left(\lambda \log \left(\frac{6\delta\lambda}{4 + z(j,\ell)\delta}\right) + \frac{1}{z(j,\ell)}\right)
\]

\[
= \sum_{j=0}^{k-1} \sum_{\ell=1}^{d_j} O \left(\lambda \log \left(\frac{1}{z(j,\ell)}\right) + \frac{1}{z(j,\ell)}\right)
\]

\[
+ \sum_{j=0}^{k-1} \sum_{\ell=1}^{d_j} \left(\lambda \log(\lambda) + \frac{\lambda}{z(j,\ell)}\right)
\]

\[
= O(k \lambda \log(n) + k\lambda \lambda(\log(\lambda) + k^2)
\]

using that \( z(j,\ell) = \Omega(1) \) for all \( \ell < d_j \), and \( z(j,d_j) = \Omega(1/j) \).

Finally, we account for failed generations where our assumption that there are less than \((1 - \zeta/2)\mu\) individuals in region \( B \) does not hold. We refer to a sequence of \( 2t_0(k)/\lambda \) generations as a phase, and call a phase good if for \( 2t_0(k)/\lambda \) consecutive generations there are fewer than \((1 - \zeta/2)\mu\) individuals in \( B \). By Lemma 6 and a union bound, a phase is good with probability \( 1 - 2t_0(k)/\lambda e^{-\Omega(\mu)} = \Omega(1) \), for \( \mu = \Omega(\log(n)) \). By Markov’s inequality, the probability of reaching a global optimum in a good phase is at least \( 1/2 \). Hence, the expected number of phases required, each costing \( 2t_0(k) \) function evaluations, is \( O(1) \).

IV. EXPERIMENTS

The theoretical analysis of the \((\mu, \lambda)\) self-adaptive EA is complemented by some experiments on a wider variety of
problems. In addition to the standard ONE MAX function, we consider

\[
JUMP_k(x) := \begin{cases} 
\text{OM}(x) + k & \text{if } \text{OM}(x) < n - k \\
\text{OM}(x) - k & \text{if } n - k \leq \text{OM}(x) < n - 1 \\
1 & \text{if } \text{OM}(x) = n 
\end{cases}
\]

\[
\text{SUBSTRING}_k(x) := \max_{1 \leq i \leq n} i \cdot \prod_{j=\max\{i-k+1,1\}}^{i} x_j.
\]

The \text{SUBSTRING}_k function is similar to the function in [6] of the same name. The value of \text{SUBSTRING}_k is the maximal position of the substring 1^k, if such a substring exists, otherwise it is just the number of leading 1-bits. While the function in [6] has a unique global optimum at the point 1^n, all strings of the form \{0, 1\}^{n-k}1^k are optimal for our \text{SUBSTRING}_k function.

In a first set of experiments, we examined how Algorithm 1 adjusts mutation rates relative to fitness on several contrasting fitness landscapes. In each run, we chose the parameter settings \( \lambda = 8 \ln(n) \), \( \mu = \lambda/15 \), \( A = 1.5 \), \( b = 0.7 \), and \( p_{\text{inc}} = 0.25 \) for Algorithm 1. The change in parameter settings was not particularly motivated, although note that both respect the conditions imposed by Theorem 2, since \( p_{\text{inc}} = 0.25 \) satisfies \( 1/16 < 1/15 < 1/4 < 2/5 \), and \( b = 0.7 \) satisfies \( 7/10 < 1/(1 + \sqrt{4/45}) \approx 0.78 \).

The results from the first set of experiments are summarized in Fig. 2. We set \( n = 500 \) for \text{LEADING ONES}, \text{SUBSTRING}_7, and \text{ONE MAX}, while for \text{JUMP}_3 we set \( n = 100 \), and performed 100 trials for each function. At the beginning of a
trial, all individuals were given a starting mutation strength of $\chi = 1$. For each function, we plotted the median mutation rate $\chi/n$ per fitness value with the 95th percentile shaded in gray. Finally, to aid interpretation we include the error threshold, i.e., the value of $\chi/n$ such that the expected number of offspring with fitness at least as good as the parent’s is only 1 [29]. For LEADINGONES, the error threshold is thus approximately the $\theta_2$ function introduced in Section III-A.

Fig. 2 shows that Algorithm 1 tuned the mutation rate of the top-ranked individual very differently depending on the fitness landscape. For LEADINGONES, we see the top individual’s mutation rate quickly rose to a small factor below $\theta_2$, then gradually lowered mutation rate as fitness increased. This supports our theoretical analysis of LEADINGONESk, in which we argued that the mutation rate rises to an “edge region” comprising of mutation rates just below the error threshold. We found similar behavior for SUBSTRINGk, where again the algorithm quickly rose to a close approximation below the error threshold. However, the results for ONE MAX and JUMP3 are less conclusive. First, we were unable to derive an exact expression for the error threshold for these functions, which makes the trajectory of the mutation rates more difficult to interpret. Instead, we include the mutation rate for a single individual to maximize the expected difference in its fitness before and after mutation, in order to provide some context for interpreting the effectiveness of mutation rates. For ONE MAX, it is known this drift-maximizing rate is $\Theta(1/n)$ when $OM(x) \geq 2n/3$ [18], while for JUMP3, the ideal rate is $3/n$ for jumping the gap. In terms of the trajectory of mutation rates, on ONE MAX the algorithm correctly increased its mutation rate at first, but also seems to have kept mutation rate well above $1/n$ for much of the search process. This could explain its relative inefficiency on ONE MAX along the edge region, in the next set of experiments. The behavior is similar for JUMP3, except that mutation rate increased toward the ideal rate while at the edge of the gap, and occasionally reached even higher values. The tendency for mutation rate to dramatically increase during lack of progress is reassuring, since a common difficulty in self-adaptation of mutation rates is that mutation rates may indefinitely decrease when it is difficult to increase fitness [35].

In the second set of experiments, summarized in Fig. 3, we compared the self-adaptive EA to the $(1 + 1)$ EA, the $(\mu, \lambda)$ EA, as well as to the $(1 + 1)_{\alpha}$ EA from [23] with the parameter settings $A = 1.2$ and $b = 0.85$ [for the $(1 + 1)_{\alpha}$ EA]. On each of the functions LEADINGONESk, SUBSTRINGk, and ONE MAX, we tested the algorithms on a range of possible choices for the adversary by performing 100 runs of each algorithm for values of $k$ between 100 and $n = 2000$. The $y$-axes in Fig. 3 shows the runtime divided by the asymptotic running time of a $(1 + 1)$ EA which knows the value $k$ beforehand. The effect of this rescaling is that algorithms which successfully adapt to the parameter $k$ should remain relatively constant along the $y$-axis as $k$ changes.

On all three functions, the two adaptive algorithms had runtimes proportional to an EA which knew $k$ beforehand. However, while both also drastically outperformed the static algorithms for smaller $k$ on LEADINGONESk and SUBSTRINGk, on ONE MAX, Algorithm 1 performed comparably to the static $(1 + 1)$ EA only for small $k$, and did worse than the $(1 + 1)$ EA as $k$ grew larger. This is somewhat expected, since it is known that the $(1 + 1)$ EA easily outperforms many population-based algorithms on ONE MAX.

It is possible that the benefits of adaptation will not overcome the penalty of maintaining a population except for much larger values of $n$.

V. Conclusion

Effective parameter control is one of the central challenges in evolutionary computation. There is empirical evidence that self-adaptation—where parameters are encoded in the chromosome of individuals—can be a successful control mechanism in evolutionary strategies. However, self-adaptation is rarely employed in discrete EAs [1], [36]. The theoretical understanding of self-adaptation is lacking.

This article demonstrated both theoretically and empirically that adopting a self-adaptation mechanism in a discrete, nonelitist EA can lead to significant speedups. We analyzed the expected runtime of the $(\mu, \lambda)$ EA with self-adaptive mutation rates on LEADINGONESk in the context of an adversarial choice of a hidden problem parameter $k$ that determines the problem structure. We gave parameter settings for which the algorithm optimizes LEADINGONESk in expected time $O(k^2)$, which is asymptotically optimal among any unary unbiased black-box algorithm which knows the hidden value $k$. This is a significant speedup compared to, e.g., the $(1+1)$ EA using any choice of static mutation rate. In fact, the algorithm even has an asymptotic speedup compared with the state-of-the-art parameter control mechanism for this problem [16]. Future work should extend the analysis to more general classes of problems, such as linear functions and multimodal problems. We expect that applying the level-based theorem over a 2-D level-structure will lead to further results about self-adaptive EAs.

APPENDIX

Lemma 7: For all $c > 0$ and $j > 0$, $1 - c^{1/j} \geq \ln(1/c)/j$.

Proof: $-(1/j) \ln(1/c) = \ln(c^{1/j}) \leq c^{1/j} - 1$.

ACKNOWLEDGMENT

The authors would like to thank Dr. Duc-Cuong Dang for suggesting the problem of determining the runtime of a self-adaptive evolutionary algorithm on the LEADINGONESk problem.

REFERENCES

[1] T. Bäck, “Self-adaptation in genetic algorithms,” in Proc. 1st Eur. Conf. Artif. Life, 1992, pp. 263–271.
[2] T. Bäck and M. Schütz, “Intelligent mutation rate control in canonical genetic algorithms,” in Proc. Found. Intell. Syst., 1996, pp. 158–167.
[3] G. Badkobeh, P. K. Lehre, and D. Sudholt, “Unbiased black-box complexity of parallel search,” in Proc. 13th Int. Conf. Parallel Problem Solving Nat. (PPSN’14), 2014, pp. 892–901.
[4] S. Böttcher, B. Doerr, and F. Neumann, “Optimal fixed and adaptive mutation rates for the leadingones problem,” in Proc. 11th Int. Conf. Parallel Problem Solving Nat. (PPSN’10), 2010, pp. 1–10.
[5] S. Cathabard, P. Lehre, and X. Yao, “Non-uniform mutation rates for problems with unknown solution lengths,” in Proc. 11th Workshop Found. Genet. Algorithms (FOGA’11), 2011, pp. 173–180.
[6] T. Chen, P. K. Lehre, K. Tang, and X. Yao, “When is an estimation of distribution algorithm better than an evolutionary algorithm?” in Proc. IEEE Congr. Evol. Comput., Trondheim, Norway, 2009, pp. 1470–1477.

[7] D. Corus, D.-C. Dang, A. V. Eremeev, and P. K. Lehre, “Level-based analysis of genetic algorithms and other search processes,” IEEE Trans. Evol. Comput., vol. 22, no. 5, pp. 707–719, Oct. 2018.

[8] D.-C. Dang and P. K. Lehre, “Self-adaptation of mutation rates in non-elitist populations,” in Proc. Int. Conf. Parallel Problem Solving Nat. (PPSN’16), 2016, pp. 803–813.

[9] D.-C. Dang, P. K. Lehre, and P. T. H. Nguyen, “Level-based analysis of the univariate marginal distribution algorithm,” Algorithmica, vol. 81, no. 2, pp. 668–702, 2019.

[10] K. De Jong, “A historical perspective,” in Evolutionary Computation: A Unifying Approach (ch. 2), 1st ed. Cambridge, MA, USA: MIT Press, 2006.

[11] B. Doerr and C. Doerr, “Optimal parameter choices through self-adjustment: Applying the 1/5-th rule in discrete settings,” in Proc. Genet. Evol. Comput. Conf. (GECCO’15), 2015, pp. 1335–1342.

[12] B. Doerr and C. Doerr, “Optimal static and self-adjusting parameter choices for the (1+λ) genetic algorithm,” Algorithmica, vol. 80, no. 5, pp. 1658–1709, 2018.

[13] B. Doerr and C. Doerr, “Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices,” 2018. [Online]. Available: arXiv:1804.05650.

[14] B. Doerr, C. Doerr, and F. Ebel, “From black-box complexity to designing new genetic algorithms,” Theor. Comput. Sci., vol. 567, pp. 87–104, Feb. 2015.

[15] B. Doerr, C. Doerr, and T. Kötzing, “Unknown solution length problems with no asymptotically optimal run time,” in Proc. Genet. Evol. Comput. Conf. (GECCO’17), 2017, pp. 1367–1374.

[16] B. Doerr, C. Doerr, and T. Kötzing, “Solving problems with unknown solution length at almost no extra cost,” Algorithmica, vol. 81, no. 2, pp. 703–748, 2019.

[17] B. Doerr, C. Doerr, and J. Lengler, “Self-adjusting mutation rates with provably optimal success rules,” in Proc. Genet. Evol. Comput. Conf. (GECCO’19), 2019, pp. 1479–1487.

[18] B. Doerr, C. Doerr, and J. Yang, “Optimal parameter choices via precise black-box analysis,” in Proc. Genet. Evol. Comput. Conf. (GECCO’16), 2018, pp. 1123–1130.

[19] B. Doerr, C. Gießen, C. Witt, and J. Yang, “The (1+λ) evolutionary algorithm with self-adjusting mutation rate,” in Proc. Genet. Evol. Comput. Conf. (GECCO’17), 2017, pp. 1351–1358.

[20] B. Doerr, T. Jansen, D. Sudholt, C. Winzen, and C. Zarges, “Mutation rate matters even when optimizing monotonous functions,” Evol. Comput., vol. 21, no. 1, pp. 1–27, Mar. 2013.

[21] B. Doerr, C. Witt, and J. Yang, “Runtime analysis for self-adaptive mutation rates,” in Proc. Genet. Evol. Comput. Conf. (GECCO’18), 2018, pp. 1475–1482.

[22] C. Doerr, “Complexity theory for discrete black-box optimization heuristics,” 2018. [Online]. Available: arXiv:1801.02037.

[23] C. Doerr and M. Wagner, “Simple on-the-fly parameter selection mechanisms for two classical discrete black-box optimization benchmark problems,” in Proc. Genet. Evol. Comput. Conf. (GECCO’18), 2018, pp. 943–950.

[24] S. Droste, T. Jansen, and I. Wegener, “Dynamic parameter control in simple evolutionary algorithms,” in Proc. Found. Genet. Algorithms (FOGA’01), 2001, pp. 275 – 294.

[25] A. E. Eiben, R. Hinterding, and Z. Michalewicz, “Parameter control in evolutionary algorithms,” IEEE Trans. Evol. Comput., vol. 3, no. 2, pp. 124–141, Jul. 1999.

[26] H. Einarsson et al., “The linear hidden subset problem for the (1 + 1) EA with scheduled and adaptive mutation rates,” in Proc. Genet. Evol. Comput. Conf. (GECCO’18), 2018, pp. 1491–1498.

[27] G. Karafotias, M. Hoogendoom, and A. E. Eiben, “Parameter control in evolutionary algorithms: Trends and challenges,” IEEE Trans. Evol. Comput., vol. 19, no. 2, pp. 167–187, Apr. 2015.

[28] I. Lassig and D. Sudholt, “Adaptive population models for offspring populations and parallel evolutionary algorithms,” in Proc. 11th Workshop Proc. Found. Genet. Algorithms (FOGA’11), 2011, pp. 181–192.

[29] P. K. Lehre, “Negative drift in populations,” in Proc. Int. Conf. Parallel Problem Solving Nat. (PPSN’10), 2010, pp. 244–253.

[30] P. K. Lehre, “Fitness-levels for non-elitist populations,” in Proc. 13th Annul. Genet. Evol. Comput. Conf. (GECCO’11), 2011, pp. 2075–2082.

[31] Per Kristian Lehre and E. Ozcan, “A runtime analysis of simple hyper-heuristics: To mix or not to mix operators,” in Proc. 12th Workshop Found. Genet. Algorithms (FOGA’13), 2013, pp. 97–104.

[32] P. K. Lehre and C. Witt, “Black-box search by unbiased variation,” Algorithmica, vol. 64, no. 4, pp. 623–642, 2012.

[33] P. K. Lehre and X. Yao, “On the impact of mutation-selection balance on the runtime of evolutionary algorithms,” IEEE Trans. Evol. Comput., vol. 16, no. 2, pp. 225–241, Apr. 2012.

[34] J. Lengler, “A general dichotomy of evolutionary algorithms on monotone functions,” in Proc. Int. Conf. Parallel Problem Solving Nat. (PPSN’18), 2018, pp. 3–15.

[35] K.-H. Liang, X. Yao, and C. Newton, “Adapting self-adaptive parameters in evolutionary algorithms,” Appl. Intell., vol. 15, no. 3, pp. 171–180, 2001.

[36] J. Smith and T. C. Fogarty, “Self-adaptation of mutation rates in a steady state genetic algorithm,” in Proc. IEEE Int. Conf. Evol. Comput., Nagoya, Japan, 1996, pp. 318–323.

[37] I. Wegener, “Simulated annealing beats metropolis in combinatorial optimization,” in Proc. Int. Colloquium Automata Languages Program. (ICALP’05), 2005, pp. 589–601.

Brendan Case received the M.Res. degree in natural computation from the University of Birmingham, Birmingham, U.K., under the supervision of Dr. P. K. Lehre in 2020. They are currently pursuing the Ph.D. degree in computer science with the University of Vermont, Burlington, VT, USA. Their current research interests involve modeling the role of insects in the spread of disease with a combination of theoretical and field-driven approaches. They were a recipient of the NRT QuEST Traineeship, which seeks to bring rising STEM doctoral students together to solve interdisciplinary problems across biology, ecology, and epidemiology.

Per Kristian Lehre received the M.Sc. and Ph.D. degrees in computer science from the Norwegian University of Science and Technology, Trondheim, Norway. He is a Senior Lecturer with the University of Birmingham, Birmingham, U.K. After finishing his Ph.D. in 2006, he held postdoctoral positions with the School of Computer Science, University of Birmingham and Technical University of Denmark, Lyngby, Denmark. From 2011 until 2017, he was a Lecturer with the School of Computer Science, University of Nottingham, Nottingham, U.K. His research interests are in theoretical aspects of nature-inspired search heuristics, in particular, runtime analysis of population-based evolutionary algorithms. Dr. Lehre was a recipient of the several best paper awards, including at GECCO in 2006, 2009, 2010, and 2013, ICSTW in 2008, and ISAAC in 2014. He is an Editorial Board Member of Evolutionary Computation, and an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION. He was the coordinator of the successful 2M euro EU-funded project SAGE which brought together the theory of evolutionary computation and population genetics.