Self-adaptation Can Help Evolutionary Algorithms Track Dynamic Optima

Lehre, Per Kristian; Qin, Xiaoyu

DOI:
10.1145/3583131.3590494

Document Version
Publisher's PDF, also known as Version of record

Citation for published version (Harvard):
Lehre, PK & Qin, X 2023, Self-adaptation Can Help Evolutionary Algorithms Track Dynamic Optima. in GECCO '23: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO: Genetic and Evolutionary Computation Conference, Association for Computing Machinery (ACM), pp. 1619–1627, GECCO: Genetic and Evolutionary Computation Conference, Lisbon, Portugal, 15/07/23. https://doi.org/10.1145/3583131.3590494

Self-adaptation Can Help Evolutionary Algorithms Track Dynamic Optima

Per Kristian Lehre*
University of Birmingham
Birmingham, United Kingdom
p.k.lehre@cs.bham.ac.uk

Xiaoyu Qin*
University of Birmingham
Birmingham, United Kingdom
xxq896@cs.bham.ac.uk

ABSTRACT

Real-world optimisation problems often involve dynamics, where objective functions may change over time. Previous studies have shown that evolutionary algorithms (EAs) can solve dynamic optimisation problems. Additionally, the use of diversity mechanisms, populations, and parallelisation can enhance the performance of EAs in dynamic environments if appropriate parameter settings are utilised. Self-adaptation, which encodes parameters in genotypes of individuals and allows them to evolve together with solutions, can help configure parameters of EAs. This parameter control mechanism has been proved to effectively handle a static problem with unknown structure. However, the benefit of self-adaptation on dynamic optimisation problems remains unknown. We consider a tracking dynamic optima problem, the so-called *Dynamic Substring Matching* (DSM) problem, which requires algorithms to successively track a sequence of structure-changing optima. Our analyses show that mutation-based EAs with a fixed mutation rate have a negligible chance of tracking these dynamic optima, while the self-adaptive EA tracks them with an overwhelmingly high probability. Furthermore, we provide a level-based theorem with tail bounds, which bounds the chance of the algorithm finding the current optima within a given evaluation budget. Overall, self-adaptation is promising for tracking dynamic optima.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**; • **Computing methodologies → Discrete space search**.

KEYWORDS

Evolutionary algorithms, self-adaptation, dynamic optimisation

ACM Reference Format:
Per Kristian Lehre and Xiaoyu Qin. 2023. Self-adaptation Can Help Evolutionary Algorithms Track Dynamic Optima. In *Genetic and Evolutionary Computation Conference (GECCO '23), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3583131.3590494

*Authors are listed in alphabetical order.

GECCO '23, July 15–19, 2023, Lisbon, Portugal

ACM ISBN 979-8-4007-0119-1/23/07.
https://doi.org/10.1145/3583131.3590494

1 INTRODUCTION

Evolutionary algorithms (EAs) can solve a wide variety of dynamic optimisation problems, where the objective function changes over time [22]. In this context, algorithms need to adapt and update solutions quickly since previously best solutions might no longer be good. Many rigorous analyses on EAs and other randomised search heuristics in dynamic environments have been published in the previous two decades. Table 1 summarises existing theoretical works on dynamic problems in evolutionary computation (EC). These studies can be categorised into three types.

The first type of research aims to evaluate the performances of algorithms on optimising a dynamic function with randomly changing optima. The criteria can be the number of evaluations when the algorithm first hits the current optima. For example, Droste [15], Droste et al. [16] proposed the dynamic OneMax, which computes the Hamming distance to the changed bitstring (the dynamic optimum). In this function, each generation generates a new optimum by bit-wisely flipping the last optimal bitstring with probability q. They demonstrated that the (1+1) EA could catch an optimal solution of this function in polynomial time if and only if $p = O\left(\log(n)/n^2\right)$. Kötzing et al. [23] extended the analysis from bitstring to larger alphabets, and showed that the number of values per dimension does not affect the performance of the (1+1) EA. Jansen and Wegener [21], for the first time, analysed a population-based algorithm but on a simple OneMax-variant problem in a two-dimensional lattice.

The second type of research is to analyse the runtime of algorithms on a dynamic function with the global and unique optimum. For example, the weights of bit positions in the noisy linear function [33] and the dynamic BinVal (DBV) function [30, 31] are randomly sampled from certain distributions before each generation, but the optimum is always 1^n. From the literature, the (1+1) EA can find 1^n in $O(n \log(n))$ on the noisy linear function [33], and the 2-tournament EAs which is a non-elitist population-based algorithm can optimise DBV in $O(n^2)$ runtimes [26]. Another examples are Magnitude [42] and Balance [42], where the magnitude and frequency of changes of functions can be set for theoretical analysis. Rohlfshagen et al. [42] found that the Balance function with the high frequency of change might be easier than that with low frequency for the (1+1) EA. Additionally, on Balance, Oliveto and Zarges [39] rigorously proved that the original (μ+1) EA fails, i.e., exponential runtime, if the frequency of change is sufficiently low; whereas the (μ+1) EA using a fitness-diversity mechanism can achieve a polynomial runtime regardless of the frequency.

The third type of study examines the ability to track dynamic optima. This type of problem has a sequence (path) of optima and the optimum is changed over time. The algorithms need to follow the

Table 1: Summary of theoretical studies of randomised search heuristics on dynamic optimisation

Type of Dynamics	Problem	Algorithm	Study
Optimising dynamic function with randomly changed optima	Dynamic OneMax [15, 16]	(1+1) EA	[15, 16]
	Generalised Dynamic OneMax [23]	(1+1) EA	[23]
	OneMax-variant in 2D lattice [20, 44]	(1+λ) EA	[20]
Optimising dynamic function with a global optimum	Magnitude [42]	(1+1) EA	[42]
	Balance [42]	(1+1) EA	[42]
		(μ+1) EA	[39]
		(μ+1) EAs with diversities	[39]
		(2+1) RLS with diversity	[39]
	Noisy linear function [33]	(1+1) EA	[33]
	Dynamic BinVal (DBV) [30, 31]	(μ+1) EA	[32]
		2-tournament EA	[26]
Tracking dynamic optima	Maze [24]	(1+1) EA	[24]
		MMAS	[24]
		(2+1) EA	[34]
		(1+λ) EA	[36]
		Parallel (1+1) EA	[36]
	Finite-alphabet Maze [34]	(μ+1) EA with diversity	[34]
		MMAS*	[34]
	Dynamic shortest path [35]	λ-MMAS	[35]
	(κ, ρ)-stable dynamic function, e.g., Moving Hamming Ball (MHB) [6]	(1+1) EA	[6]
		Non-elitist EAs	[6]
	Dynamic matching substring (DSM) (Def. 2.1)	Static mutation-based EAs	(Thm. 5.1)
		(μ, λ) self-adaptive EA	(Thm. 4.1)

path and find and hold the current optimal solutions before the next change; otherwise, they will get lost soon. For example, Kötzing and Molter [24] constructed the Maze function, in which the optimum is changed from 1^n to 0^n and the optimal bitstring oscillates between two bitstrings that have only one specific bit difference. For some constant $k > 0$, the optimal solution of the function changes every $(kn^3 \log(n))$-generations. Their analyses showed that a simple version of the Max-Min Ant System (MMAS), an ant colony optimisation (ACO) algorithm, could track and reach the final optimums, while the (1+1) EA loses track of optima. Additionally, using diversity mechanisms, population and parallelisation can help EAs in tracking the Maze function [34, 36]. A finite-alphabet variant of the Maze problem and the dynamic shortest path problem were proposed to illustrate the efficiency of the diversity mechanism and the MMAS [34, 35]. More recently, Dang et al. [6] introduced a class of dynamic optimisation problems to explain that the population is essential in dynamic environments. They proved that the (1+1) EA and the RLS lose the optimal solution region with constant probability at any generation, whereas the non-elitist population-based EAs remain within the optimal region for a long time with an overwhelmingly high probability [6].

The parameter settings, e.g., population size, mutation rate, are critical in EAs [10]. The above studies usually suggest parameter settings on specific dynamic optimisation problems. Self-adaptation is a strategy for parameter control, where the parameters are encoded within the genomes of individuals and evolve concurrently with the solutions . Self-adaptive EAs have been proven to be efficient on several optimisation problems. For a toy function, Doerr et al. [14] proved that the $(1, \lambda)$ self-adaptive EA optimises OneMax in $O(n \log(n))$ runtime. For an artificial two-peak function, Dang and Lehre [7] showed that the 2-tournament EA using two self-adapting mutation rates can escape the local optimum, while using neither a fixed mutation rate nor a uniformly selected mutation rate leads to failure. For an unknown structure function LeadingOnes_k, Case and Lehre [1] proved that the (μ, λ) self-adaptive EA can optimise it in $O(k^2)$ runtime, which is asymptotically optimal among all unary unbiased black-box algorithms. Note that the *structure* of the problem here means the number of relevant bit-positions. Recently, Lehre and Qin [27, 40] proposed the MOSA-EA which treats parameter control from the perspective of multi-objective optimisation. The runtime analysis showed that the MOSA-EA could efficiently escape from a local optimum with unknown sparsity. In an empirical study, the self-adaptation parameter control mechanism was shown to respond to changes of the fitness function, i.e., from OneMax to ZeroMax [43]. However, the benefit of self-adaptation on dynamic optimisation problems remains unknown.

In this paper, we explore whether self-adaptation can be beneficial in dynamic optimisation. We specifically examine a tracking dynamic optima problem with changing structure that require adjustable parameter settings. The structure, as previously discussed, refers to the number of relevant bits. This problem, the so-called *Dynamic Substring Matching* (DSM) problem, requires algorithms to successively find and hold the solutions that match a sequence of bit-flipping and length-varying target substrings (structure-changing optima) within specified evaluation budgets. We show that the EAs with any fixed mutation rate get lost with constant probability somewhere during tracking the DSM problem (Lemma 5.1), resulting in an exponentially small probability of achieving the final optimum (Theorem 5.1). Therefore, the variable mutation rates may be necessary for a successful track. The main contribution is the first rigorous study of self-adaptive parameter control mechanisms on dynamic optimisation. We demonstrate that the (μ, λ) self-adaptive EA can track every optimum in the DSM problem (Lemma 4.1) and reach the final optimum with an overwhelmingly high probability

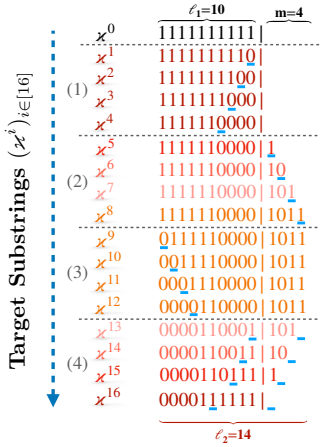

Figure 1: A sequence of target substrings in an example of DSM$^{\varkappa,m,\varepsilon,k}$ ($\varkappa = 1^{10}$, $n = 20$, $m = 4$), s.t. $\ell_1 = 10$ and $\ell_2 = 14$.

(Theorem 4.1). Another contribution is the level-based theorem with tail bounds (Theorem 3.1). To assess the capacity of the self-adaptive EA in tracking dynamic optima, it is necessary to determine a lower bound of the probability of achieving the current optimum within the specified evaluation budget. To address our requirements, we develop the level-based theorem with tail bounds.

2 PRELIMINARIES

In this section, we introduce the dynamic problem and the algorithms. We first define some notation for use later. The natural logarithm is denoted by $\ln(\cdot)$. For any $n \in \mathbb{N}$, we define $[n] := \{1, \cdots, n\}$ and $[0..n] := [n] \cap \{0\}$. For any $a, b \in \mathbb{N}$, where $a \leq b-1$, we define $[a..b] := [b] \setminus [a-1]$. Let $|x|$ be the number of bits in a bit-string x. For any $\ell \in \mathbb{N}$, we define $x_{1:m} := x_1 \cdots x_m$ be a substring of $x \in \{0,1\}^\ell$ consisting of the first m position bits, where $m \in [\ell]$. Let $\diamond$ donate bit-string concatenation. The Hamming distance is denoted by $\mathrm{H}(\cdot,\cdot)$. The Hamming-neighbours around a search point x are denoted by $\mathrm{N}(x) := \{y \mid \mathrm{H}(x,y) = 1\}$ where $x, y \in \{0,1\}^\ell$ and $\ell \in \mathbb{N}$. For any $n \in \mathbb{N}$, let $\mathcal{X} := \{0,1\}^n$ and let $f : \mathcal{X} \to \mathbb{R}$ be any pseudo-Boolean function. We define the matching function as $\mathrm{M}(x, \varkappa) := \begin{cases} 1 & \text{if } \mathrm{H}(x_{1:|\varkappa|}, \varkappa) = 0 \\ 0 & \text{otherwise} \end{cases}$, where $x \in \mathcal{X}$ is a bitstring and $\varkappa \in \{0,1\}^\ell$ is a substring where $\ell \in [n]$, then we say that a bitstring x matches a substring $\varkappa$ if $\mathrm{M}(x, \varkappa) = 1$.

2.1 Dynamic Substring Matching Problem

In dynamic environments, the number of pertinent bits may fluctuate (structure changing). This paper consider the DSM problem where involves this situation. Let $\varepsilon \in (0,1)$, $k > 0$, $\varkappa \in \{0,1\}^{\ell_1}$ where $\ell_1 \in [n-1]$, and $m \in [n-\ell_1]$ be the parameters of the DSM problem, then the DSM$^{\varkappa,m,\varepsilon,k}$ problem aims to match a sequence of bit-flipping and length-varying target substrings $(\varkappa^i)_{i\in[4m]}$ in a sequence of corresponding evaluation budgets $(\mathcal{T}_i)_{i\in[4m]}$. The length of target substrings varies between ℓ_1 and ℓ_2 where $\ell_2 = \ell_1 + m$ resulting in structure changing. The dynamics of the DSM problem is updated in four phases.

(1) for $i \in [m]$, the previous target substring $\varkappa^{i-1}$ and the current target substring $\varkappa^i$ are the same length but one bit different: $\varkappa^i$ generated by uniformly at random change one bit of $\varkappa^{i-1}$;
(2) for $i \in [m+1..2m]$, the target substrings are becoming longer: $\varkappa^i$ generated by appending one random bit in the end of $\varkappa^{i-1}$;
(3) for $i \in [2m+1..3m]$, similar to stage (1): $\varkappa^{i-1}$ and $\varkappa^i$ are the same length but one-bit different;
(4) for $i \in [3m+1..4m]$, the target substrings are becoming shorter: $\varkappa^{i-1}$ generated by removing the last bit of $\varkappa^{i-1}$ and uniformly at random flip one of the rest of bits.

Figure 1 illustrates a sequence of target substrings in an example DSM problem. The sequence of corresponding evaluation budgets $(\mathcal{T}_i)_{i\in[4m]}$ depends on the lengths of the target substrings, i.e., $kn^\varepsilon|\varkappa^i|$. The target substrings are changed after evaluation budgets run out. We call a period between two times of the target change a phase. We assume that the algorithms start from a starting substring $\varkappa^0$. The algorithms are required to find and hold solutions matching the current target substring before the target changes. The DSM problem is formally defined in Definition 2.1.

Definition 2.1. *Let $\varkappa$ be some starting target substring where $|\varkappa| =: \ell_1 \in [n-1]$, and m be a positive integers where $\ell_1 + m =: \ell_2 \leq n$. Let $(\varkappa^i)_{i\geq 0}$ be a sequence of target substrings generated by*

$$\varkappa^i := \begin{cases} \varkappa & \text{if } i = 0, \\ z, \text{ where } z \sim \mathrm{Unif}(\mathrm{N}(\varkappa^{i-1})) & \text{if } 1 \leq i \leq m, \\ \varkappa^{i-1} \diamond a, \text{ where } a \sim \mathrm{Unif}(\{0,1\}) & \text{if } m+1 \leq i \leq 2m, \\ z, \text{ where } z \sim \mathrm{Unif}(\mathrm{N}(\varkappa^{i-1})) & \text{if } 2m+1 \leq i \leq 3m, \\ z, \text{ where } z \sim \mathrm{Unif}\left(\mathrm{N}\left(\varkappa^{i-1}_{1:(|\varkappa^{i-1}|-1)}\right)\right) & \text{if } 3m+1 \leq i \leq 4m. \end{cases}$$

Let $(\mathcal{T}_i)_{i\in\mathbb{N}}$ be a sequence of the numbers of evaluation moving from $\varkappa^{i-1}$ to $\varkappa^i$ (evaluation budget for $\varkappa^i$) generated by $\mathcal{T}_i := kn^\varepsilon|\varkappa_{i+1}|$, where $\varepsilon \in (0,1)$ and $k > 0$ are some constants. For $t \in \mathbb{N}$, the dynamic substring matching (DSM) problem with the starting target substring $\varkappa$ is defined as:

$$DSM_t^{\varkappa,m,\varepsilon,k}(x) := \begin{cases} 2 & \text{if } \mathrm{M}(\varkappa(t), x) = 1, \\ 1 & \text{else if } \mathrm{M}(\varkappa'(t), x) = 1, \\ 0 & \text{otherwise}, \end{cases} \tag{1}$$

where $\varkappa(t) := \varkappa^i$, and $\varkappa'(t) := \varkappa^{i-1}$,

$$\text{for } i = \begin{cases} 1 & \text{if } t \leq \mathcal{T}_1, \\ 1 + \max\left\{j \mid \sum_{i=1}^{j} \mathcal{T}_i \leq t\right\} & \text{otherwise.} \end{cases}$$

Several methods to evaluate how well algorithms track dynamic optima have been proposed [6, 24]. Regarding the DSM problem, algorithms may fail to reach the final optimum if they lose track during some phases. Therefore, we define the criteria for tracking in Definition 2.2.

Definition 2.2. *A sequence of sets of solutions $(Q_t)_{t\in\mathbb{N}}$, where $Q_t \in \mathcal{X}^\lambda$ and $\lambda \in \mathbb{N}$, tracks the DSM$^{\varkappa,m,\varepsilon,k}$ if it begins with the initial set Q_0, where all solutions x satisfy $\mathrm{M}(x, \varkappa) = 1$, and at least one solution x' in $Q_{\bar{t}}$ satisfies $\mathrm{M}(x', \varkappa^{4m}) = 1$, where $\bar{t} = \lceil(\sum_{i=1}^{4m} \mathcal{T}_i)/\lambda\rceil$ denotes the end of the final phase.*

2.2 Algorithms

This subsection introduces the algorithms studied in this paper. This paper investigates the essentiality of using an adjustable mutation rate on a structure-changing dynamic optimisation problem. Thus, we consider the elitist and non-elitist versions of the static

Algorithm 1 (μ, λ) self-adaptive EA [1]

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 $b, p_{\text{inc}}, \epsilon \in (0, 1)$.
Require: Initial population $P_0 \in \mathcal{Y}^\lambda$.
1: **for** $t = 0, 1, 2, \cdots$ until termination condition met **do**
2: Sort P_t based on $P_t(1) \geq \cdots \geq P_t(\lambda)$ [†].
3: **for** $i = 1, \ldots, \lambda$ **do**
4: Set $(x, \chi/n) := P_t(I_t(i))$, $I_t(i) \sim \text{Unif}([\mu])$.
5: Set $\chi' := \begin{cases} \min\{A\chi, n/2\} \text{ with probability } p_{\text{inc}} \\ \max\{b\chi, \epsilon n\} \text{ otherwise.} \end{cases}$
6: Create x' by independently flipping each bit of x with probability χ'/n.
7: Set $P_{t+1}(i) := (x', \chi'/n)$.

mutation-based EAs (in Appendix A). There exist several comparative theoretical studies between elitism and non-elitism in EAs [3, 4, 8, 19]. Section 5 shows that mutation-only EAs with any fixed mutation rate, are not able to track optima in the DSM problem.

Algorithm 1 describes the (μ, λ) self-adaptive EA, which has demonstrated efficacy in solving unknown structure problems [1]. In the self-adaptive EA, each individual encodes its own mutation rate, in addition to its solution, defining the self-adaptive population $P_t \in \mathcal{Y}^\lambda$, where $\mathcal{Y} = \{0, 1\}^n \times (0, 1/2]$. Given a self-adaptive population $P_t = (x_i, \chi_i/n)_{i \in [\lambda]}$, we define the set of solutions of P_t as $Q_t := (x_i)_{i \in [\lambda]}$. In generation t, the population P_t is sorted first by fitness function f and then by individuals' mutation rates, preferring higher values for both (Line 2). Each individual is then produced through (μ, λ) selection (Line 4), mutation rate adaptation (Line 5), and solution mutation (Line 6). In the mutation rate adaptation, the selected individual inherits an increased mutation parameter $A\chi$ with probability p_{inc}, and a reduced mutation parameter $b\chi$ otherwise, where $A > 1$ and $b, p_{\text{inc}} \in (0, 1)$ are algorithm parameters. The solution of the individual is then mutated bitwisely with the new mutation rate. Thus, the sorting and mutation rate adaptation methods are crucial components of the algorithm, allowing the mutation rate to vary from ϵ to $1/2$, where $\epsilon > 0$ represents the minimum mutation rate.

3 LEVEL-BASED THEOREM (TAIL BOUNDS)

The level-based theorems [2, 4, 13] are general tools that provide an upper bound of the runtime of non-elitist algorithms which follow the scheme of Algorithm 2 with a population $P_t \in \mathcal{X}^\lambda$, where $\mathcal{X}^\lambda$ is the space of all populations of size λ. Assume that the search space $\mathcal{X}$ is partitioned into ordered disjoint subsets (called levels) $A_1, \cdots, A_m$. Let $A_{\geq j} := \cup_{k=j}^m A_k$ be the search points in level j and higher, and let D be some mapping from the set of all possible populations $\mathcal{X}^\lambda$ into the space of probability distributions of $\mathcal{X}$. Given any subset $A \subseteq \mathcal{X}$, we define $|P_t \cap A| := |\{i \mid P_t(i) \in A\}|$, i.e., the number of individuals in P_t that belong to A. To estimate an upper bound on the runtime using level-based theorems, three conditions must typically be satisfied: (G1) requires the probability of level "upgrading", i.e., creating an individual in higher levels; (G2) requires the probability of the number of individuals in higher levels "growing"; (G3) requires a sufficient population size. Specifically, in [4], a new level-based theorem has been proposed to address the issue of "deceptive" regions B that contains individuals with a higher selection probability but at a lower level. The theorem includes an additional condition (G0) that requires the probability of producing a "deceptive" individual to decrease if too many such individuals in the population.

In the theory of EC, besides the expected runtime, tail bounds can be other performance criteria of EAs, which indicate the probability of runtime within a given evaluation budget. Few theoretical tools were developed about tail bounds for EAs [12, 28, 29, 38]. In this paper, we are interested in the probability that an algorithm finds the current optimum in a specific evaluation budget. To address our requirements, we derive the level-based theorem with tail bounds (shown in Theorem 3.1). We assume that the search space $\mathcal{X}$ partitions into $B, A_0, A_1, \cdots, A_m$, then assume that the initial population P_0 contains sufficient individuals in level A_1 and the termination condition is to gain a population with enough individuals in A_m. Theorem 3.1 also consider a "deceptive region" B (condition (C0)). The assumption that there are not too many individuals in the region B holds for the initial population , i.e., $|P_0 \cap B| \leq \psi_0 \lambda$. Conditions (C1)-(C2) correspond to conditions (G1)-(G2) in the original level-based theorems [2, 4, 13], and no condition corresponds to condition (G3) since the tail bound is determined by the population size λ. Eventually, Theorem 3.1 gives the lower bound of the probability of runtime within $\eta\tau$ evaluation times by choosing η and λ. Compared to the original level-based theorems, the runtime from Theorem 3.1 is mainly one more multiplicative factor $\eta\lambda^{17/\delta^3}$, where η is used to tune the upper bound for $\Pr(T \leq \eta\tau)$. In order to fulfil the requirements into our scenario in Section 4, we refrain from employing the multiple restarts argument that was utilised in the proofs of the original level-based theorems [12, 28, 29, 38].

Theorem 3.1. *Let $(B, A_0, A_1, \cdots, A_m)$ be a partition of $\mathcal{X}$. Suppose there exist $z_1, \cdots, z_{m-1}$, $\delta \in (0, 1)$, and $\gamma_0, \psi_0 \in (0, 1)$, such that the following conditions hold for any population $P \in \mathcal{X}^\lambda$ in Algorithm 2,*
(C0) for all $\psi \in [\psi_0, 1]$, if $|P \cap B| \leq \psi\lambda$, then $\Pr_{y \sim D(P)}(y \in B) \leq (1 - \delta)\psi$,
(C1) for all $j \in [m-1]$, if $|P \cap B| \leq \psi_0\lambda$ and $|P \cap A_{\geq j}| \geq \gamma_0\lambda$, then $\Pr_{y \sim D(P)}(y \in A_{\geq j+1}) \geq z_j$,
(C2) for all $j \in [0..m-1]$, and $\gamma \in [1/\lambda, \gamma_0]$ if $|P \cap B| \leq \psi_0\lambda$ and $|P \cap A_{\geq j}| \geq \gamma_0\lambda$ and $|P \cap A_{\geq j+1}| \geq \gamma\lambda$, then $\Pr_{y \sim D(P)}(y \in A_{\geq j+1}) \geq (1 + \delta)\gamma$.
Let $T := \min\{t\lambda \mid |P_t \cap A_m| \geq \gamma_0\lambda \text{ and } |P_t \cap B| \leq \psi_0\lambda\}$, and assume the algorithm with population size $\lambda \in \mathbb{N}$ and an initial population P_0 satisfying $|P_0 \cap A_1| \geq \gamma_0\lambda$ and $|P_0 \cap B| \leq \psi_0\lambda$, then

$$\Pr(T \leq \eta\tau) > \left(1 - 2\eta\tau e^{-\delta^2 \min\{\psi_0, \gamma_0\}\lambda/4}\right)\left(1 - me^{-\eta\rho^{-\frac{\ln(\gamma_0)}{\ln(1+\delta/2)}-2}}\right)$$

for any $\eta \in \left(0, e^{\delta^2 \min\{\psi_0, \gamma_0\}\lambda/4}/\tau\right)$, where $\rho = \frac{e^{\delta^2/8}}{e^{\delta^2/8}-1}$ and $\tau := \lambda^{17/\delta^3}\left(\sum_{j=1}^{m-1} \frac{1}{z_j} + m\lambda\left(\frac{\ln(\gamma_0\lambda)}{\ln(1+\delta/2)} + 1\right)\right)$.

To apply Theorem 3.1, we need to satisfy conditions (C0)-(C2), which is the same as applying the original level-based theorems, then we need to choose η and λ to gain a desired upper bound for

[†] $(x, \chi) \geq (x', \chi') \Leftrightarrow f(x) > f(x') \vee (f(x) = f(x') \wedge \chi \geq \chi')$,

Algorithm 2 Population-based Algorithm

Require: Finite state space $\mathcal{X}$ and population size $\lambda \in \mathbb{N}$; Map D from $\mathcal{X}^\lambda$ to the space of probability distributions over $\mathcal{X}$
Require: Initial population $P_0 \in \mathcal{X}^\lambda$
1: **for** $t = 0, 1, 2, \cdots$ until termination condition met **do**
2: **for** $i = 1$ to λ **do**
3: Sample $P_{t+1}(i) \sim D(P_t)$

$\Pr(T \le \eta\tau)$. For example, if conditions (C0)-(C2) hold for some constants $\delta, \psi_0, \gamma_0 \in (0,1)$, $z_j := \Omega(1/n)$ and $m \in \text{poly}(n)$ by an instantiated algorithm with population size λ, then the upper bound for $\Pr(T \le \eta\tau)$ is $\left(1 - \eta\tau e^{-\Omega(\lambda)}\right)\left(1 - me^{-\Omega(\eta)}\right)$. If choosing $\lambda = \eta = n^\varepsilon$ for a constant $\varepsilon \in (0,1)$, then $\Pr(T \le \eta\tau) = 1 - e^{-\Omega(n^\varepsilon)}$. In Section 4, we will use Theorem 3.1 to prove that the (μ, λ) self-adaptive EA can generate enough individuals matching the current target substring in evaluation budgets from the previous target substring with an overwhelmingly high probability. In comparison to the original level-based theorems, Theorem 3.1 typically necessitates a larger population size, e.g., cn^ε, in order to achieve a sufficiently large tail probability, i.e., $1 - e^{-\Omega(n^\varepsilon)}$, where constants $c, \varepsilon > 0$.

Now we informally explain the proof idea. Pessimistically, the algorithm gradually increases its level from A_1 to A_m. There are m steps from A_1 to A_m. In each step, the algorithm needs to generate a higher-level individual, and then accumulate such individuals until the population contains a sufficient number of such individuals. We estimate the lower bound of the successful probability of each step in certain evaluation times. Next, we consider the probability of the "failure events", i.e., the algorithm goes back to the previous level, or produces too many "bad" (B region) individuals during the algorithm running. Eventually, we compute the lower bound of the probability that every step is completed without "failure events".

Proof of Theorem 3.1. We first define some notation for later use. For any level $j \in [m]$ and $t \in \mathbb{N}_0$, let the random variable $X_t^{(j)} := |P_t \cap A_{\ge j}|$ denote the number of individuals in levels $A_{\ge j}$ at time t. Let the random variable $Y_t := |P_t \cap B|$ denote the number of individuals in the region B at time t. The level J_t of population P_t at time t is defined as $J_t := \max\{j \in [m] : X_t^j \ge \gamma_0\lambda\}$. We say the algorithm *upgrades its level* in h generations if $J_{t+h} \ge J_t + 1$.

We now estimate the probability that no "failure events" occur. More precisely, "failure events" include less than $\gamma_0\lambda$ individuals of population P_t in level $A_{\ge J_{t-1}}$ and more than $\psi_0\lambda$ individuals of population P_t in the region B. Given $t \ge 1$, let $\mathcal{E}_t$ be the event that $J_t \ge J_{t-1}$ and $Y_t \le \psi_0\lambda$, and define $\hat{\mathcal{E}}_t := \mathcal{E}_1 \wedge \cdots \wedge \mathcal{E}_t$. By condition (C2), the random variable $(X_t^{(J_{t-1})} \mid \hat{\mathcal{E}}_{t-1})$ stochastically dominates $Z \sim \text{Bin}(\lambda, (1+\delta)\gamma_0)$. By condition (C0), the random variable $(Y_t \mid \hat{\mathcal{E}}_{t-1})$ is stochastically dominated by $Z' \sim \text{Bin}(\lambda, (1-\delta)\psi_0)$. Therefore, using a union bound, we have

$$\begin{aligned}
\Pr\left(\hat{\mathcal{E}}_t \mid \hat{\mathcal{E}}_{t-1}\right) &\ge 1 - \Pr\left(J_t < J_{t-1} \mid \hat{\mathcal{E}}_{t-1}\right) - \Pr\left(Y_t > \psi_0\lambda \mid \hat{\mathcal{E}}_{t-1}\right) \\
&= 1 - \Pr\left(X_t^{(J_{t-1})} < \gamma_0\lambda \mid \hat{\mathcal{E}}_{t-1}\right) - \Pr\left(Y_t > \psi_0\lambda \mid \hat{\mathcal{E}}_{t-1}\right) \\
&\ge 1 - \Pr\left(Z < \gamma_0\lambda\right) - \Pr\left(Z' > \psi_0\lambda\right).
\end{aligned}$$

We then use Chernoff bounds to estimate the upper bounds of $\Pr(Z < \gamma_0\lambda)$ and $\Pr(Z' > \psi_0\lambda)$:

$$\begin{aligned}
\Pr\left(Z < \gamma_0\lambda\right) &= \Pr\left(Z < \gamma_0\lambda(1+\delta)\left(1 - \frac{\delta}{1+\delta}\right)\right) \\
&= \Pr\left(Z < E[Z]\left(1 - \frac{\delta}{1+\delta}\right)\right) \\
&< e^{-\frac{\delta^2}{(1+\delta)^2}E[Z]/2} = e^{-\frac{\delta^2\gamma_0\lambda}{2(1+\delta)}} < e^{-\delta^2\gamma_0\lambda/4}, \qquad \text{and}
\end{aligned}$$

$$\begin{aligned}
\Pr\left(Z' > \psi_0\lambda\right) &= \Pr\left(Z' > \psi_0\lambda(1-\delta)\left(1 + \frac{\delta}{1-\delta}\right)\right) \\
&= \Pr\left(Z' > E[Z']\left(1 + \frac{\delta}{1-\delta}\right)\right) < e^{-\frac{\delta^2}{(1-\delta)^2}E[Z']/\left(2+\frac{\delta}{1-\delta}\right)} \\
&= e^{-\frac{\delta^2\psi_0\lambda}{1-\delta}/\left(2+\frac{\delta}{1-\delta}\right)} < e^{-\frac{\delta^2\psi_0\lambda}{2-\delta}} < e^{-\delta^2\psi_0\lambda/2}. \qquad \text{Thus,}
\end{aligned}$$

$$\begin{aligned}
\Pr\left(\hat{\mathcal{E}}_t \mid \hat{\mathcal{E}}_{t-1}\right) &> 1 - e^{-\delta^2\gamma_0\lambda/4} - e^{-\delta^2\psi_0\lambda/2} \\
&\ge 1 - 2e^{-\delta^2\min\{\gamma_0,\psi_0\}\lambda/4}. \qquad (2)
\end{aligned}$$

Next, we consider the number of generations to increase the level of the algorithm. We note that, by condition (C2), the random variable $(X_{t+1}^{(J_t+1)} \mid X_t^{(J_t+1)} \ge \gamma\lambda, \hat{\mathcal{E}}_t)$ stochastically dominates $Z'' \sim \text{Bin}(\lambda, (1+\delta)\min\{\gamma, \gamma_0\})$ for any $\gamma\lambda \ge 1$. Therefore, if $1 \le \gamma\lambda \le \gamma_0\lambda$, we have

$$\begin{aligned}
&\Pr\left(X_{t+1}^{(J_t+1)} \ge (1+\delta/2)\gamma\lambda \mid X_t^{(J_t+1)} \ge \gamma\lambda, \hat{\mathcal{E}}_t\right) \\
&\ge \Pr\left(Z'' \ge (1+\delta/2)\gamma\lambda\right) = 1 - \Pr\left(Z'' < (1+\delta/2)\gamma\lambda\right) \\
&= 1 - \Pr\left(Z'' < (1+\delta)\gamma\lambda\left(1 - \frac{\delta}{2(1+\delta)}\right)\right) \\
&= 1 - \Pr\left(Z'' < E[Z'']\left(1 - \frac{\delta}{2(1+\delta)}\right)\right), \text{ then by a Chernoff bound,} \\
&> 1 - e^{-\frac{\delta^2}{4(1+\delta)^2}E[Z'']} = 1 - e^{-\frac{\delta^2\gamma\lambda}{4(1+\delta)^2}} > 1 - e^{-\delta^2\gamma\lambda/8} \ge 1 - e^{-\delta^2/8}.
\end{aligned}$$

Then we define $h := \lceil \log_{1+\delta/2}(\gamma_0\lambda) \rceil \le \ln(\gamma_0\lambda)/\ln(1+\delta/2) + 1$. Informally, h is the number of consecutive "growing" steps required for the algorithm to upgrade its level. If there exists an individual of the population P_t in level $A_{\ge J_t+1}$, then the probability of the algorithm upgrading its level in h generations is at least

$$\begin{aligned}
&\Pr\left(X_{t+h}^{(J_t+1)} \ge \gamma_0\lambda \mid X_t^{(J_t+1)} \ge 1, \hat{\mathcal{E}}_{t+h-1}\right) \\
&\ge \prod_{i=1}^{h} \Pr\left(X_{t+i}^{(J_t+1)} \ge (1+\delta/2)^i \mid X_{t+i-1}^{(J_t+1)} \ge (1+\delta/2)^{i-1}, \hat{\mathcal{E}}_{t+i-1}\right) \\
&\ge \prod_{i=1}^{h} \left(1 - e^{-\delta^2/8}\right) = \left(1 - e^{-\delta^2/8}\right)^h = \rho^{-h}
\end{aligned}$$

where we define $\rho := \frac{e^{\delta^2/8}}{e^{\delta^2/8}-1} > 1/(1-1/e)$ by $\delta \in (0,1)$.

Then, we note that by condition (C1) and following by $(1 + x/n)^n \le e^x$ for $n \ge 1$, $|x| \le n$, for any t,

$$\begin{aligned}
&\Pr\left(\exists s_j \le \lceil 1/(z_j\lambda) \rceil : X_{t+s_j}^{(J_t+1)} \ge 1 \mid \hat{\mathcal{E}}_{t+s_j-1}\right) \\
&\ge 1 - \left(1 - z_j\right)^{\lambda\lceil 1/(z_j\lambda)\rceil} \ge 1 - 1/e > \rho^{-1}.
\end{aligned}$$

Thus, if we define $\tau(j) := \lceil 1/(z_j\lambda)\rceil + h$, we obtain the following lower bound for the probability of the algorithm upgrading its level in $\tau(j)$ generations,

$$\begin{aligned}&\Pr\left(J_{t+\tau(J_t)} \geq J_t + 1 \mid \hat{\mathcal{E}}_{t+\tau(J_t)-1}\right)\\ &\geq \Pr\left(\exists s_j \leq \lceil 1/(z_j\lambda)\rceil : X^{(J_t+1)}_{t+s_j} \geq 1 \mid \hat{\mathcal{E}}_{t+s_j-1}\right)\\ &\quad\cdot \Pr\left(X^{(J_t+1)}_{t+\tau(J_t)} \geq \gamma_0\lambda \mid X^{J_t+1}_t \geq 1, \hat{\mathcal{E}}_{t+\tau(J_t)-1}\right) \geq \rho^{-h-1}.\end{aligned}$$

In particular, we have the following upper bound for the probability that the algorithm does not upgrade its level in $\eta\lambda^{17/\delta^3}\tau(J_t)$ generations,

$$\Pr\left(J_{t+\eta\lambda^{17/\delta^3}\tau(J_t)} \leq J_t \mid \hat{\mathcal{E}}_{t+\eta\lambda^{17/\delta^3}\tau(J_t)}\right) \leq \left(1-\rho^{-h-1}\right)^{\eta\lambda^{17/\delta^3}}.$$

Define $\hat{\tau}(j) := \eta\lambda^{17/\delta^3}\sum_{i=1}^{j}\tau(i)$, and note that $\tau \geq \hat{\tau}(m-1)\lambda/\eta$. We then have

$$\begin{aligned}&\Pr\left(T \leq \eta\tau \mid \hat{\mathcal{E}}_{\eta\tau}\right) \geq \Pr\left(\bigcap_{j=1}^{m-1}\left(J_{\hat{\tau}(j)} \geq j+1\right) \mid \hat{\mathcal{E}}_{\eta\tau}\right)\\ &= 1-\Pr\left(\bigcup_{j=1}^{m-1}\left(J_{\hat{\tau}(j)} < j+1\right) \mid \hat{\mathcal{E}}_{\eta\tau}\right), \text{ by a union bound,}\\ &\geq 1-\sum_{j=1}^{m-1}\Pr\left(J_{\hat{\tau}(j)} < j+1 \mid J_{\hat{\tau}(j-1)} \geq j, \hat{\mathcal{E}}_{\eta\tau}\right)\\ &\geq 1-\sum_{j=1}^{m-1}\Pr\left(J_{\hat{\tau}(j)} < j+1 \mid J_{\hat{\tau}(j-1)} = j, \hat{\mathcal{E}}_{\eta\tau}\right)\\ &= 1-\sum_{j=1}^{m-1}\Pr\left(J_{\hat{\tau}(j-1)+\eta\lambda^{17/\delta^3}\tau(j)} \leq j \mid J_{\hat{\tau}(j-1)} = j, \hat{\mathcal{E}}_{\eta\tau}\right)\\ &\geq 1-m\left(1-\rho^{-h-1}\right)^{\eta\lambda^{17/\delta^3}}.\end{aligned}$$

Recall Eq. (2), we know that, using the Bernoulli's inequality $(1+x)^r \geq 1+rx$ for $-1 < x < 0$ and $r \in \mathbb{N}_{\neq}$,

$$\begin{aligned}&\Pr\left(\hat{\mathcal{E}}_{\eta\tau}\right) \geq \prod_{t=1}^{\eta\tau}\Pr\left(\hat{\mathcal{E}}_t \mid \hat{\mathcal{E}}_{t-1}\right) \geq \left(1-2e^{-\delta^2\min\{\gamma_0,\psi_0\}\lambda/4}\right)^{\eta\tau}\\ &\geq 1-2\eta\tau e^{-\delta^2\min\{\gamma_0,\psi_0\}\lambda/4}. \qquad \text{Hence,}\\ &\Pr(T \leq \eta\tau) \geq \Pr\left(T \leq \eta\tau \mid \hat{\mathcal{E}}_{\eta\tau}\right)\cdot\Pr\left(\hat{\mathcal{E}}_{\eta\tau}\right)\\ &\geq \left(1-m\left(1-\rho^{-h-1}\right)^{\eta\lambda^{17/\delta^3}}\right)\cdot\left(1-2\eta\tau e^{-\delta^2\min\{\gamma_0,\psi_0\}\lambda/4}\right).\end{aligned}$$

Since

$$\begin{aligned}&\rho^{-h-1} \geq \rho^{-\frac{\ln(\gamma_0\lambda)}{\ln(1+\delta/2)}-2} \geq \rho^{-\frac{\ln(\gamma_0)}{\ln(1+\delta/2)}-\frac{\ln(\lambda)}{\ln(1+\delta/2)}-2}\\ &= \rho^{-\frac{\ln(\gamma_0)}{\ln(1+\delta/2)}-2}\lambda^{-\frac{1}{\log_\rho(e)\ln(1+\delta/2)}} = \rho^{-\frac{\ln(\gamma_0)}{\ln(1+\delta/2)}-2}\lambda^{-\frac{\delta^2/8-\ln(e^{\delta^2/8}-1)}{\ln(1+\delta/2)}}\\ &\geq \rho^{-\frac{\ln(\gamma_0)}{\ln(1+\delta/2)}-2}\lambda^{-\frac{\delta^2/8+8/(2\delta^2)}{\delta/2-\delta^2/4}} \geq \rho^{-\frac{\ln(\gamma_0)}{\ln(1+\delta/2)}-2}\lambda^{-17/\delta^3},\end{aligned}$$

then by $(1+x/n)^n \leq e^x$ for $n \geq 1, |x| \leq n$, $\Pr(T \leq \eta\tau) \geq \left(1-me^{-\eta\rho^{-\frac{\ln(\gamma_0)}{\ln(1+\delta/2)}-2}}\right)\left(1-2\eta\tau e^{-\delta^2\min\{\gamma_0,\psi_0\}\lambda/4}\right)$. □

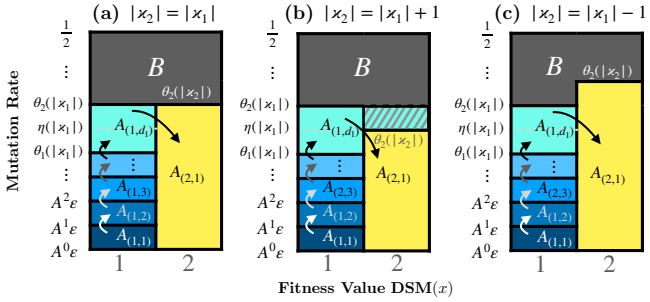

Figure 2: Level partitions for three cases in the proof of Lemma 4.1. Note that level A_0 is omitted in the subfigures.

4 SELF-ADAPTIVE EA TRACKS

We now show that the (μ, λ) self-adaptive EA can track the DSM problem. To prove this, we first derive Lemma 4.1 which shows that the algorithm obtains a population with a sufficient number of individuals matching the current target substring within the evaluation budget with an overwhelming probability in a phase if it successfully tracks in the previous phases. Then, Theorem 4.1 via Lemma 4.1 states the efficiency of the (μ, λ) self-adaptive EA on solving the DSM problem.

Case and Lehre [1] proved that the runtime of the (μ, λ) self-adaptive EA on optimising $\textsc{LeadingOnes}_k(x) := \sum_{i=1}^{k}\prod_{j=1}^{i}x_j$ is $O(k^2)$, where k is unknown for the algorithm. They divided the search space $\mathcal{Y}$ into a two-dimensional level partition, fitness levels and mutation rate sub-levels. Informally, to estimate the runtime, they counted the number of generations to increase the mutation rate until it is sufficiently high, and then counted the number of generations until the solution improved. It is well-known that too high mutation rates might fail non-elitist EAs. More precisely, there exist *error thresholds* for non-elitist EAs, which lead to exponential runtime on any function with a unique optimum if the mutation rate exceeds the threshold [25]. Since the (μ, λ) self-adaptive EA applies a non-elitist selection mechanism, they also defined a "bad" region B which contains individuals with too high mutation rates. They assumed that the algorithms restart from the first level if there are too many individuals in the B region. However, algorithms cannot be allowed to restart, since algorithms should keep the previous optimal solution in our scenario. Therefore, it is essential to limit the number of individuals in the B region in every generation to avoid losing track while tracking the dynamic function.

Lemma 4.1 provides the probability of obtaining sufficient individuals matching the current target substring $\varkappa_2$ in a phase of the $\text{DSM}^{\varkappa,m,\varepsilon,k}$ function within a given evaluation budget $\mathcal{T}_2$, by assuming that enough individuals match the previous target substring $\varkappa_1$ at the beginning of the phase. Our proof idea is similar to [1]. We first define the level partition in a phase of the DSM problem. We partition the search space $\mathcal{Y}$ into three fitness levels: $A_2 =: \{x \mid \text{M}(x, \varkappa_2) = 1\}$, $A_1 =: \{x \mid \text{M}(x, \varkappa_1) = 1\}\backslash A_2$, and $A_0 =: \mathcal{X}\backslash(A_2 \cup A_1)$. We further divide A_1 into mutation rate sub-levels. Finally, we use the three threshold values θ_2, η, θ_1 based on the lengths of $\varkappa_1, \varkappa_2$ to describe mutation rate sub-levels, which will be defined later. Informally, a mutation rate between θ_1 and θ_2 is suitable for increasing fitness level, i.e., from A_1 to A_2. Let $d_1 := \min\left\{\ell \in \mathbb{N} \mid \epsilon A^\ell \geq \theta_1(|\varkappa_1|)\right\}$ be the number of sub-levels of

A_1, and let A_2 only contain one sub-level $A_{(2,1)}$ with the number of sub-levels $d_2 := 1$. We cannot allow too many individuals with too high mutation rates, thus a "bad" region B is defined which contains all search point above a threshold value θ_2. Additionally, the important assumption in Lemma 4.1 is $|P \cap B| \leq \psi_0 \lambda$ for the population at beginning of the phase. In overall,

$$A_{(1,\ell)} := A_1 \times \left[A^{\ell-1}\epsilon, \min\left(A^{\ell}\epsilon, \theta_1(|\varkappa_1|)\right)\right) \text{ for } \ell \in [d_1 - 1]; \quad (3)$$

if $|\varkappa_2| = |\varkappa_1| + 1$, we define

$$A_{(1,d_1)} := A_1 \times \left[\theta_1(|\varkappa_1|), \min\left(\frac{1}{2}, \theta_2(|\varkappa_1|)\right)\right] \cup A_2 \times \left(\min\left(\frac{1}{2}, \theta_2(|\varkappa_2|)\right), \min\left(\frac{1}{2}, \theta_2(|\varkappa_1|)\right)\right], \quad (4)$$

if $|\varkappa_2| = |\varkappa_1|$ or $|\varkappa_2| = |\varkappa_1| - 1$, we define

$$A_{(1,d_1)} := A_1 \times \left[\theta_1(|\varkappa_1|), \min\left(\frac{1}{2}, \theta_2(|\varkappa_1|)\right)\right], \quad (5)$$

$$A_{(2,1)} := A_2 \times \left[\epsilon, \min\left(A^{\ell}\epsilon, \theta_2(|\varkappa_2|)\right)\right); \quad (6)$$

$$B := \cup_{i=1}^{2} A_i \times \left(\theta_2(|\varkappa^i|), \frac{1}{2}\right]. \quad (7)$$

Note that the sub-levels definition depends on the lengths of substrings $\varkappa_1$ and $\varkappa_2$. Figure 2 illustrates the three cases of the definition of the level partitions: (a) $|\varkappa_2| = |\varkappa_1|$ corresponding stages (1) and (3), (b) $|\varkappa_2| = |\varkappa_1| + 1$ corresponding stages (2), and (c) $|\varkappa_2| = |\varkappa_1| - 1$ corresponding stages (4).

We apply the same threshold values defined in [1]: for $j \geq 1$, let

$$\eta(j) := \frac{1}{2A}\left(1 - \left(\frac{1+\delta}{\alpha_0 p_{\text{inc}}}\right)^{1/j}\right), \text{ where } \alpha_0 := \lambda/\mu, \quad (8)$$

$$\theta_1(j) := b\eta(j) \quad (9)$$

$$\theta_2(j) := 1 - q^{1/j}, \text{ where} \quad (10)$$

$$q := \frac{1-\zeta}{\alpha_0}, r_0 := \frac{1+\delta}{\alpha_0(1-p_{\text{inc}})}, \text{ and } \zeta := 1 - \alpha_0(r_0)^{1+\sqrt{r_0}}. \quad (11)$$

Lemma 4.1. *Let $\varepsilon, a, \beta \in (0,1)$ and $k > 0$ be some constants satisfying $1/34 \leq a < \beta \leq 1$. Let $\varkappa \in \{0,1\}^{\ell_1}$ be some starting target substring where $\ell_1 \in \Theta(n^a)$, and $m \in \Theta(n^\beta)$. Let $\varkappa_1$ and $\varkappa_2$ be any two neighbouring substrings of the sequence of target substrings $(\varkappa^i)_{i\geq 0}$ in the DSM$^{\varkappa,m,\varepsilon,k}$. Let $\mathcal{T}_2$ be the evaluation budget for $\varkappa_2$ in the DSM$^{\varkappa,m,\varepsilon,k}$. Suppose that the initial population P_0 in Algorithm 1 satisfies $|P_0 \cap A_{\geq(1,1)}| \geq \gamma_0\lambda$ and $|P_0 \cap B| \leq \psi_0\lambda$, where γ_0 and ψ_0 are constants in $(0,1)$. Then, there exist constants δ and ξ in $(0,1)$ such that the probability of Algorithm 1 with parameters $\lambda, \mu = \Theta(n^\xi)$, $\lambda/\mu = \alpha_0 \geq 4$, $A > 1$, $0 < b < 1/(1+\sqrt{1/\alpha_0(1-p_{\text{inc}})})$, $(1+\delta)/\alpha_0 < p_{\text{inc}} < 2/5$, $\epsilon = b'/n$ for any constant $b' > 0$, where A and b are constants, obtaining a population P_t with $|P_t \cap A_{\geq(2,1)}| \geq \gamma_0\lambda$ and $|P_t \cap B| \leq \psi_0\lambda$, where $t \leq \mathcal{T}_2$, is $1 - e^{-\Omega(n^\xi)}$.*

We apply the level-based theorem with tail bounds (Theorem 3.1) to prove Lemma 4.1. We use serval lemmas to break down the proof of Lemma 4.1. Lemma 4.2 implies the probability of selecting individuals in a self-adaptive population. Lemmas 4.3-4.5 correspond to conditions (C0), (C2) and (C1) in Theorem 3.1, respectively. Lemma 4.4 looks similar to Lemma 4 in [1], but their proofs are different since the definitions of the level partitions are different. The proofs of Lemmas 4.2-4.5 are in Appendices C.1-C.4.

Lemma 4.2. *(Selection probability) If at least $\gamma\lambda$ individuals of a population P_t of Algorithm 1 with $\lambda/\mu = \alpha_0$ are in $A_{\geq(i,\ell)}$ and at most $\psi_0\lambda$ individuals in B satisfying $\psi_0 + \gamma \leq 1/\alpha_0$ for $\psi_0, \gamma \in (0,1]$, then for all $i \in \{1,2\}$ and $\ell \in [d_i]$, a parent $(x, \chi/n)$ selected in step 4 satisfies,* $\Pr\left((x,\chi/n) \in A_{\geq(i,\ell)}\right) \leq \gamma\alpha_0$.

Lemma 4.3. *(Condition (C0)) Assume that the parameters A, b and p_{inc} satisfy the constraints in Lemma 4.1. Then there exists a constant $\delta \in (0,1)$ such that if Algorithm 1 in step 4 selects a parent $(x,\chi/n)$, then the offspring $(x',\chi'/n)$ created in steps 5-6 satisfies,* $\Pr((x',\chi'/n) \in B) \leq \frac{1-\zeta}{\alpha_0}$.

Lemma 4.4. *(Condition (C2)) Assume that the parameters A, b and p_{inc} satisfy the constraints in Lemma 4.1. Then there exists a constant $\delta \in (0, 1/10)$ such that for all $j \in \{1,2\}$ and $\ell \in [d_j]$, if Algorithm 1 in step 4 selects a parent $(x,\chi/n) \in A_{\geq(i,\ell)}$, then the offspring $(x',\chi'/n)$ created in steps 5-6 satisfies,* $\Pr\left((x',\chi'/n) \in A_{\geq(i,\ell)}\right) \geq \frac{1+\delta}{\alpha_0}$.

Lemma 4.5. *(Condition (C1)) Assume that the parameters A, b and p_{inc} satisfy the constraints in Lemma 4.1. Then there exists a constant $\delta \in (0, 1/10)$ such that if Algorithm 1 in step 4 selects a parent $(x,\chi/n) \in A_{\geq(1,\ell)}$ for $\ell \in [d_j - 1]$, then the offspring $(x',\chi'/n)$ created in step 5-6 satisfies,* $\Pr\left((x',\chi'/n) \in A_{\geq(1,\ell+1)}\right) \geq \frac{1+\delta}{\alpha_0}$, *and if the selected parent $(x,\chi/n) \in A_{(1,d_1)}$, then the offspring $(x',\chi'/n)$ satisfies,* $\Pr\left((x',\chi'/n) \in A_{\geq(2,1)}\right) = \Omega(1/|\varkappa_2|)$.

Proof of Lemma 4.1. With the assumptions on P_0, we can apply Theorem 3.1 to prove Lemma 4.1. We first list some values from the DSM$^{\varkappa,m,\varepsilon,k}$ for later use: $\mathcal{T}_2 := kn^\varepsilon|\varkappa_2|$ for some constant $k > 0$, $||\varkappa_1| - |\varkappa_2|| \leq 1$, $|\varkappa_1|, |\varkappa_2| \in \Omega(n^a) \cap O(n^\beta)$. We also define some values $\psi_0 := (1-\zeta/2)/\alpha_0$, $\gamma_0 := \zeta/2$, $\xi := \delta^3\varepsilon/34$, $\eta := n^{\varepsilon/2}$. We then know $\delta, b, p_{\text{inc}}, \gamma_0, \psi_0, \zeta, \varepsilon, a, \beta, \xi \in (0,1)$ and $\alpha_0 \geq 4$ are constants. We use the level partition defined in Eq. (3)-(7).

Condition (C0) implies not too many individuals in the B region, i.e. at most $\psi_0\lambda$ individuals, which is verified by Lemma 4.3, such that $\Pr((x',\chi') \in B) < \frac{1-\zeta}{\alpha_0} \leq (1-\delta)\frac{1-\zeta/2}{\alpha_0} \leq (1-\delta)\psi$ for some constant $\delta \leq \frac{\zeta}{2-\zeta}$.

To verify condition (C2), we must estimate the probability of producing an offspring in $A_{\geq(i,\ell)}$ for $i \in \{1,2\}$ and $\ell \in [d_i]$, assuming at least $\gamma\lambda$ individuals in $A_{\geq(i,\ell)}$ for any $\gamma \in (0,\gamma_0]$ and at most $\psi_0\lambda$ individuals in B. To produce such offspring, it is necessary to first select a parent (x,χ) in $A_{\geq(i,\ell)}$, and to create an offspring (x',χ') in $A_{\geq(i,\ell)}$. The probability of selecting a parent $(x,\chi) \in A_{\geq(i,\ell)}$ is at least $\gamma\lambda/\mu = \gamma\alpha_0$ by Lemma 4.2. Then the probability to create an offspring $(x',\chi') \in A_{\geq(i,\ell)}$ is at least $(1+\delta)/\alpha_0$ by Lemma 4.4. Thus, condition (C2) is satisfied by $\gamma\alpha_0(1+\delta)/\alpha_0 = \gamma(1+\delta)$.

To verify condition (C1), we need to estimate the probability of producing an offspring in a level higher than $A_{\geq(1,\ell)}$ for $\ell \in [d_i]$, if at least $\gamma_0\lambda$ individuals in $A_{\geq(1,\ell)}$. We only consider the case that the parent (x,χ) is selected from $A_{\geq(1,\ell)}$, in which its probability is $\gamma_0\alpha_0$ from Lemma 4.2. Then by Lemma 4.5, the probability of producing an offspring in $A_{(1,\ell+1)}$ is $(1+\delta)\gamma_0 =: z_{(1,\ell)}$ for all $\ell \in$

$[d_1 - 1]$, and the probability of producing an offspring in $A_{(2,1)}$ is $z_{(1,d_1)} = \Omega(1/|\varkappa_2|)$ for $\ell = 1$.

Now, we can compute the lower bound of probability of runtime $T \leq \mathcal{T}_2$. By Theorem 3.1,

$$\tau = \eta\lambda^{17/\delta^3}\left(\frac{1}{z_{(1,d_1)}} + ((1+\delta)\gamma_0)(d_1 - 1) + d_1\lambda\left(\frac{\ln(\gamma_0\lambda)}{\ln(1+\delta/2)} + 1\right)\right)$$

since $\delta, \gamma_0 \in (0,1)$ are constants and $d_1 \in \log(n)$,

$$= O\left(\eta\lambda^{17/\delta^3}(|\varkappa_2| + \lambda\log(n)\log(\log(n)))\right)$$

since $\lambda \in \Theta(n^{\xi})$, $|\varkappa_2| \in \Omega(n^a)$ and $\xi = \delta^3\varepsilon/34 < 1/34 \leq a$,

$$= O\left(\eta\lambda^{17/\delta^3}|\varkappa_2|\right) = O\left(n^{\varepsilon}|\varkappa_2|\right)$$

if we let $\eta = \Theta\left(n^{\varepsilon/2}\right)$. Thus there exists $\tau \in O\left(n^{\varepsilon}|\varkappa_2|\right)$ and constant $c > 0$ satisfying $\mathcal{T}_2 = cn^{\varepsilon}|\varkappa_2| \geq \eta\tau$, Such that,

$$\begin{aligned}\Pr(T \leq \mathcal{T}_2) &\geq \Pr(T \leq \eta\tau)\\ &> \left(1 - 2\eta\tau e^{-\delta^2\min\{\psi_0,\gamma_0\}\lambda/8}\right)\left(1 - me^{-\eta\rho^{-\frac{\ln(\gamma_0)}{\ln(1+\delta/2)}-2}}\right)\end{aligned}$$

since $\delta, \psi_0, \gamma_0 \in (0,1)$ are constants, and $\eta = \Theta(n^{\varepsilon/2})$, $\lambda \in \Theta(n^{\xi})$,

$$= \left(1 - e^{-\Omega(\lambda)}\right)\left(1 - e^{-\Omega(\eta)}\right) = 1 - e^{-\Omega(n^{\xi})}.$$

□

Theorem 4.1 then presents the efficiency of the (μ, λ) self-adaptive EA in addressing the DSM problem. Apart from population size, other parameters such as A, b, and p_{inc} align with the previous study about unknown-structure optimisation in [1]. This demonstrates that there is no need to tune these parameters based on the specific problem. Utilising a larger population size may be necessary for tracking dynamic optima. In previous studies [5, 6], a similarly large population size of $\Theta(n^{\xi})$ was employed to track optima.

Theorem 4.1. *Let $\varepsilon, a, \beta \in (0,1)$ and $k > 0$ be some constants satisfying $1/34 \leq a < \beta \leq 1$. Let $\epsilon := b'/n$ for any constant $b' > 0$. Consider a starting target substring $\varkappa \in \{0,1\}^{\ell_1}$ with $\ell_1 \in \Theta(n^a)$ and $m \in \Theta(n^{\beta})$. Assume that all individuals in the initial population P_0 in Algorithm 1 match $\varkappa$ and have a mutation rate of ϵ. Then, there exists a constant $\xi \in (0,1)$ such that the probability of Algorithm 1 with parameters $\lambda, \mu = \Theta(n^{\xi})$, $\lambda/\mu = \alpha_0 \geq 4$, $A > 1$, $0 < b < 1/(1 + \sqrt{1/\alpha_0(1-p_{\text{inc}})})$, $(1+\delta)/\alpha_0 < p_{\text{inc}} < 2/5$, and ϵ, where A and b are constants, tracking $\text{DSM}^{\varkappa,m,\varepsilon,k}$ is $1 - e^{-\Omega(n^{\xi})}$.*

5 STATIC MUTATION-BASED EAS GET LOST

This section shows that static mutation-based EAs (Algorithms 3-4 in Appendix A) get lost in tracking the DSM problem. We first derive Lemma 5.1, which provides the upper bound of the probability of the static mutation-based EAs moving from one optimum to the next optimum in the evaluation budget of the DSM problem. This upper bound is with respect to the length of the current target substring and the mutation rate of the static mutation-based EAs. From Lemma 5.1, too high or too low mutation rates lead the algorithms to achieve the current optimum in the given evaluation budget with an insufficient probability, i.e., at most, a constant probability. More precisely, too high mutation rates are χ/n where $\chi \in \Omega(n^{1+\varepsilon}/\ell)$ and too low mutation rates are $\chi \in O(n^{1-\varepsilon}/\ell)$. Then Theorem 5.1 is proved via Lemma 5.1, which shows that there is no existing mutation rate that tracks the DSM problems with a high probability (the proof is in Appendix C.6).

Lemma 5.1. *Let $\varkappa_1$ and $\varkappa_2$ be two substrings where $\varkappa_1, \varkappa_2 \in \{0,1\}^{\ell}$ and $\mathrm{H}(\varkappa_1, \varkappa_2) = 1$ for $\ell \in [n]$. Let $\varepsilon \in (0,1)$, $\gamma_0 \in (0,1]$ and $k > 0$ be arbitrary constants. Assume that all individuals of the initial population P_0 of the static mutation-based EA are matching $\varkappa_1$. Then the probability that the static mutation-based EAs using mutation rate χ/n where $\chi \in (0, n/2]$ find a solution matching $\varkappa_2$ in $kn^{\varepsilon}\ell$ evaluations is $p < 1 - \exp\left(-\chi e^{-\chi\frac{\ell-1}{n}}\frac{kn^{\varepsilon}\ell}{n}\right)\left(1 - \frac{\chi^2 e^{-2\chi\frac{\ell-1}{n}}}{n}\right)^{\frac{kn^{\varepsilon}\ell}{n}}$.*

Proof. By the assumption of $\mathrm{H}(\varkappa_1, \varkappa_2) = 1$, in any generation t, if there is no individual in P_t matching $\varkappa_2$, then any individual in P_t has at least one mismatched bit to $\varkappa_2$. We optimistically assume that the selection operators (Line 3 of Algorithm 3 and Line 3 of Algorithm 4) always return the closest neighbour of the set of solutions matching $\varkappa_2$, i.e., with one mismatched bit and $\ell - 1$ matched bits. To obtain a solution matching $\varkappa_2$, the mutation operator must flip the mismatched bit of x, and do not flip any mismatched bit in which the probability is $(1-\chi/n)^{\ell-1}(\chi/n)$. Then, such an event happens at least once in $kn^{\varepsilon}\ell$ evaluations with $p =$

$$\begin{aligned}&1 - \left(1 - \left(1-\frac{\chi}{n}\right)^{\ell-1}\frac{\chi}{n}\right)^{kn^{\varepsilon}\ell} = 1 - \left(1 - \left(1-\frac{\chi}{n}\right)^{n\cdot\frac{\ell-1}{n}}\frac{\chi}{n}\right)^{n\cdot\frac{kn^{\varepsilon}\ell}{n}}\\ &\leq 1 - \left(1 - e^{-\chi\frac{\ell-1}{n}}\frac{\chi}{n}\right)^{n\cdot\frac{kn^{\varepsilon}\ell}{n}} \text{ by } (1+x/n)^n \leq e^x \text{ and Lemma B.1}\\ &\leq 1 - \exp\left(-\chi e^{-\chi\frac{\ell-1}{n}}\frac{kn^{\varepsilon}\ell}{n}\right)\left(1 - \frac{\chi^2 e^{-2\chi\frac{\ell-1}{n}}}{n}\right)^{\frac{kn^{\varepsilon}\ell}{n}}.\end{aligned}$$

□

Theorem 5.1. *Let $\varepsilon, a, \beta \in (0,1)$ and $k > 0$ be some constants satisfying $1/2 + \varepsilon < a + 2\varepsilon < \beta \leq 1 - \varepsilon$. Let $\varkappa \in \{0,1\}^{\ell_1}$ be some starting target substring where $\ell_1 \in \Theta(n^a)$, and $m \in \Theta(n^{\beta})$. Then the static mutation-based EAs using any mutation rate $\chi/n \in (0, 1/2]$ and population size $\lambda \in \mathbb{N}$ tracks $\text{DSM}^{\varkappa,m,\varepsilon,k}$ with probability $e^{-\Omega(n^{\beta})}$.*

6 CONCLUSION

This paper demonstrates the benefits of self-adaptation in dynamic optimisation. Our analyses show that static mutation-based EAs have a negligible chance of tracking this dynamic optima with changing structure, while the self-adaptive EA can track them. We also provide a level-based theorem with tail bounds to evaluate the performance of the self-adaptive EA on the DSM problem. Future work involves investigating self-adaptive EAs in more general settings of the DSM problem, as well as other existing dynamic optimisation problems, and identifying the limitations of self-adaptation. Additionally, future work also includes exploring other parameter control mechanisms [9, 11, 17, 18, 41] under dynamics.

ACKS. Thanks to Alistair Benford for proof assistance, and to all the reviewers for their valuable feedback. Lehre was supported by a Turing AI Fellowship (EPSRC grant ref EP/V025562/1).

REFERENCES

[1] Brendan Case and Per Kristian Lehre. 2020. Self-adaptation in non-Elitist Evolutionary Algorithms on Discrete Problems with Unknown Structure. *IEEE Transactions on Evolutionary Computation* 24, 4 (2020), 650–663.

[2] Dogan Corus, Duc-Cuong Dang, Anton Eremeev, and Per Kristian Lehre. 2018. Level-Based Analysis of Genetic Algorithms and Other Search Processes. *IEEE Transactions on Evolutionary Computation* 22, 5 (2018), 707–719.

[3] Duc-Cuong Dang, Anton Eremeev, and Per Kristian Lehre. 2021. Escaping Local Optima with Non-Elitist Evolutionary Algorithms. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35. 12275–12283.

[4] Duc-Cuong Dang, Anton Eremeev, and Per Kristian Lehre. 2021. Non-Elitist Evolutionary Algorithms Excel in Fitness Landscapes with Sparse Deceptive Regions and Dense Valleys. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '21)*. Association for Computing Machinery, 1133–1141.

[5] Duc-Cuong Dang, Thomas Jansen, and Per Kristian Lehre. 2015. Populations can be Essential in Dynamic Optimisation. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*. ACM, 1407–1414.

[6] Duc-Cuong Dang, Thomas Jansen, and Per Kristian Lehre. 2017. Populations Can Be Essential in Tracking Dynamic Optima. *Algorithmica* 78, 2 (2017), 660–680.

[7] Duc-Cuong Dang and Per Kristian Lehre. 2016. Self-adaptation of Mutation Rates in Non-elitist Populations. In *Parallel Problem Solving from Nature – PPSN XIV*, Vol. 9921. Springer International Publishing, 803–813.

[8] Benjamin Doerr. 2022. Does Comma Selection Help to Cope with Local Optima? *Algorithmica* 84, 6 (2022), 1659–1693.

[9] Benjamin Doerr and Carola Doerr. 2018. Optimal Static and Self-Adjusting Parameter Choices for the $(1+(\lambda, \lambda))$ Genetic Algorithm. *Algorithmica* 80, 5 (2018), 1658–1709.

[10] Benjamin Doerr and Carola Doerr. 2020. Theory of Parameter Control for Discrete Black-Box Optimization: Provable Performance Gains Through Dynamic Parameter Choices. In *Theory of Evolutionary Computation: Recent Developments in Discrete Optimization*, Benjamin Doerr and Frank Neumann (Eds.). Springer International Publishing, 271–321.

[11] Benjamin Doerr, Carola Doerr, and Johannes Lengler. 2019. Self-Adjusting Mutation Rates with Provably Optimal Success Rules. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '19)*. Association for Computing Machinery, 1479–1487.

[12] Benjamin Doerr and Leslie Ann Goldberg. 2010. Drift Analysis with Tail Bounds. In *Parallel Problem Solving from Nature, PPSN XI*. Springer Berlin Heidelberg, 174–183.

[13] Benjamin Doerr and Timo Kötzing. 2021. Multiplicative Up-Drift. *Algorithmica* 83, 10 (2021), 3017–3058.

[14] Benjamin Doerr, Carsten Witt, and Jing Yang. 2021. Runtime Analysis for Self-adaptive Mutation Rates. *Algorithmica* 83, 4 (2021), 1012–1053.

[15] Stefan Droste. 2003. Analysis of the (1+1) EA for a Dynamically Bitwise Changing OneMax. In *Genetic and Evolutionary Computation — GECCO 2003*, Vol. 2723. Springer Berlin Heidelberg, 909–921.

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

[17] Mario Alejandro Hevia Fajardo and Dirk Sudholt. 2021. Self-Adjusting Population Sizes for Non-Elitist Evolutionary Algorithms: Why Success Rates Matter. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '21)*. Association for Computing Machinery, 1151–1159.

[18] Mario Alejandro Hevia Fajardo and Dirk Sudholt. 2022. Hard Problems Are Easier for Success-Based Parameter Control. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '22)*. Association for Computing Machinery, 796–804.

[19] Jens Jagerskupper and Tobias Storch. 2007. When the Plus Strategy Outperforms the Comma Strategyand When Not. In *2007 IEEE Symposium on Foundations of Computational Intelligence*. IEEE, 25–32.

[20] Thomas Jansen and Ulf Schellbach. 2005. Theoretical Analysis of a Mutation-Based Evolutionary Algorithm for a Tracking Problem in the Lattice. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation (GECCO '05)*. Association for Computing Machinery, 841–848.

[21] Thomas Jansen and Ingo Wegener. 2006. On the analysis of a dynamic evolutionary algorithm. *Journal of Discrete Algorithms* 4, 1 (2006), 181–199.

[22] Yaochu Jin and Jürgen Branke. 2005. Evolutionary optimization in uncertain environments-a survey. *IEEE Transactions on Evolutionary Computation* 9, 3 (2005), 303–317.

[23] Timo Kötzing, Andrei Lissovoi, and Carsten Witt. 2015. (1+1) EA on Generalized Dynamic OneMax. In *Proceedings of the 2015 ACM Conference on Foundations of Genetic Algorithms XIII (FOGA '15)*. Association for Computing Machinery, 40–51.

[24] Timo Kötzing and Hendrik Molter. 2012. ACO Beats EA on a Dynamic Pseudo-Boolean Function. In *Parallel Problem Solving from Nature - PPSN XII*. Springer Berlin Heidelberg, 113–122.

[25] Per Kristian Lehre. 2010. Negative Drift in Populations. In *Parallel Problem Solving from Nature, PPSN XI*. Springer Berlin Heidelberg, 244–253.

[26] Per Kristian Lehre and Xiaoyu Qin. 2022. More Precise Runtime Analyses of Non-elitist Evolutionary Algorithms in Uncertain Environments. *Algorithmica* (2022).

[27] Per Kristian Lehre and Xiaoyu Qin. 2022. Self-Adaptation via Multi-Objectivisation: A Theoretical Study. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '22)*. Association for Computing Machinery, 1417–1425.

[28] Per Kristian Lehre and Dirk Sudholt. 2020. Parallel Black-Box Complexity With Tail Bounds. *IEEE Transactions on Evolutionary Computation* 24, 6 (2020), 1010–1024.

[29] Per Kristian Lehre and Carsten Witt. 2021. Tail bounds on hitting times of randomized search heuristics using variable drift analysis. *Combinatorics, Probability and Computing* 30, 4 (2021), 550–569.

[30] Johannes Lengler and Jonas Meier. 2020. Large Population Sizes and Crossover Help in Dynamic Environments. In *Parallel Problem Solving from Nature – PPSN XVI*. Springer International Publishing, 610–622.

[31] Johannes Lengler and Jonas Meier. 2022. Large population sizes and crossover help in dynamic environments. *Natural Computing* (2022).

[32] Johannes Lengler and Simone Riedi. 2021. Runtime Analysis of the $(\mu + 1)$-EA on the Dynamic BinVal Function. In *Evolutionary Computation in Combinatorial Optimization*. Springer International Publishing, 84–99.

[33] Johannes Lengler and Ulysse Schaller. 2018. The (1+1)-EA on Noisy Linear Functions with Random Positive Weights. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. 712–719.

[34] Andrei Lissovoi. 2016. *Analysis of Ant Colony Optimization and Population-Based Evolutionary Algorithms on Dynamic Problems*. PhD Thesis. Technical University of Denmark.

[35] Andrei Lissovoi and Carsten Witt. 2015. Runtime analysis of ant colony optimization on dynamic shortest path problems. *Theoretical Computer Science* 561 (2015), 73–85.

[36] Andrei Lissovoi and Carsten Witt. 2017. A Runtime Analysis of Parallel Evolutionary Algorithms in Dynamic Optimization. *Algorithmica* 78, 2 (2017), 641–659.

[37] Constantin P. Niculescu and Andrei Vernescu. 2004. A two sided estimate of $e^x - (1 + x/n)^n$. *Journal of Inequalities in Pure and Applied Mathematics* 5, 3 (2004).

[38] Pietro S. Oliveto and Carsten Witt. 2011. Simplified Drift Analysis for Proving Lower Bounds in Evolutionary Computation. *Algorithmica* 59, 3 (2011), 369–386.

[39] Pietro S. Oliveto and Christine Zarges. 2013. Analysis of Diversity Mechanisms for Optimisation in Dynamic Environments with Low Frequencies of Change. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation (GECCO '13)*. Association for Computing Machinery, 837–844.

[40] Xiaoyu Qin and Per Kristian Lehre. 2022. Self-adaptation via Multi-objectivisation: An Empirical Study. In *Parallel Problem Solving from Nature – PPSN XVII*. Springer International Publishing, 308–323.

[41] Amirhossein Rajabi and Carsten Witt. 2020. Self-Adjusting Evolutionary Algorithms for Multimodal Optimization. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20)*. Association for Computing Machinery, 1314–1322.

[42] Philipp Rohlfshagen, Per Kristian Lehre, and Xin Yao. 2009. Dynamic Evolutionary Optimisation: An Analysis of Frequency and Magnitude of Change. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09)*. Association for Computing Machinery, 1713–1720.

[43] Jim Smith. 2001. Modelling Gas with Self Adaptive Mutation Rates. In *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation (GECCO'01)*. Morgan Kaufmann Publishers Inc., 599–606.

[44] Karsten Weicker. 2005. Analysis of local operators applied to discrete tracking problems. *Soft Computing* 9, 11 (2005), 778–792.