
Proceedings Paper:
Hall, G.T., Oliveto, P.S. and Sudholt, D. (2020) Analysis of the performance of algorithm configurators for search heuristics with global mutation operators. In: Coello Coello, C.A., (ed.) GECCO 2020: Proceedings of the Genetic and Evolutionary Computation Conference. GECCO '20: Genetic and Evolutionary Computation Conference, 08-12 Jul 2020, Cancún Mexico (Online). ACM Digital Library, pp. 823-831. ISBN: 9781450371285.

https://doi.org/10.1145/3377930.3390218

Analysis of the Performance of Algorithm Configurators for Search Heuristics with Global Mutation Operators

George T. Hall
Department of Computer Science
University of Sheffield, Sheffield, UK

Pietro S. Oliveto
Department of Computer Science
University of Sheffield, Sheffield, UK

Dirk Sudholt
Department of Computer Science
University of Sheffield, Sheffield, UK

ABSTRACT

Recently it has been proved that a simple algorithm configurator called ParamRLS can efficiently identify the optimal neighbourhood size to be used by stochastic local search to optimise two standard benchmark problem classes. In this paper we analyse the performance of algorithm configurators for tuning the more sophisticated global mutation operator used in standard evolutionary algorithms, which flips each of the n bits independently with probability χ/n and the best value for χ has to be identified. We compare the performance of configurators when the best-found fitness values within the cutoff time κ are used to compare configurations against the actual optimisation time for two standard benchmark problem classes, Ridge and LeadingOnes. We rigorously prove that all algorithm configurators that use optimisation time as performance metric require cutoff times that are at least as large as the expected optimisation time to identify the optimal configuration. Matters are considerably different if the fitness metric is used. To show this we prove that the simple ParamRLS-F configurator can identify the optimal mutation rates even when using cutoff times that are considerably smaller than the expected optimisation time of the best parameter value for both problem classes.

CCS CONCEPTS

• **Theory of computation** → *Theory of randomized search heuristics*;

KEYWORDS

Parameter tuning, Algorithm configurators, Runtime analysis

1 INTRODUCTION

General purpose search heuristics, such as evolutionary algorithms, are designed with the aim of optimising a problem given minimal knowledge about it. Usually all that is needed is a means of representing solutions for the problem and a fitness function to compare the quality of different candidate solutions. Whilst these algorithms have been shown to be effective for solving a large variety of hard optimisation problems, a common difficulty is that of choosing a suitable algorithm for the problem at hand and setting its parameter values such that it will have good performance. A result of this is that it has become very common to use automated methodologies for algorithm development [3–5, 13, 17–19, 21].

Traditionally, parameter values were chosen manually by the user, applying the algorithm to a specific problem and subsequently refining the choices according to the algorithm's performance with the tested parameters. This method, however, is time-consuming, tedious, and error-prone. As a result automatic algorithm configurators have gradually become the standard methodology used to tune the parameters of an algorithm for a class of problems. Popular tuners include ParamILS, which uses iterated local search to navigate the space of configurations [10]; irace, which iteratively evaluates many configurations concurrently, eliminates those which statistically have worst performance, and then uses the best to update the distribution used to generate new candidate configurations [20]; and the surrogate model-based tuners SPOT [1] and SMAC [11], which use approximations of the parameter landscape in order to avoid many lengthy evaluations of configurations.

Despite their widespread adoption, there is a lack of understanding of the behaviour and performance of algorithm configurators. More specifically, given an algorithm and a class of problems, it is unclear how good the parameter values returned by a given configurator actually are, how long the configurator should be run such that good parameter values are identified, nor is there any rigorous guidance available on how to set the configurator's inherent parameters.

Recently, Kleinberg *et al.* [14] provided preliminary answers to these questions. They performed a worst-case runtime analysis of standard algorithm configurators, in which an adversary causes every deterministic choice to play out as poorly as possible, while observations of random variables are unbiased samples from the underlying distribution. They proved that in this scenario all popular configurators will perform poorly. On the other hand, they presented a worst-case tailored algorithm called Structured Procrastination (SP) that provably performs better in the worst-case. Several improvements to this approach have recently been published [15, 23, 24]. Naturally, it is unlikely that the worst-case scenario occurs in practical applications of algorithm configurators. In fact, Pushak and Hoos [22] recently investigated the structure of configuration search landscapes. Their experimental analysis suggests that the search landscapes are largely unimodal and convex when tuning algorithms for well-known instance sets of a variety of NP-hard problems including SAT, MIP and TSP. Thus, they provided evidence that generally algorithm configuration landscapes are much more benign for popular gradient-based configurators than in the worst-case scenario. Thus, it is important to rigorously evaluate for which applications a given algorithm configurator will be efficient and for which it will perform poorly.

The only available time complexity analysis deriving the time required by an algorithm configurator to identify the optimal parameters for an algorithm for specific problem classes is an earlier publication of ours [7]. We proved that a simplified version of ParamILS, called ParamRLS, can efficiently identify the optimal neighbourhood size k of a randomised local search algorithm RLS_k for two standard benchmark problem classes. An important insight gained from our earlier analysis is that, if the best identified fitness within some cutoff time is used for configuration comparisons, then

much smaller cutoff times than the actual optimisation time of the optimal configurations may be used by ParamRLS to identify the optimal parameter. On the other hand, if the optimisation time is used for the comparisons, then the cutoff time has to be much larger i.e., at least the expected runtime of the optimal parameter setting.

In order to gain a deeper understanding of the performance of algorithm configurators, in this paper we consider the problem of tuning the mutation rate of the global mutation operator that is commonly used in standard evolutionary algorithms. The operator, called *standard bit mutation* (SBM), flips each bit of a bit string of length n with probability χ/n, and χ is the parameter value to be tuned. This operator is considerably more sophisticated than the local mutations used by the RLS algorithm considered in our earlier work, since an arbitrary number of bits between 0 and n may be flipped by the operator in each mutation operation. Furthermore, the nature of the parameter to be tuned yields a significantly more complex parameter landscape. While the parameter of RLS_k may only take discrete values, the search space for standard bit mutation's parameter is continuous as the parameter χ may take any real value. Small differences in χ (e. g. $1/n$ vs. $1.1/n$) are hardly visible as in most mutations the number of flipped bits is identical. In stark contrast, RLS_k behaves very differently when always flipping, say, $k = 1$ bit or always flipping $k = 2$ bits. Hence, identifying the optimal standard bit mutation rate is much harder than tuning RLS_k as in our earlier work.

We embed the SBM operator into a simple evolutionary algorithm, the (1+1) EA, and consider the problem of identifying its optimal parameter value for two benchmark function classes: Ridge and LeadingOnes. The first function class is chosen because for each instance the optimal mutation rate for SBM is always $1/n$ independent of the position of the current solution in the search space. This characteristic is ideal for a first time complexity analysis as it should be easy for the configurator to identify the optimal parameter value and, at the same time, it keeps the analysis simple. The second function class is more challenging because the best mutation rate decreases as the algorithm approaches the optimum and the configurator has to identify that the best compromise is a mutation rate of $\approx 1.59/n$ which minimises the overall expected runtime of the (1+1) EA [2].

Our aim is to characterise the impact of the performance metric on the cutoff time required for algorithm configurators to identify the optimal parameter value of the (1+1) EA for the considered problem classes. As in our publication considering RLS_k, in this paper we consider two performance metrics: *Optimisation time*, where the winner of a comparison is the configuration which reaches the optimum in the fewest iterations; and *Best fitness* where the winner of a comparison of two configurations is the configuration which achieves the highest fitness value within the cutoff time κ.

We prove that, with overwhelming probability (w. o. p.)[1], any algorithm configurator that uses Optimisation time as performance metric requires a cutoff time that is at least as large as the optimisation time of the optimal parameter value for both Ridge and LeadingOnes. For smaller cutoff times it returns a parameter value chosen uniformly at random from the parameter space w. o. p. unless the configurator is inherently biased towards some areas of the search space. For the simple Ridge and LeadingOnes problem classes, a random parameter value is returned w. o. p. respectively for cutoff times of $\kappa \leq (1-\varepsilon)en^2$ and $\kappa \leq 0.772075n^2$. Matters change considerably for algorithm configurators that use Best fitness as performance metric. To prove this it suffices to consider the simple randomized local search ParamRLS-F algorithm analysed in our earlier work which uses Best fitness as performance metric. ParamRLS-F efficiently returns the optimal parameter value $\chi = 1$ of the (1+1) EA for Ridge for any cutoff time that is at least linear in the problem size. Notice that the configurator is efficient for cutoff times that are a linear factor smaller than the expected optimisation time of the (1+1) EA with the optimal $1/n$ mutation rate. For LeadingOnes, we prove that, w. o. p., ParamRLS-F is able to find the optimal parameter setting of $\chi = 1.6$ (where χ is allowed to take values from the set $\{0.1, 0.2, \ldots, 2.9, 3.0\}$) for any cutoff time $\kappa \geq 0.721118n^2$. Note that this is $\approx 0.05n^2$ smaller than the expected optimisation time for any configuration of the (1+1) EA for LeadingOnes [2]. For over 99% of cutoff times in the range between $0.000001n^2$ and $0.720843n^2$, we prove that ParamRLS-F returns the optimal parameter setting for the cutoff time, w. o. p. That is, the smaller the cutoff time, the higher the optimal mutation rate. Some proofs are omitted due to space restrictions[2]. For both Ridge and LeadingOnes note that, while the formal proof is provided for the ParamRLS-F tuner, the analysis implies that all algorithm configurators capable of hillclimbing are efficient at tuning the (1+1) EA for the same cutoff time values if they use Best fitness as performance metric.

2 PRELIMINARIES

We first give an overview of the algorithm configuration problem. This is a formalisation of the problem which all parameter tuners attempt to address. We then outline the three main subjects of analysis in this work. We first present the algorithm configurator. We then outline the target algorithm (the algorithm which we are analysing the ability of ParamRLS to tune) before giving an overview of the two benchmark target function classes which we consider.

2.1 The Algorithm Configuration Problem

The Algorithm Configuration Problem (ACP) is that of choosing the parameters for a *target algorithm* $\mathcal{A}$ to optimise its performance across a class of problems Π. Let us denote the set of all configurations of $\mathcal{A}$ as Θ and algorithm $\mathcal{A}$ with its parameters set according to some configuration $\theta \in \Theta$ as $\mathcal{A}(\theta)$. Then the Algorithm Configuration Problem is the task of identifying a configuration θ^* such that

$$\theta^* \in \arg\min_{\theta \in \Theta} cost(\theta)$$

where $cost(\theta)$ is some measure of the cost of running $\mathcal{A}(\theta)$ on the problem class Π.

We must therefore define the measure $cost(\theta)$, which depends on several factors: The size of the *cutoff time* κ (the number of iterations in a single run of a comparison); The number of *runs*

[1] We define an event A as occurring with overwhelming probability if and only if $\Pr(A) = 1 - \exp(-\Omega(n^\alpha))$, for some positive constant α. Note that, by the union bound, the intersection of polynomially many such events still has an overwhelming probability.

[2] A full version of this paper is available on arXiv [8].

per comparison r (the number of times we evaluate two configurations in a single comparison); Which *metric* is used to evaluate the performance of a configuration on a problem instance; How to *aggregate* performance measures over multiple runs; How many *instances* (and which ones) to include in the training set.

In this work, we address the configurator's parameters as follows. All results in this work hold for any polynomial number of runs per comparison (i.e. the positive results hold even for just one run and the negative results hold even for a large polynomial number of runs). We consider two performance metrics. The Optimisation time metric quantifies the performance of a configuration by the time taken to reach the optimum. A penalisation constant multiplied by the cutoff time κ is returned if the configuration does not reach the optimum within κ iterations (called PAR10 for penalisation constant of 10). The Best fitness performance metric considers the highest fitness value achieved within the cutoff time. The definition of the training set and the method of aggregation are both irrelevant within this paper, since performance on the single problem instances we consider here generalises to any other instance of the problem class.

Let T be the number of configuration comparisons carried out before the tuner returns the optimal configuration θ^* w. o. p. Then the total required tuning budget is $\mathcal{B} = 2 \cdot T \cdot |\Pi| \cdot \kappa \cdot r$. In this work, we want to estimate how the two performance metrics impact the cutoff time κ and the total number of comparisons T required for a simple algorithm configurator ParamRLS to tune the (1+1) EA for two benchmark problem classes.

2.2 The Configurator: ParamRLS

We follow our earlier work in analysing a simplified version of the popular ParamILS parameter tuner, called ParamRLS [7].

At the heart of ParamRLS is the *active parameter*. ParamRLS initialises the active parameter uniformly at random. It then repeatedly mutates it and accepts the offspring (that is, updates the active parameter to this new value) if it performs at least as well as the active parameter according to a routine `eval`. Each call to `eval` is called a *comparison* and the *runtime* T is defined as the number of comparisons until the optimal parameter value is identified. The high-level pseudocode for ParamRLS is given in Algorithm 1. The `eval` routine takes as arguments both configurations to be compared, as well as the cutoff time κ and number of runs r. Both configurations are then executed for r runs (each of length κ) where at the end of each run the winner of the run is decided according to one of two *performance metrics*. Under the Best fitness metric, the winner of a run is the configuration which has the highest fitness value after κ iterations. If both configurations have the same fitness value at time κ, then the winner is the one that found it first. The winner of the overall comparison is the configuration which won the most runs. Ties are broken uniformly at random. The variant of ParamRLS using this performance metric is called ParamRLS-F, and its pseudocode is given in Algorithm 2. Under the Optimisation time metric, the optimisation times of both comparisons are summed for r runs. If in a run a configuration fails to reach the optimum within κ iterations then its optimisation time is taken to be $p \cdot \kappa$, where p is a penalty constant. If there is a tie after all runs have been completed, the winner is decided uniformly at random. This measure is called Penalised Average Runtime (PAR). It is commonly used in configurators such as ParamILS. The variant of ParamRLS using this performance metric is called ParamRLS-T, and its pseudocode is given in Algorithm 3.

Algorithm 1 ParamRLS ($\mathcal{A}, \Theta, \Pi, \kappa, r$). Recreated from [7] with minor typographical modifications.

```
θ ←initial parameter value chosen uniformly at random
while termination condition not satisfied do
    θ' ← mutate(θ)
    θ ← eval(A, θ, θ', π, κ, r)
return θ
```

As in [7], the `mutate` routine in ParamRLS alters the current configuration according to some *local search operator* $\pm\{1\}$. This operator simply increases or decreases the current parameter value by $1/d$, both with probability 0.5. If the new value for the parameter is 0 or $\phi + (1/d)$ then this new configuration loses any comparison with probability 1 (that is, it is rejected with certainty).

Since we consider a continuously-valued parameter, χ, we discretise the parameter space (the search space of all configurations) as this is also the method used to deal with continuous parameters in ParamILS [10]. We do so using a *discretisation factor* d. We define the parameter space as consisting of all numbers z/d where $z \in \{1, 2, 3, \ldots, \phi \cdot (d-2), \phi \cdot (d-1), \phi \cdot d\}$, for an integer ϕ. For example, if $d = 4$ then the possible values for χ will be $0.25, 0.5, 0.75, 1, 1.25, 1.5, 1.75, 2, \ldots, \phi - 0.25, \phi$. We do not consider $\chi = 0$ since this means that bits are never flipped in the target algorithm. We call the fitness landscape induced by the target algorithm on the target function under a performance metric the *configuration landscape*. We say that a configuration landscape is *unimodal* if and only if there is only one optimal configuration and, for all pairs of neighbouring configurations (two configurations are neighbours if one can be reached from the other in a single mutation using the local search operator), the configuration closer to the optimum wins a comparison w. o. p.

We call a tuner *blind* if there is an event A that occurs w. o. p. and, conditional on A, the tuner returns a configuration chosen according to a distribution which would be generated if all configurations had the same performance. For ParamRLS-T this implies that the configuration will be chosen uniformly at random, since if there is no information to separate two configurations then the winner is chosen uniformly at random. Only if a tuner is inherently biased will the outcome be non-uniformly distributed.

2.3 The Target Algorithm: The (1+1) EA

In each iteration, the $(1+1)_\chi$ EA, shown in Algorithm 4, creates a new solution by flipping each bit in the bit string of the current solution independently with probability χ/n. The offspring is accepted if its fitness is at least that of its parent. We analyse the number of comparisons for ParamRLS to identify the optimal value for χ (and thus the optimal mutation rate) for a given problem class. In this work we assume that χ is constant.

Algorithm 2 The `eval-F`$(\mathcal{A}, \theta, \theta', \pi, \kappa, r)$ subroutine in ParamRLS-F. Recreated from [7] with minor typographical modifications.

$Wins \leftarrow 0$; $Wins' \leftarrow 0$ {count number of wins for θ and θ'}
$R \leftarrow 0$
while $R < r$ **do**
  $ImprTime \leftarrow 0$
  $ImprTime' \leftarrow 0$
  $Fitness \leftarrow \mathcal{A}(\theta)$ fitness after κ iterations;
  $Fitness' \leftarrow \mathcal{A}(\theta')$ fitness after κ iterations;
  $ImprTime \leftarrow$time of last improvement of $\mathcal{A}(\theta)$
  $ImprTime' \leftarrow$time of last improvement of $\mathcal{A}(\theta')$
  if $Fitness > Fitness'$ **then**
    $Wins \leftarrow Wins + 1$
  else if $Fitness' > Fitness$ **then**
    $Wins' \leftarrow Wins' + 1$
  else
    if $ImprTime < ImprTime'$ **then**
      $Wins \leftarrow Wins + 1$
    else if $ImprTime' < ImprTime$ **then**
      $Wins' \leftarrow Wins' + 1$
  $R \leftarrow R + 1$
if $Wins > Wins'$ **then return** θ
else if $Wins' > Wins$ **then return** θ'
else return a uniform choice of θ or θ'

Algorithm 3 The `eval-T`$(\mathcal{A}, \theta, \theta', \pi, \kappa, r)$ subroutine in ParamRLS-T. Recreated from [7] with minor typographical modifications.

$Time \leftarrow 0$; $Time' \leftarrow 0$ {count optimisation times for $\mathcal{A}(\theta)$ and $\mathcal{A}(\theta')$}
$R \leftarrow 0$
while $R < r$ **do**
  $Time \leftarrow Time +$ `CapOptTime`$(\mathcal{A}(\theta), \kappa, p)$
  $Time' \leftarrow Time' +$ `CapOptTime`$(\mathcal{A}(\theta'), \kappa, p)$
  $R \leftarrow R + 1$
if $Time < Time'$ **then return** θ
else if $Time' < Time$ **then return** θ'
else return a uniform choice of θ or θ'

Algorithm 4 The (1+1) EA maximising a function f

initialise x {according to initialisation scheme}
while termination criterion not met **do**
  $x' \leftarrow x$ with each bit flipped with probability χ/n
  if $f(x') \geq f(x)$ **then** $x \leftarrow x'$

2.4 Problem Classes: Ridge and LeadingOnes

We analyse the ability of ParamRLS to configure the (1+1) EA for two standard benchmark problem classes: Ridge and LeadingOnes. Despite their similar definitions, the (1+1) EA exhibits substantially different behaviour when optimising them.

The most commonly analysed instance of the Ridge problem class is the function

$$\textsc{Ridge}(x) = \begin{cases} n + |x|_{\textsc{ones}}, & \text{if } x \text{ in form } 1^i 0^{n-i} \\ n - |x|_{\textsc{ones}}, & \text{otherwise} \end{cases}$$

where $|x|_{\textsc{ones}}$ is the number of ones in the bit string x.

The other instances of the problem class are provided by its black box definition, given by [6] and used by [7], which in a nutshell takes the XOR with another bit string $a \in \{0,1\}^n$: $\textsc{Ridge}_a := \textsc{Ridge}(x_1 \oplus a_1 \dots x_n \oplus a_n)$. We thus analyse only the problem instance $\textsc{Ridge}_{0^n}$, and observe that the best parameter value for this problem instance will be optimal for all 2^n instances of the problem class.

Following previous work [7, 12], we assume that the (1+1) EA is initialised to 0^n. This means that the algorithm builds a string of consecutive 1 bits followed by a string of consecutive 0 bits, and the optimum is the 1^n bit string.

For the (1+1) EA, it is optimal to set $\chi = 1$ to achieve the smallest expected optimisation time for Ridge. We prove this in Lemma 3.1(ii). The Ridge problem class is a natural one to analyse initially since it is always best to use a mutation rate of $1/n$. This characteristic simplifies the analysis.

The second problem class is LeadingOnes, or 'LO' for short. In the problem instance we consider, the fitness value of a bit string is equal to the number of consecutive 1 bits at the beginning of the string, $\text{LO}(x) = \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$. This differs from Ridge since Ridge requires the bit string always be in the form $1^i 0^{n-i}$, whereas LO places no such importance on the bits following the first 0.

Droste *et al.* define the black box optimisation class of LO as the class consisting of problem instances $\text{LO}_a(x)$, where this is taken to be the length of the longest prefix of x in which all bits match the prefix of $\overline{a}$ [6]. Naturally, the best mutation rate for one instance will also be optimal for all the other instances in the problem class.

Böttcher *et al.* proved that setting $\chi = 1.59\dots$ leads to the shortest expected optimisation time for the (1+1) EA for the LO problem class for any static mutation rate [2]. LO presents a more complex problem for which to tune than that presented by Ridge, since it is beneficial to use a higher mutation rate earlier in the optimisation process. Intuitively, this is because it is necessary to preserve the current prefix of leading ones in order to make progress, thus as the prefix grows, higher mutation rates are more likely to flip bits within it. Hence it is challenging to determine the behaviour of the tuner for different cutoff times.

3 TUNING THE (1+1) EA FOR RIDGE

Before we can prove any results on the performance of ParamRLS when tuning the (1+1) EA for Ridge, it is first necessary to analyse the performance of the (1+1) EA on this function. We therefore begin this section by deriving a lemma which tells us several facts about its behaviour. We first bound the *drift* (the change of the fitness of the current individual) in one generation of the (1+1) EA. We denote the drift of the $(1+1)_\chi$ EA on Ridge by Δ_χ, and define it as $\Delta_\chi(x_t) := \textsc{Ridge}(x_{t+1}) - \textsc{Ridge}(x_t)$. The following lemma summarises key statements about the performance of the $(1+1)_\chi$ EA on Ridge.

Lemma 3.1. *Consider the (1+1) EA optimising Ridge. Assume that it was initialised to 0^n. Then the following is true:*

(i) *The expected drift of the (1+1)$_\chi$ EA,* $\mathrm{E}[\Delta_\chi(x_t) \mid x_t]$*, is bounded as follows:*

$$\frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^{n-1} \leq \mathrm{E}[\Delta_\chi(x_t) \mid x_t, x_t \neq 1^n]$$
$$\mathrm{E}[\Delta_\chi(x_t) \mid x_t] \leq \frac{\chi}{n}\left(1-\frac{\chi}{n}\right)^{n-1} + O\left(\frac{1}{n^2}\right)$$

(ii) *Setting* $\chi = 1$ *yields the smallest expected optimisation time for any constant* χ*, which is at most* en^2*, assuming* n *is large enough.*

The first statement follows as the probability of improving the fitness of $x_t \neq 1^n$ by 1 is $\chi/n \cdot (1-\chi/n)^{n-1}$ as it is necessary and sufficient to flip the first 0-bit and not to flip the other $n-1$ bits. Larger jumps have an exponentially decaying probability, reflected in the $O(1/n^2)$ term. For the second statement we use additive drift theory [9, 16], which in a nutshell derives (bounds on) first hitting times from (bounds on) the expected drift and the initial distance from the target state. Hence the expected optimisation time is minimised by the parameter that maximises the drift. The second statement follows from the first one since[3] $(1-\chi/n)^{n-1} \approx e^{-\chi}$ and the function $\chi e^{-\chi}$ is maximised for $\chi = 1$.

3.1 Analysis of ParamRLS-F Tuning for Ridge

We now prove that, for any discretisation factor d, ParamRLS-F is able to configure the (1+1) EA for Ridge. In particular, we show that, given any cutoff time $\kappa \geq c'n$, for a sufficiently large constant $c' > 0$, the expected number of comparisons in ParamRLS-F before the active parameter is set to $\chi = 1$ is at most $6d\phi$. Moreover, after dn^ε comparisons with cutoff time $\kappa \geq n^{1+\varepsilon}$, for any positive constant ε, ParamRLS-F returns the optimal configuration w. o. p.

Theorem 3.2. *Consider ParamRLS-F tuning the (1+1) EA for Ridge, where the target algorithm is initialised to* 0^n*. Assume that* $d, \phi = \Theta(1)$ *and* $\kappa \in \mathrm{poly}(n)$*. Then:*

- *Using cutoff times* $\kappa \geq c'n$ *for a sufficiently large constant* $c' > 0$*, the expected number of comparisons in ParamRLS-F before the active parameter value has been set to* $\chi = 1$ *is at most* $6d\phi$*.*
- *Using cutoff times* $\kappa \geq n^{1+\varepsilon}$*, for some constant* $\varepsilon > 0$*, if ParamRLS-F runs for* dn^ε *comparisons then it returns the parameter value* $\chi = 1$ *with overwhelming probability.*

We prove the above theorem by bounding the probability that one configuration has a higher fitness than another after κ iterations. For large enough cutoff times, in a comparison between two configurations which either both have $\chi \leq 1$ or both have $\chi \geq 1$, the configuration with χ closer to 1 wins.

Lemma 3.3. *Assume that the (1+1)$_a$ EA and the (1+1)$_b$ EA, with* a *and* b *two positive constants such that* $ae^{-a} > be^{-b}$*, are both initialised to* 0^n*. Then with probability at least*

$$1 - 3\exp(-\Omega(\kappa/n)) - \kappa\exp(-\Omega(n))$$

the (1+1)$_a$ EA wins in a comparison in ParamRLS-F against the (1+1)$_b$ EA on Ridge with cutoff time κ*. Note that if* a *and* b *satisfy either* $0 < b < a \leq 1$ *or* $1 \leq a < b \leq \phi$ *then the condition* $ae^{-a} > be^{-b}$ *is implied.*

The lemma follows from showing, through the use of appropriate Chernoff bounds, that either the (1+1)$_a$ EA is ahead of the (1+1)$_b$ EA after κ iterations or that the (1+1)$_a$ EA finds the optimum sooner than the (1+1)$_b$ EA.

Now we are able to prove Theorem 3.2.

Proof of Theorem 3.2. Given a comparison of a pair of configurations, let us call the configuration with a value of χ closer to 1 the 'better' configuration, and the other configuration the 'worse' configuration.

By Lemma 3.3, using $\kappa \geq c'n$, the probability that the better configuration wins a comparison with cutoff time κ is at least $1 - \exp(-\Omega(\kappa/n)) - \kappa\exp(-\Omega(n)) \geq 2/3$, the inequality holding since we can choose the constant $c' > 0$ appropriately and $\kappa \in \mathrm{poly}(n)$. The current configuration is compared against a better one with probability at least $1/2$, and it is compared against a worse one with probability at most $1/2$. Hence the distance to the optimal parameter value decreases in expectation by at least $1/d \cdot (1/2 \cdot 2/3 - 1/2 \cdot 1/3) = 1/(6d)$. The initial distance is at most ϕ. By additive drift arguments (Theorem 5 in [16]), the expected time to reach the optimal parameter value for the first time is at most $6d\phi$.

For the second statement, we use that if $\kappa \geq n^{1+\varepsilon}$ then the probability of accepting a worse configuration is exponentially small. Hence, w. o. p., within any polynomial number of comparisons, we never experience the event that the worse configuration wins a comparison. This implies that $\max(1, \phi-1)(d+1)$ steps decreasing the distance towards the optimal parameter are sufficient. By Chernoff bounds, the probability of not seeing this many steps in dn^ε iterations is exponentially small. Finally, once the optimal parameter is reached, it is never left w. o. p. Thus, after dn^ε iterations, the optimal parameter is returned with overwhelming probability. □

Lemma 3.3 implies that any parameter tuner capable of hillclimbing will return the optimal configuration w. o. p., given sufficiently many comparisons, if it uses cutoff times of $\kappa \geq n^{1+\varepsilon}$ and the highest fitness value performance metric.

3.2 Analysis of Optimisation Time-Based Comparisons When Tuning for Ridge

While ParamRLS-F succeeds at tuning the (1+1) EA for Ridge with a cutoff time of $\kappa \geq n^{1+\varepsilon}$, we now show that all algorithm configurators that use Optimisation time as performance metric fail w. o. p. to identify the optimal configuration if the cutoff time is at most $\kappa \leq (1-\varepsilon)en^2$. Established tuners such as irace, ParamILS, and SMAC as well as recent theory-driven approaches such as Structured Procrastination all fall into this category if Optimisation time is used as performance metric. For such cutoff times, all configurations (i.e. the target algorithm with any parameter value) fail to find the optimum w. o. p.

[3] We use an $\approx$ symbol for illustration purposes only. Proofs use the double inequality $(1-\chi/n)^n \leq e^{-\chi} \leq (1-\chi/n)^{n-\chi}$.

Lemma 3.4. *For all constants $\chi, \varepsilon > 0$, the (1+1)$_\chi$ EA requires more than $(1-\varepsilon)en^2$ iterations to reach the optimum of Ridge, with probability $1 - \exp(-\Omega(n))$.*

This yields that all configurators that use the Optimisation time performance metric are blind if the cutoff time is at most $\kappa \leq (1-\varepsilon)en^2$.

Theorem 3.5. *Consider any configurator using the Optimisation time performance metric tuning the (1+1) EA for Ridge for any positive constant ϕ and discretisation factor d. If the cutoff time for each run is never allowed to exceed $\kappa \leq (1-\varepsilon)en^2$, for some constant $\varepsilon > 0$, then, after any polynomial number of comparisons and runs per comparison, the configurator is blind.*

Proof of Theorem 3.5. Lemma 3.4 tells us that, for cutoff times $\kappa \leq (1-\varepsilon)en^2$, all configurations of the (1+1)$_\chi$ EA, for every constant choice of χ, fail to reach the optimum of Ridge, with overwhelming probability. If this happens in all comparisons, then, since the Optimisation time metric is being used, all configurations will have the same fitness: κ multiplied by the penalisation constant. Thus, there is no attraction towards the optimal parameter value. Therefore, with overwhelming probability, by the union bound, the configurator will behave as if all configurations have the same performance, and therefore it is blind. □

4 TUNING THE (1+1) EA FOR LEADINGONES

We now show that ParamRLS-F is able to find optimal parameter values for the (1+1)$_\chi$ EA optimising LeadingOnes for almost all quadratic cutoff times κ. The analysis is considerably more complicated than for Ridge since the progress depends (mildly, but significantly) on the current fitness. For a search point with k leading ones, the probability of improving the fitness is exactly $\chi/n \cdot (1-\chi/n)^k$ as it is necessary and sufficient to flip the first 0-bit while not flipping the k leading ones. This probability decreases over the course of a run, from $\chi/n \cdot (1-\chi/n)^0 = \chi/n$ for $k = 0$ to $\chi/n \cdot (1-\chi/n)^{n-1} \approx \chi/(en)$ for $k = n-1$. This effect is similar to that observed for the function OneMax in [7]. We therefore follow our approach in that work and establish intervals that bound the "typical" fitness at various stages of a run. This allows us to locate the final fitness after κ iterations with high precision and w. o. p. For almost all cutoff times, our fitness intervals reveal that the configuration closer to the optimal one leads to a better final fitness, w. o. p.

Due to the increased complexity of the analysis, we focus on one specific discretisation factor d and choice of ϕ as a proof of concept. We are confident that our method generalises to any constant discretisation factor by increasing the precision of our analytical results (by means of the period length introduced in Lemma 4.2) in relation to the granularity of the parameter space (given by d). We provide a Python tool which applies our proof technique for an arbitrary period length[4]. Therefore the user can repeatedly decrease the period length until this tool is able to prove the desired results for their chosen parameter space.

We choose a discretisation factor of $d = 10$ and $\phi = 3$, which implies $\chi \in \{0.1, 0.2, \ldots, 2.9, 3.0\}$. The mutation rate which produces the smallest expected optimisation time for the (1+1) EA optimising LO is $\approx 1.59/n$ [2], and it is easily verified that the optimal parameter with the chosen granularity is $\chi = 1.6$. We expect the tuner to return $\chi = 1.6$ when the cutoff time is large enough. For smaller cutoff times we expect the tuner to return larger values of χ, since for LO it is beneficial to flip more bits when early in the optimisation process. We prove that, for ParamRLS-F, both of these intuitions are correct. However, tuners using the Optimisation time performance metric require larger cutoff times in order to identify the optimal configuration.

4.1 Analysis of ParamRLS-F Tuning for LO

In this section we prove two results. We first prove that the parameter landscape is unimodal for over 99% of cutoff times in the range $0.000001n^2$ to $0.772075n^2$. We then prove that the parameter landscape is unimodal for all cutoff times of at least $0.772076n^2$, and that the optimal mutation rate (and that returned by ParamRLS-F) for these cutoff times is $1.6/n$, as expected. These results imply that, given sufficiently many comparisons, ParamRLS-F will, w. o. p., return the mutation rate with the smallest expected optimisation time for all cutoff times of at least $0.772076n^2$, and for over 99% of cutoff times in the range $0.000001n^2$ to $0.772075n^2$ it will return the mutation rate which is optimal (that is, it achieves the highest fitness) for that cutoff time.

Theorem 4.1. *Consider ParamRLS-F tuning the (1+1) EA for LO with $\chi \in \{0.1, 0.2, \ldots, 2.9, 3.0\}$ (i. e. $d = 10, \phi = 3$). For all cutoff times in one of the ranges listed in Table 1 and $\kappa \geq 0.772076n^2$ it holds that, for any positive constant ε:*

- *The expected number of comparisons in ParamRLS-F before the active parameter is set to the optimal value for the cutoff time (see Table 1) is at most $2d\phi + \exp(-\Omega(n^\varepsilon))$.*
- *If ParamRLS-F is run for a number of comparisons which is both polynomial and at least n^ε then it returns the optimal parameter value for the cutoff time with overwhelming probability.*

In order to prove Theorem 4.1, we first bound the progress made by the (1+1) EA in n^2/ψ iterations (for a positive constant ψ), a length of time we call a *period*. We define progress as the difference between the distance to the optimum at the beginning of the period and at the end of the period. We then sum the progress made in a constant number of periods in order to bound the fitness of the individual in the (1+1) EA after a quadratic number of iterations. We compute the cutoff times required such that these intervals do not overlap. This tells us which configuration will win in a comparison of that length.

We first derive progress bounds for the (1+1)$_\chi$ EA.

Lemma 4.2. *Consider the (1+1) EA optimising LO for a period of n^2/ψ iterations, for some positive constant ψ, that starts with a fitness of j. Let Z be the amount of progress made by the algorithm over the period. Then, w. o. p.:*

(i) $Z \leq \frac{2\chi n}{\psi \cdot \exp\left(\frac{\chi j}{n}\right)} + o(n)$

(ii) For every i with $j \leq i < n$, $Z \geq \frac{2\chi n}{\psi \cdot \exp\left(\frac{\chi i}{n}\right)} - o(n)$, or the algorithm exceeds fitness i at the end of the period.

[4] Available at https://github.com/george-hall-sheff/leading_ones_recurrences_tool.

χ	lower bound on κ	upper bound on κ
3.0	$0.000030n^2$	$0.225138n^2$
2.9	$0.225628n^2$	$0.241246n^2$
2.8	$0.241720n^2$	$0.259143n^2$
2.7	$0.259600n^2$	$0.279105n^2$
2.6	$0.279545n^2$	$0.301461n^2$
2.5	$0.301885n^2$	$0.326611n^2$
2.4	$0.327018n^2$	$0.355040n^2$
2.3	$0.355431n^2$	$0.387346n^2$
2.2	$0.387720n^2$	$0.424266n^2$
2.1	$0.424623n^2$	$0.466723n^2$
2.0	$0.467064n^2$	$0.515884n^2$
1.9	$0.516208n^2$	$0.573238n^2$
1.8	$0.573546n^2$	$0.640714n^2$
1.7	$0.641006n^2$	$0.720843n^2$
1.6	$0.721118n^2$	$0.772075n^2$

Table 1: Ranges of κ for which the parameter landscape is unimodal with the optimum at χ. The parameter landscape is also unimodal with the optimum at $\chi = 1.6$ for cutoff times $\kappa \geq 0.772076n^2$, as shown in Theorem 4.1.

The intuition behind these bounds is that the probability of improving the fitness of a search point with k leading ones is $\chi/n \cdot (1-\chi/n)^k$, which is at least $\chi/n \cdot (1-\chi/n)^i \approx \chi/n \cdot \exp(-\chi i/n)$ and at most $\chi/n \cdot (1-\chi/n)^j \approx \chi/n \cdot \exp(-\chi j/n)$ if $j \leq k \leq i$. The factor of 2 stems from the fact that, when the first 0-bit is flipped, the fitness increases by 2 in expectation as the following bits may be set to 1.

By applying the progress bounds from Lemma 4.2 inductively, we derive the following bounds on the current fitness of the (1+1) EA after an arbitrary number of periods.

Lemma 4.3. *Consider the (1+1) EA optimising LO. Let a run of length αn^2 be split into $\alpha\psi$ periods of length n^2/ψ (for positive constants α and ψ). Define $\ell_{\chi,0} := 0$ and $u_{\chi,0} := \sqrt{n}$. Then for $i \leq \alpha$ there exist $u_{\chi,i+1}$ and $\ell_{\chi,i+1}$ with*

$$u_{\chi,i+1} = u_{\chi,i} + \frac{2\chi n}{\psi \exp\left(\frac{\chi u_{\chi,i}}{n}\right)} + o(n)$$

$$\ell_{\chi,i+1} = \ell_{\chi,i} + \frac{2\chi n}{\psi \exp\left(\frac{\chi u_{\chi,i+1}}{n}\right)} - o(n)$$

such that, with overwhelming probability, the following holds. At the end of period i, for $0 \leq i \leq \alpha$, the current fitness is in the interval $[\ell_{\chi,i}, u_{\chi,i}]$ or an optimum has been found, and throughout period i the fitness is in $[\ell_{\chi,i-1}, u_{\chi,i}]$ or an optimum has been found.

Since we do not have a closed form for the intervals derived in Lemma 4.3, we follow the approach in [7] and iterate them computationally in order to derive exact bounds on the fitness after a given number of iterations, using our Python tool mentioned earlier. We observe that the (1+1) EA makes, in expectation, a linear amount of progress during a period which consists of a quadratic number of iterations (as is the case in Lemma 4.3). This fact implies that we can check whether one configuration is ahead of another by computing the leading constant of the $\Theta(n)$ term in the fitness bounds from Lemma 4.3 and check whether the intervals are overlapping. If they are not overlapping then, w. o. p., one configuration is ahead of another by a linear amount. If n is large enough then the $o(n)$ terms from Lemma 4.3 can be ignored as the fitness is determined exclusively be the leading constants of the linear terms. We extract the relevant leading constants from the fitness bounds in the following lemma.

Lemma 4.4. *Let $c_{\ell,\chi,i}$ and $c_{u,\chi,i}$ denote the leading constants in the definition of $\ell_{\chi,i}$ and $u_{\chi,i}$ from Lemma 4.3, respectively (i. e., $\ell_{\chi,i} = c_{\ell,\chi,i} \cdot n - o(n)$ and $u_{\chi,i} = c_{u,\chi,i} \cdot n + o(n)$). Then $c_{u,\chi,i+1}$ and $c_{\ell,\chi,i+1}$ can be expressed using the recurrences $c_{\ell,\chi,0} = c_{u,\chi,0} = 0$,*

$$c_{u,\chi,i+1} = c_{u,\chi,i} + \frac{2\chi}{\psi \exp\left(\chi \cdot c_{u,\chi,i}\right)}$$

$$c_{\ell,\chi,i+1} = c_{\ell,\chi,i} + \frac{2\chi}{\psi \exp\left(\chi \cdot c_{u,\chi,i+1}\right)}.$$

We can now prove that the parameter landscape is unimodal.

Lemma 4.5. *For ParamRLS-F tuning the (1+1) EA for LO, the parameter values in the set $\{1.6, 1.7, \ldots, 2.9, 3.0\}$ are optimal for the ranges of κ given in Table 1, if n is large enough. Furthermore, the parameter landscape is unimodal for these cutoff times.*

Having proved that the parameter landscape is unimodal for the cutoff times given in Table 1, we now turn our attention to cutoff times $\kappa \geq 0.772076n^2$. We prove that, for cutoff times in this range, the parameter value $\chi = 1.6$ wins ParamRLS-F comparisons against any other configuration w. o. p., and again the parameter landscape is unimodal. In order to prove this result, it is first necessary to prove a helper lemma. This lemma takes two configurations and the distance between them and gives a condition which, if satisfied, implies that, w. o. p., the configuration which is closer to the optimum reaches the optimum before the configuration which is behind covers the initial distance between them. That is, the configuration which is closer to the optimum wins a ParamRLS-F comparison with overwhelming probability.

Lemma 4.6. *Assume that the fitness of the individuals in the $(1+1)_a$ EA and the $(1+1)_b$ EA optimising LO are contained in the intervals $[c_{\ell,a,i} \cdot n - o(n), c_{u,a,i} \cdot n + o(n)]$ and $[c_{\ell,b,i} \cdot n - o(n), c_{u,b,i} \cdot n + o(n)]$, respectively, as defined in Lemma 4.4. Assume that $c_{\ell,a,i} > c_{u,b,i}$ (that is, the $(1+1)_a$ EA is ahead of the $(1+1)_b$ EA by some linear distance). If*

$$\frac{\left(\frac{2b}{(c_{\ell,a,i} - c_{u,b,i}) \cdot \exp(b c_{\ell,b,i})} + \varepsilon\right)}{\left(\frac{2a}{(1 - c_{\ell,a,i}) \cdot \exp(a)}\right)} \leq 1$$

for some positive constant ε then, with overwhelming probability, the $(1+1)_a$ EA reaches the optimum before the $(1+1)_b$ EA has covered the initial distance between the two algorithms.

We now use this lemma to prove that the parameter landscape is unimodal for cutoff times $\kappa \geq 0.772076n^2$, with the configuration $\chi = 1.6$ as the optimum.

Lemma 4.7. *Consider ParamRLS-F tuning the (1+1) EA for LO with $\chi \in \{0.1, 0.2, \ldots, 2.9, 3.0\}$ (i. e. $d = 10, \phi = 3$). For all cutoff times $\kappa \geq 0.772076n^2$, and for the $(1+1)_a$ EA and the $(1+1)_b$ EA, with either $0.1 \leq b < a \leq 1.6$ or $1.6 \leq a < b \leq 3.0$, the $(1+1)_a$ EA wins a ParamRLS-F comparison against the $(1+1)_b$ EA w. o. p.*

Lemma 4.7 proves that, with overwhelming probability, for all cutoff times $\kappa \geq 0.772076n^2$ the configuration $\chi = 1.6$ wins a comparison in ParamRLS-F against any other configuration, and also that in a ParamRLS-F comparison between two configurations both on the same side of the configuration $\chi = 1.6$, the configuration closer to $\chi = 1.6$ wins the comparison w. o. p. Having proved that the parameter landscape is unimodal, we can now finally prove Theorem 4.1.

Proof of Theorem 4.1. We proceed in the same manner to the proof of Theorem 3.2. We pessimistically assume that the active parameter value is initialised as far away from the optimal parameter value as possible. The initial distance is clearly bounded by $d\phi$.

Given a comparison of a pair of configurations which are both on the same side of the optimal configuration, let us call the configuration with a value of χ closer to the optimum the 'better' configuration, and the other configuration the 'worse' configuration. Lemma 4.7 tells us that in a comparison between any pair of configurations which are both on the same side of the optimum, the better configuration wins w. o. p. Let us assume that the better configuration always beats the worse configuration. Since with the local search operator $\pm\{1\}$ the tuner will mutate the current configuration to one closer to the optimum, the tuner take a step towards the optimum with probability 1/2. With the remaining probability, the active parameter will remain the same. The expected time to move closer to the optimum is thus 2. Since the tuner needs to take at most $d\phi$ steps towards the optimal configuration in this case, this implies that $\mathrm{E}[T] \leq 2d\phi$. In the overwhelmingly unlikely event that the worse configuration wins a comparison then we restart the argument. Therefore, $\mathrm{E}[T] \leq 2d\phi + \exp(-\Omega(n^\varepsilon))$ for some positive constant ε from the definition of overwhelming probabilities.

Using a Chernoff bound to count the number of times that the tuner takes a step towards the optimal configuration proves that, with overwhelming probability, n^ε comparisons, for any positive constant ε, suffice for ParamRLS-F to set the active parameter value to the optimum. By the union bound, the value of the active parameter remains at the optimum w. o. p. once it has been found, since there are polynomially many comparisons. This implies that, w. o. p., the tuner returns the optimal configuration for the cutoff time if run for at least this many comparisons. □

4.2 Analysis of Optimisation Time-Based Comparisons when Tuning for LO

As in Section 3.2, we prove here that w. o. p. any configurator that uses the Optimisation time performance metric is unable to tune the (1+1) EA for LO if $\kappa \leq 0.772075n^2$.

Theorem 4.8. *Consider any configurator using the Optimisation time performance metric tuning the (1+1) EA for LO for any positive constant ϕ and discretisation factor d. If the cutoff time for each run is never allowed to exceed $\kappa \leq 0.772075n^2$ then, after any polynomial number of comparisons and runs per comparison, the configurator is blind.*

To prove Theorem 4.8 we first show that, w. o. p., no configuration of the (1+1) EA here reaches the optimum of LO within this cutoff time.

Lemma 4.9. *For all configurations $\chi \in \{0.1, 0.2, \ldots, 2.9, 3.0\}$, the (1+1)$_\chi$ EA does not reach the optimum of LO within $0.772075n^2$ iterations, w. o. p.*

Proof of Lemma 4.9. After 772075 periods of length $n^2/1000000$ (that is, with $\psi = 1000000$) we observe that the value $c_{u,\chi,i}$ for all $\chi \in \{0.1, 0.2, \ldots, 2.9, 3.0\}$ is less than 1. This implies that, with overwhelming probability, after $0.772075n^2$ iterations, no configuration has found the optimum of LO. □

Using Lemma 4.9, we are now able to prove Theorem 4.8.

Proof of Theorem 4.8. Since by Lemma 4.9 we know that, with overwhelming probability, no configuration finds the optimum of LO within $0.772075n^2$ iterations, then we argue that the result follows for the same reasons as in the proof of Theorem 3.5. □

5 CONCLUSIONS

Recent experimental work has provided evidence that the algorithm configuration search landscapes for various NP-hard problems are more benign than in worst-case scenarios. In this paper we rigorously proved that this is the case for the parameter landscape induced by the standard bit mutation (SBM) operator, used in evolutionary computation, for the optimisation of two standard benchmark problem classes, Ridge and LeadingOnes. In particular we have proved that the parameter landscape for both problems is largely unimodal. This effectively allows gradient-following algorithm configurators, including ParamRLS, to efficiently identify optimal mutation rates for both problems.

To the best of our knowledge, the only other time complexity analysis of algorithm configurators for specific problems is our earlier work [7], where we considered ParamRLS to tune the neighbourhood size of a more simple stochastic local search algorithm. This analysis pointed out that using the best identified fitness as performance measure (i.e., ParamRLS-F), rather than the optimisation time (i.e., ParamRLS-T), allows us to identify the optimal parameter value with considerably smaller cutoff times i.e., more efficiently. Our analysis reveals that this insight is also true for the much more sophisticated parameter landscape of the global mutation operator SBM. In particular, we proved for a wide range of cutoff times that ParamRLS-F tuning for LeadingOnes identifies that the smaller the cutoff time, the higher is the optimal mutation rate. For almost every given cutoff time, the optimal mutation rate for that cutoff time is returned efficiently, with overwhelming probability. Conversely, any algorithm configurator using optimisation time as performance metric is blind when using cutoff times that are smaller than the expected optimisation time of the optimal configuration.

Acknowledgements. This work was supported by the EPSRC under grant EP/M004252/1.

REFERENCES

[1] Thomas Bartz-Beielstein, Christian Lasarczyk, and Mike Preuß. 2010. The sequential parameter optimization toolbox. In *Experimental methods for the analysis of optimization algorithms*. Springer, 337–362.

[2] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. 2010. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. *Parallel Problem Solving from Nature, PPSN XI*, 1–10.

[3] Edmund K. Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. 2013. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society* 64, 12 (2013), 1695–1724.

[4] Benjamin Doerr, Carola Doerr, and Jing Yang. 2016. k-Bit Mutation with Self-Adjusting k Outperforms Standard Bit Mutation. In *Proc. of the International Conference on Parallel Problem Solving from Nature, LNCS 9921 (PPSN '16)*. Springer International Publishing, 824–834.

[5] Benjamin Doerr, Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2018. On the Runtime Analysis of Selection Hyper-Heuristics with Adaptive Learning Periods. In *Proceedings of the Genetic and Evolutionary Computation Conference 2018 (GECCO '18)*. ACM, 1015–1022.

[6] Stefan Droste, Thomas Jansen, Karsten Tinnefeld, and Ingo Wegener. 2002. A New Framework for the Valuation of Algorithms for Black-Box Optimization. In *Proceedings of the Seventh Workshop on Foundations of Genetic Algorithms*. Morgan Kaufmann, 253–270.

[7] George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. 2019. On the Impact of the Cutoff Time on the Performance of Algorithm Configurators. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '19)*. ACM, 907–915.

[8] George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. 2020. Analysis of the Performance of Algorithm Configurators for Search Heuristics with Global Mutation Operators. *arXiv preprint arXiv:2004.04519* (2020).

[9] Jun He and Xin Yao. 2001. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence* 127, 1 (2001), 57–85.

[10] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2009. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research* 36, 1 (2009), 267–306.

[11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. 2011. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*. Springer, 507–523.

[12] Thomas Jansen and Christine Zarges. 2014. Performance analysis of randomised search heuristics operating with a fixed budget. *Theoretical Computer Science* 545 (2014), 39–58.

[13] Ashiqur R. KhudaBukhsh, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2016. SATenstein: Automatically building local search SAT solvers from components. *Artificial Intelligence* 232 (2016), 20–42.

[14] Robert Kleinberg, Kevin Leyton-Brown, and Brendan Lucier. 2017. Efficiency Through Procrastination: Approximately Optimal Algorithm Configuration with Runtime Guarantees. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI'17)*. AAAI Press, 2023–2031.

[15] Robert Kleinberg, Kevin Leyton-Brown, Brendan Lucier, and Devon Graham. 2019. Procrastinating with Confidence: Near-Optimal, Anytime, Adaptive Algorithm Configuration. In *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 8881–8891.

[16] Timo Kötzing. 2016. Concentration of First Hitting Times Under Additive Drift. *Algorithmica* 75, 3 (2016), 490–506.

[17] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2019. Hyper-heuristics can Control the Neighbourhood Size of Randomized Local Search Optimally for Leading Ones (In press - Online). *Evolutionary Computation* (2019). https://doi.org/10.1162/evco_a_00258

[18] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2019. On the Time Complexity of Algorithm Selection Hyper-Heuristics for Multimodal Optimisation. In *Thirty-Third AAAI Conference on Artificial Intelligence (AAAI '19)*. AAAI Press, 2322–2329.

[19] Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2020. How the Duration of the Learning Period Affects the Performance of Random Gradient Selection Hyper-heuristics (In press). In *Proceedings of the thirty-fourth international AAAI conference on artificial Intelligence (AAAI-20)*. AAAI Press, –.

[20] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3 (2016), 43–58.

[21] Steven Minton. 1993. Integrating Heuristics for Constraint Satisfaction Problems: A Case Study. In *AAAI 1993*. AAAI Press, 120–126.

[22] Yasha Pushak and Holger H. Hoos. 2018. Algorithm Configuration Landscapes: More Benign Than Expected?. In *Parallel Problem Solving from Nature – PPSN XV*. Springer International Publishing, 271–283.

[23] Gellért Weisz, Andras Gyorgy, and Csaba Szepesvári. 2018. LeapsAndBounds: A Method for Approximately Optimal Algorithm Configuration. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 80. PMLR, 5257–5265.

[24] Gellért Weisz, Andras Gyorgy, and Csaba Szepesvári. 2019. CapsAndRuns: An improved method for approximately optimal algorithm configuration. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Vol. 97. PMLR, 6707–6715.