---

**Proceedings Paper:**
Hall, G.T., Oliveto, P.S. and Sudholt, D. orcid.org/0000-0001-6020-1646 (2020) Fast perturbative algorithm configurators. In: Bäck, T., Preuss, M., Deutz, A., Wang, H., Doerr, C. and Emm, M., (eds.) Parallel Problem Solving from Nature – PPSN XVI. International Conference on Parallel Problem Solving from Nature (PPSN 2020), 05-09 Sep 2020, Leiden, Netherlands. Lecture Notes in Computer Science, 12269 . Springer , pp. 19-32. ISBN 9783030581114

https://doi.org/10.1007/978-3-030-58112-1_2

---

# Fast Perturbative Algorithm Configurators

George T. Hall, Pietro S. Oliveto, and Dirk Sudholt

The University of Sheffield, Sheffield, United Kingdom
{gthall1,p.oliveto,d.sudholt}@sheffield.ac.uk

**Abstract.** Recent work has shown that the ParamRLS and ParamILS algorithm configurators can tune some simple randomised search heuristics for standard benchmark functions in linear expected time in the size of the parameter space. In this paper we prove a linear lower bound on the expected time to optimise any parameter tuning problem for ParamRLS, ParamILS as well as for larger classes of algorithm configurators. We propose a harmonic mutation operator for perturbative algorithm configurators that provably tunes single-parameter algorithms in polylogarithmic time for unimodal and approximately unimodal (i.e., non-smooth, rugged with an underlying gradient towards the optimum) parameter spaces. It is suitable as a general-purpose operator since even on worst-case (e.g., deceptive) landscapes it is only by at most a logarithmic factor slower than the default ones used by ParamRLS and ParamILS. An experimental analysis confirms the superiority of the approach in practice for a number of configuration scenarios, including ones involving more than one parameter.

**Keywords:** Parameter tuning · Algorithm configurators · Runtime analysis.

## 1 Introduction

Many algorithms are highly dependent on the values of their parameters, all of which have the potential to affect their performance substantially. It is therefore a challenging but important task to identify parameter values that lead to good performance for a class of problems. This task, called *algorithm configuration* or *parameter tuning*, was traditionally performed by hand: parameter values were updated manually and the performance of each *configuration* assessed, allowing the user to determine which parameter settings performed best. In recent years there has been an increase in popularity of automated *algorithm configurators* [13].

Examples of popular algorithm configurators are *ParamILS*, which uses iterated local search to traverse the *parameter space* (the space of possible configurations) [14]; *irace*, which evaluates a set of configurations in parallel and eliminates those which can be shown statistically to be performing poorly [20]; and *SMAC*, which uses surrogate models to reduce the number of configuration evaluations [15]. Despite their popularity, the foundational understanding of algorithm configurators remains limited. Key questions are still unanswered, such

as whether a configurator is able to identify (near) optimal parameter values, and, if so, the amount of time it requires to do so. While analyses of worst-case performance are available, as well as algorithms that provably perform better in worst-case scenarios [18, 23, 24, 19], the above questions are largely unanswered regarding the performance of the popular algorithm configurators used in practice for typical configuration scenarios.

Recently, the performance of ParamRLS and ParamILS was rigorously analysed for tuning simple single-parameter search heuristics for some standard benchmark problems from the literature. It was proved that they can efficiently tune the neighbourhood size $k$ of the randomised local search algorithm ($\text{RLS}_k$) for Ridge and OneMax [10] and the mutation rate of the simple (1+1) EA for Ridge and LeadingOnes [11]. The analyses, though, also reveal some weaknesses of the search operators used by the two algorithm configurators. The *$\ell$-step* mutation operator used by ParamRLS, which changes a parameter value to a neighbouring one at a distance of at most $\ell$, may either get stuck on local optima if the neighbourhood size $\ell$ is too small, or progress too slowly when far away from the optimal configuration. On the other hand, the mutation operator employed by ParamILS, that changes one parameter value uniformly at random, lacks the ability to efficiently fine-tune the current solution by searching locally around the identified parameter values. Indeed both algorithms require linear expected time in the number of parameter values to identify the optimal configurations for the studied unimodal or approximately unimodal parameter spaces induced by the target algorithms and benchmark functions [10, 11].

In this paper we propose a more robust mutation operator that samples a step size according to the harmonic distribution [6, 7]. The idea is to allow small mutation steps with sufficiently high probability to efficiently fine-tune good parameter values while, at the same time, enabling larger mutations that can help follow the general gradient from a macro perspective, e.g., by tunnelling through local optima. This search operator can be easily used in any perturbative algorithm configurator that maintains a set of best-found configurations and mutates them in search for better ones. Both ParamRLS and ParamILS fall into this large class of configurators.

We first prove that large classes of algorithm configurators, which include ParamRLS and ParamILS with their default mutation operators, require linear expected time in the number of possible configurations to optimise any parameter configuration landscape. Then we provide a rigorous proof that the harmonic search operator can identify the optimal parameter value of single-parameter target algorithms in polylogarithmic time if the parameter landscape is either unimodal or approximately unimodal (i.e., non-smooth, rugged landscapes with an underlying monotonically decreasing gradient towards the optimum). It is also robust as even on deceptive worst-case landscapes it is only by at most a logarithmic factor slower than the default operators of ParamRLS and ParamILS.

We complement the theory with an experimental analysis showing that both ParamRLS and ParamILS have a statistically significant smaller average optimisation time to identify the optimal configuration in single-parameter unimodal

**Algorithm 1** ParamRLS $(\mathcal{A}, \Theta, \Pi, \kappa, r)$. Adapted from [11].

1: $\theta \leftarrow$initial parameter value chosen uniformly at random
2: **while** termination condition not satisfied **do**
3: $\quad \theta' \leftarrow$ `mutate`$(\theta)$
4: $\quad \theta \leftarrow$ `better`$(\mathcal{A}, \theta, \theta', \pi, \kappa, r)$ {called `eval` in [11]}
5: **return** $\theta$

and approximately unimodal landscapes and for a well-studied MAX-SAT configuration scenario where two parameters have to be tuned. The latter result is in line with analyses of Pushak and Hoos that suggests that even in complex configuration scenarios (for instance state-of-the-art SAT, TSP, and MIP solvers), the parameter landscape is often not as complex as one might expect [22].

## 2 Preliminaries

*The ParamRLS Configurator.* ParamRLS is a simple theory-driven algorithm configurator defined in Algorithm 1 [11]. The algorithm chooses an initial configuration uniformly at random (u.a.r.) from the parameter space. In each iteration, a new configuration is generated by mutating the current solution. The obtained offspring replaces the parent if it performs better. By default, ParamRLS uses the *$\ell$-step* operator which selects a parameter and a step size $d \in \{1, \ldots, \ell\}$ both u.a.r. and then moves to a parameter value at distance[1] $+d$ or $-d$ (if feasible).

*The ParamILS Configurator.* ParamILS (Algorithm 2) is a more sophisticated iterated local search algorithm configurator [14]. In the initialisation step it selects $R$ configurations uniformly at random and picks the best performing one. In the iterative loop it performs an iterated local search (Algorithm 3) until a local optimum is reached, followed by a perturbation step where up to $s$ random parameters are perturbed u.a.r. A random restart occurs in each iteration with some probability $p_{\text{restart}}$. The default local search operator selects from the neighbourhood uniformly at random without replacement (thus we call this the *random* local search operator). The neighbourhood of a configuration contains all configurations that differ by exactly one parameter value.

*The Harmonic-step Operator.* The harmonic-step mutation operator selects a parameter uniformly at random and samples a step size $d$ according to the harmonic distribution. In particular, the probability of selecting a step size $d$ is $1/(d \cdot H_{\phi-1})$, where $H_m$ is the $m$-th harmonic number (i.e. $H_m = \sum_{k=1}^{m} \frac{1}{k}$) and $\phi$ is the range of possible parameter values. It returns the best parameter value

[1] Throughout this paper, we consider parameters from an interval of integers for simplicity, where the distance is the absolute difference between two integers. This is not a limitation: if parameters are given as a vector of real values $z_1, z_2, \ldots, z_\phi$, we may simply tune the index, which is an integer from $\{1, \ldots, \phi\}$. Then changing the parameter value means that we change the index of this value.

**Algorithm 2** ParamILS pseudocode, recreated from [14].

**Require:** Initial configuration $\theta_0 \in \Theta$, algorithm parameters $r, p_{\text{restart}}$, and $s$.
**Ensure:** Best parameter configuration $\theta$ found.

**for** $i = 1, \ldots, R$ **do**
  $\theta \leftarrow$ random $\theta \in \Theta$
  **if** `better`$(\theta, \theta_0)$ **then** $\theta_0 \leftarrow \theta$
$\theta_{\text{inc}} \leftarrow \theta_{\text{ils}} \leftarrow$ *IterativeFirstImprovement*$(\theta_0)$ {Algorithm 3}
**while not** *TerminationCriterion*() **do**
  $\theta \leftarrow \theta_{\text{ils}}$
  **for** $i = 1, \ldots, s$ **do** $\theta \leftarrow$ random $\theta' \in Nbh(\theta)$
  {*Nbh* contains all neighbours of a configuration}
  $\theta \leftarrow$ *IterativeFirstImprovement*$(\theta)$
  **if** `better`$(\theta, \theta_{\text{ils}})$ **then** $\theta_{\text{ils}} \leftarrow \theta$
  **if** `better`$(\theta_{\text{ils}}, \theta_{\text{inc}})$ **then** $\theta_{\text{inc}} \leftarrow \theta_{\text{ils}}$
  **with probability** $p_{\text{restart}}$ **do** $\theta_{\text{ils}} \leftarrow$ random $\theta \in \Theta$
**return** $\theta_{\text{inc}}$

**Algorithm 3** IterativeFirstImprovement($\theta$) procedure, adapted from [14].

**repeat**
  $\theta' \leftarrow \theta$
  **for all** $\theta'' \in$ *UndiscNbh*$(\theta')$ in randomised order **do**
    {*UndiscNbh* contains all undiscovered neighbours of a configuration}
    **if** `better`$(\theta'', \theta')$ **then** $\theta \leftarrow \theta''$; **break**
**until** $\theta' = \theta$
**return** $\theta$

at distance $\pm d$. This operator was originally designed to perform fast greedy random walks in one-dimensional domains [6] and was shown to perform better than the *1-step* and the random local search (as in ParamILS) operators for optimising the multi-valued OneMax problem [7]. We refer to ParamRLS using the Harmonic-step operator as ParamHS.

## 3 General Lower Bounds for Default Mutation Operators

To set a baseline for the performance gains obtained by ParamHS, we first show general lower bounds for algorithm configurators, including ParamRLS and ParamILS. Our results apply to a class of configurators described in Algorithm 4. We use a general framework to show that the poor performance of default mutation operators is not limited to particular configurators, and to identify which algorithm design aspects are the cause of poor performance.

We show that mutation operators that only change one parameter by a small amount, such as the $\ell$-step operator with constant $\ell$, lead to linear expected times in the number of parameter values (sum of all parameter ranges).

**Theorem 1.** *Consider a setting with $D$ parameters and ranges $\phi_1, \ldots, \phi_D \geq 2$ such that there is a unique optimal configuration. Let $M = \sum_{i=1}^{D} \phi_i$. Consider an*

**Algorithm 4** General scheme for algorithm configurators.

1: Initialise an incumbent configuration uniformly at random
2: **while** optimal configuration not found **do**
3: Pick a mutation operator according to the history of past evaluations.
4: Apply the chosen mutation operator.
5: Apply selection to choose new configuration from the incumbent configuration and the mutated one.

*algorithm configurator $\mathcal{A}$ implementing the scheme of Algorithm 4 whose mutation operator only changes a single parameter and does so by at most a constant absolute value (e.g. ParamRLS with local search operator $\pm\{\ell\}$ for constant $\ell$). Then $\mathcal{A}$ takes time $\Omega(M)$ in expectation to find the optimal configuration.*

*Proof.* Consider the $L_1$ distance of the current configuration $x = (x_1, \dots, x_D)$ from the optimal one $\text{opt} = (\text{opt}_1, \dots, \text{opt}_D)$: $\sum_{i=1}^{D} |x_i - \text{opt}_i|$. For every parameter $i$, the expected distance between the uniform random initial configuration and $\text{opt}_i$ is minimised if $\text{opt}_i$ is at the centre of the parameter range. Then, for odd $\phi_i$, there are two configurations at distances $1, 2, \dots, (\phi_i - 1)/2$ from $\text{opt}_i$, each being chosen with probability $1/\phi_i$. The expected distance is thus at least $1/\phi_i \cdot \sum_{j=1}^{(\phi_i-1)/2} 2j = (\phi_i - 1)(\phi_i + 1)/(4\phi_i) = (\phi_i - 1/\phi_i)/4 \geq \phi_i/8$. For even $\phi_i$, the expectation is at least $\phi_i/4$. By linearity of expectation, the expected initial distance is at least $\sum_{i=1}^{D} \phi_i/8 \geq M/8$. Every mutation can only decrease the distance by $O(1)$, hence the expected time is bounded by $(M/8)/O(1) = \Omega(M)$. □

The same lower bound also applies if the mutation operator chooses a value uniformly at random (with or without replacement), as is done in ParamILS.

**Theorem 2.** *Consider a setting with $D$ parameters and ranges $\phi_1, \dots, \phi_D \geq 2$ such that there is a unique optimal configuration. Let $M = \sum_{i=1}^{D} \phi_i$. Consider an algorithm configurator $\mathcal{A}$ implementing the scheme of Algorithm 4 whose mutation operator only changes a single parameter and does so by choosing a new value uniformly at random (possibly excluding values previously evaluated). Then $\mathcal{A}$ takes time $\Omega(M)$ in expectation to find the optimal configuration.*

*Proof.* Let $T_i$ be the number of times that parameter $i$ is mutated (including the initial step) before it attains its value in the optimal configuration. After $j - 1$ steps in which parameter $i$ is mutated, at most $j$ parameter values have been evaluated (including the initial value). The best case is that $\mathcal{A}$ always excludes previous values, which corresponds to a complete enumeration of the $\phi_i$ possible values in random order. Since every step of this enumeration has a probability of $1/\phi_i$ of finding the optimal value, the expected time spent on parameter $i$ is $\mathrm{E}(T_i) \geq \sum_{j=0}^{\phi_i - 1} j/\phi_i = (\phi_i - 1)/2$. The total expected time is at least $\sum_{i=1}^{D} \mathrm{E}(T_i) - D + 1$ as the initial step contributes to all $T_i$ and each following step only contributes to one value $T_i$. Noting $\sum_{i=1}^{D} \mathrm{E}(T_i) - D + 1 = \sum_{i=1}^{D} (\phi_i - 1)/2 - D/2 + 1 \geq M/4$ (as $\phi_i \geq 2$ for all $i$) proves the claim. □

ParamILS is not covered directly by Theorem 2 as it uses random sampling during the initialisation that affects all parameters. However, it is easy to show that the same lower bound also applies to ParamILS.

**Theorem 3.** *Consider a setting with $D$ parameters and ranges $\phi_1, \ldots, \phi_D \geq 2$ such that there is a unique optimal configuration. Let $M = \sum_{i=1}^{D} \phi_i$. Then ParamILS takes time $\Omega(M)$ in expectation to find the optimal configuration.*

*Proof.* Recall that ParamILS first evaluates $R$ random configurations. If $R \geq M/2$ then the probability of finding the optimum during the first $M/2$ random samples is at most $M/2 \cdot \prod_{i=1}^{D} 1/\phi_i \leq 1/2$ since $M = \sum_{i=1}^{D} \phi_i \leq \prod_{i=1}^{D} \phi_i$. Hence the expected time is at least $1/2 \cdot M/2 = M/4$. If $R < M/2$ then with probability at least $1/2$ ParamILS does not find the optimum during the $R$ random steps and starts the IterativeFirstImprovement procedure with a configuration $\theta_0$. This procedure scans the neighbourhood of $\theta_0$, which is all configurations that differ in one parameter; the number of these is $\sum_{i=1}^{D}(\phi_i - 1) = M - D$. If the global optimum is not among these, it is not found in these $M - D$ steps. Otherwise, the neighbourhood is scanned in random order and the expected number of steps is $(M - D - 1)/2$ as in the proof of Theorem 2. In both cases, the expected time is at least $(M - D - 1)/4 \geq M/16$ (as $M \geq 2D$). □

## 4 Performance of the Harmonic Search Operator

In the setting of Theorem 1, mutation lacks the ability to explore the search space quickly, whereas in the setting of Theorems 2 and 3, mutation lacks the ability to search locally. The harmonic search operator is able to do both. It is able to explore the space, but smaller steps are made with a higher probability, enabling the search to exploit gradients in the parameter landscape.

For simplicity and lack of space we only consider configuring one parameter with a range of $\phi$ (where the bounds from Theorems 1–3 simplify to $\Omega(\phi)$), however the operator improves performance in settings with multiple parameters in the same way. We show that ParamHS is robust in a sense that it performs well on all landscapes (with only a small overhead in the worst case, compared to the lower bounds from Theorem 1–3), and it performs extremely well on functions that are unimodal or have an underlying gradient that is close to being unimodal.

To capture the existence of underlying gradients and functions that are unimodal to some degree, we introduce a notion of approximate unimodality.

**Definition 1.** *Call a function $f$ on $\{1, \ldots, m\}$ $(\alpha, \beta)$-approximately unimodal for parameters $\alpha \geq 1$ and $1 \leq \beta \leq m$ if for all positions $x$ with distance $\beta \leq i \leq m$ from the optimum and all positions $y$ with distance $j > \alpha i$ to the optimum we have $f(x) < f(y)$.*

Intuitively, this means that only configurations with distance to the optimal one that is by a factor of $\alpha$ larger than that of the current configuration can be better. This property only needs to hold for configurations with distance to the

optimum $i$ with $\beta \leq i \leq m$, to account for landscapes that do not show a clear gradient close to the optimum.

Note that a $(1,1)$-approximately unimodal function is unimodal and a $(1,\beta)$-approximately unimodal function is unimodal within the states $\{\beta,\ldots,m\}$. Also note that all functions are $(1,m)$-approximately unimodal.

The following performance guarantees for ParamHS show that it is efficient on all functions and very efficient on functions that are close to unimodal.

**Theorem 4.** *Consider ParamHS configuring an algorithm with a single parameter having $\phi$ values and a unique global optimum. If the parameter landscape is $(\alpha,\beta)$-approximately unimodal then the expected number of calls to* `better()` *before the optimal parameter value is sampled is at most*

$$4\alpha H_{\phi-1}\log(\phi) + 4\alpha\beta H_{\phi-1} = O(\alpha\log^2(\phi) + \alpha\beta\log\phi),$$

*where $H_{\phi-1}$ is the $(\phi-1)$-th harmonic number (i.e. $\sum_{i=1}^{\phi-1}\frac{1}{i}$).*

**Corollary 1.** *In the setting of Theorem 4,*

*(a) every unimodal parameter landscape yields a bound of $O(\log^2\phi)$.*
*(b) for every parameter landscape, a general upper bound of $O(\phi\log\phi)$ applies.*

Hence ParamHS is far more efficient than the $\Omega(\phi)$ lower bound for general classes of tuners (Theorems 1–3) on approximately unimodal landscapes and is guaranteed never to be worse than default operators by more than a $\log\phi$ factor.

*Proof of Theorem 4.* Let $f(i)$ describe the performance of the configuration with the $i$-th largest parameter value. Then $f$ is $(\alpha,\beta)$-approximately unimodal and we are interested in the time required to locate its minimum.

Let $d_t$ denote the current distance to the optimum and note that $d_0 \leq \phi$. Let $d_t^*$ denote the smallest distance to the optimum seen so far, that is, $d_t^* = \min_{t'\leq t} d_{t'}$. Note that $d_t^*$ is non-increasing over time. Since ParamHS does not accept any worsenings, $f(d_t) \leq f(d_t^*)$.

If $d_t^* \geq \beta$ then by the approximate unimodality assumption, for all $j > \alpha d_t^*$, $f(j) > f(d_t^*) \geq f(d_t)$, that is, all points at distance larger than $\alpha d_t^*$ have a worse fitness than the current position and will never be visited.

Now assume that $d_t^* \geq 2\beta$. We estimate the expected time to reach a position with distance at most $\lfloor d_t^*/2\rfloor$ to the optimum. This includes all points that have distance $i$ to the global optimum, for $0 \leq i \leq \lfloor d_t^*/2\rfloor$, and distance $d_t - i$ to the current position. The probability of jumping to one of these positions is at least

$$\sum_{i=0}^{\lfloor d_t^*/2\rfloor}\frac{1}{2(d_t-i)H_{\phi-1}} \geq \sum_{i=0}^{\lfloor d_t^*/2\rfloor}\frac{1}{2d_tH_{\phi-1}} \geq \frac{d_t^*}{4d_tH_{\phi-1}} \geq \frac{d_t^*}{4\alpha d_t^*H_{\phi-1}} = \frac{1}{4\alpha H_{\phi-1}}.$$

Hence, the expected half time for $d_t^*$ is at most $4\alpha H_{\phi-1}$ and the expected time to reach $d_t^* < 2\beta$ is at most $4\alpha H_{\phi-1}\log\phi$.

Once $d_t^* < 2\beta$, the probability of jumping directly to the optimum is at least $\frac{1}{2d_tH_{\phi-1}} \geq \frac{1}{2\alpha d_t^*H_{\phi-1}} \geq \frac{1}{4\alpha\beta H_{\phi-1}}$ and the expected time to reach the optimum is at most $4\alpha\beta H_{\phi-1}$. Adding the above two times and using the well-known fact that $H_{\phi-1} = O(\log\phi)$ yields the claim. □

## 5 Experimental Analysis

We have proved that, given some assumptions about the parameter landscape, it is beneficial to use the harmonic-step operator instead of the default operators used in ParamRLS and ParamILS. In this section, we verify experimentally that these theoretical results are meaningful beyond parameter landscapes assumed to be (approximately) unimodal.

We investigated the impact of using the harmonic-step operator on the time taken for ParamRLS and ParamILS to identify the optimal configuration (or in one case a set of near-optimal configurations) in different configuration scenarios. Note that ParamRLS using this operator is equivalent to ParamHS. We analysed the number of configuration comparisons (that is, calls to the `better()` procedure present in both ParamRLS and ParamILS) required for the configurators to identify the optimal mutation rate (the optimal value $\chi$ in the mutation rate $\chi/n$) for the (1+1) EA optimising Ridge and the (1+1) EA optimising LeadingOnes as in [11] and identifying the optimal neighbourhood size $k$ (the number of bits flipped during mutation) for $\text{RLS}_k$ optimising OneMax as in [10]. Finally, we considered optimising two parameters of the SAT solver SAPS optimising MAX-SAT [17], searching for one of the five best-performing configurations found during an exhaustive search of the parameter space.

In the first two configuration scenarios, with probability $1 - 2^{-\Omega(n^\varepsilon)}$, the configurator can identify that a neighbouring parameter value is better, hence the landscape is unimodal [11] (see Figures 1a and 1b). In such landscapes, we expect the harmonic-step operator to perform well. In the third scenario, the parameter landscape is *not* unimodal (see Figure 1c: $k = 2c + 1$ outperforms $k = 2c$), but it is (2,1)-approximately unimodal with respect to the expected fitness (as for all $k$, the parameter value $k$ outperforms all parameter values $k' > 2k$) both empirically (Figure 1c) and theoretically [8]. In the fourth scenario, the parameter landscape is more complex since we configure two parameters, but it still appears to be approximately unimodal (see Figure 1d).

### 5.1 Experimental Setup

In all scenarios we measured the number of calls to the `better()` procedure before the optimal configuration (or a set of near-optimal configurations in the scenario configuring SAPS) is first sampled. We varied the size of the parameter space to investigate how the performance of the mutation operators (i.e. $\ell$-step, random, and harmonic-step) depends on the size of the parameter space.

For ParamILS, the BasicILS variant was used. That is, each call to `better()` resulted in the two competing configurations both being run the same, set number of times. For each size of the parameter spaces, the experiment was repeated 200 times and the mean number of calls to `better()` was recorded. For the MAX-SAT scenario 500 repetitions were used to account for the increased complexity of the configuration scenario. The cutoff time $\kappa$ (the number of iterations for which each configuration is executed for each run in a comparison) varied with the choice of problem class. A fitness-based performance metric was

used, as recommended in [10, 11], in which the winner of a comparison is the configuration which achieves the highest mean fitness in $r$ runs each lasting $\kappa$ iterations. In each run, both configurators were initialised uniformly at random. We set $R = 0$ in ParamILS since preliminary experiments indicated that initial random sampling was harmful in the configuration scenarios considered here.

*Benchmark functions* For RIDGE, LEADINGONES and ONEMAX, we used $n = 50$ and 1500 runs per configuration comparison (i.e. $r = 1500$). For RIDGE, we used a cutoff time of $\kappa = 2500$. The value of $\ell$ in the $\ell$-step operator was set to $\ell = 1$. The first parameter space that we considered was $\chi \in \{0.5, 1.0, \ldots, 4.5, 5.0\}$, where $\chi/n$ is the mutation rate and $\chi = 1$ is optimal for RIDGE [11]. We increased the size of the parameter space by adding the next five largest configurations (each increasing by 0.5) until the parameter space $\{0.5, \ldots, 25.0\}$ was reached. Following [11], for RIDGE, the (1+1) EA was initialised at the start of the ridge, in order to focus on the search on the ridge (as opposed to the initial approach to the ridge, for which the optimal mutation rate may be different from $1/n$).

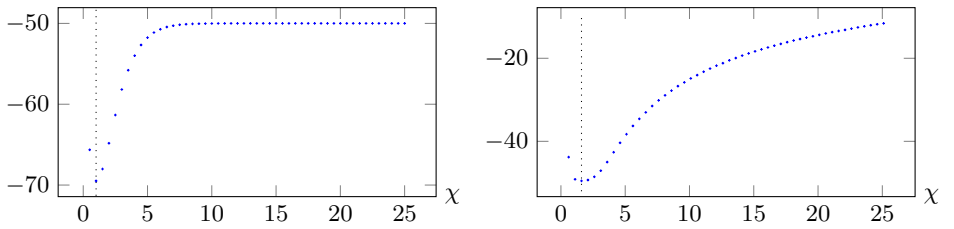
When configuring the mutation rate $\chi/n$ of the (1+1) EA for LEADINGONES, we initialised the individual u.a.r. and used $\kappa = 2500$ and $\ell = 1$. The size of the parameter space was increased in the same way as in the RIDGE experiments, and the initial parameter space was $\chi \in \{0.6, 1.1, \ldots, 4.6, 5.1\}$ as the optimal value for $\chi$ is approximately 1.6 [1, 11]. The final parameter space was $\{0.6, \ldots, 25.1\}$

When configuring the neighbourhood size of $\text{RLS}_k$ for ONEMAX, we initialised the individual u.a.r. and set $\kappa = 200$. The initial parameter space was $\{1, 2, \ldots, 9, 10\}$, where $k = 1$ is the optimal parameter [10], and the next five largest integers were added until $\{1, 2, \ldots, 49, 50\}$ was reached. Since this parameter landscape is only approximately unimodal, we set $\ell = 2$ (as recommended in [10]: $\ell = 1$ would fail to reach the optimal value $k = 1$ unless initialised there).

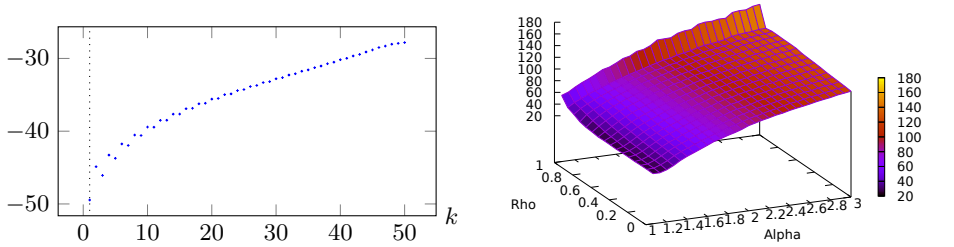
*SAPS for MAX-SAT* We considered tuning two parameters of SAPS – $\alpha$ and $\rho$ – for ten instances[2] of the circuit-fuzz problem set (available in AClib [16]). Due to the complexity of the MAX-SAT problem class it was no longer obvious which configurations can be considered optimal. Therefore we conducted an exhaustive search of the parameter space in order to identify configurations that perform well. We did so by running the validation procedure in ParamILS for each configuration with $\alpha \in \{\frac{16}{15}, \frac{17}{15}, \ldots, \frac{44}{15}, \frac{45}{15}\}$ and $\rho \in \{0, \frac{1}{15}, \ldots, \frac{14}{15}, 1\}$. Each configuration was evaluated 2000 times on each of the ten considered circuit-fuzz problem instances. In each evaluation, the cutoff time was $10,000$ iterations and the quality of a configuration was the number of satisfied clauses. We selected the set of the five best-performing configurations to be the target.

Since it was not feasible to compute the quality of a configuration each time it was evaluated in a tuner, we instead took the average fitness values generated during the initial evaluation of the parameter landscape to be the fitness of each configuration. As these runs were repeated many times we believe they provide an accurate approximation of the fitness values of the configurations.

[2] Problem instances number 78, 535, 581, 582, 6593, 6965, 8669, 9659, 16905, 16079.




(a) (1+1) EA and RIDGE, $\kappa = 2500$ (b) (1+1) EA and LEADINGONES, $\kappa = 2500$




(c) RLS$_k$ and ONEMAX, $\kappa = 200$ (d) SAPS and MAX-SAT, $\kappa = 10000$

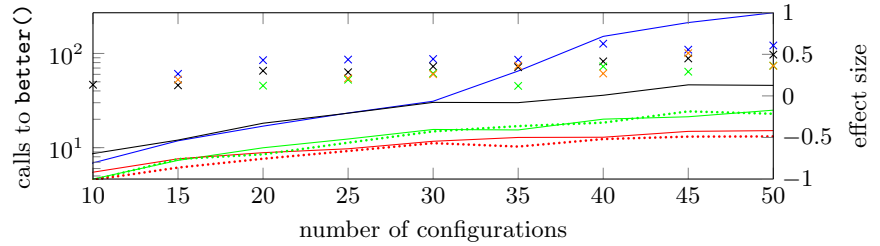
Fig. 1: (a),(b),(c): Mean fitness of the individual in the algorithms with $n = 50$, averaged over 10,000 runs for each parameter value, multiplied by $-1$ to obtain a minimisation problem. The dotted line indicates the optimal configuration for each scenario. (d): The parameter landscape for SAPS in terms of $\alpha$ and $\rho$ computed for a set of ten SAT instances from the circuit-fuzz dataset. In all figures lower values are better.

In this experiment, we kept the range of values of $\rho$ as the set $\{0, \frac{1}{15}, \ldots, \frac{14}{15}, 1\}$ and the value of the two other parameters of SAPS as $ps = 0.05$ and $wp = 0.01$ (their default values). We then increased the size of the set of possible values of $\alpha$. The initial range for $\alpha$ was the set $\{\frac{16}{15}, \frac{17}{15}, \frac{18}{15}\}$, which contains all five best-performing configurations. We then generated larger parameter spaces by adding a new value to the set of values for $\alpha$ until the set $\{\frac{16}{15}, \ldots, \frac{45}{15}\}$ was reached.

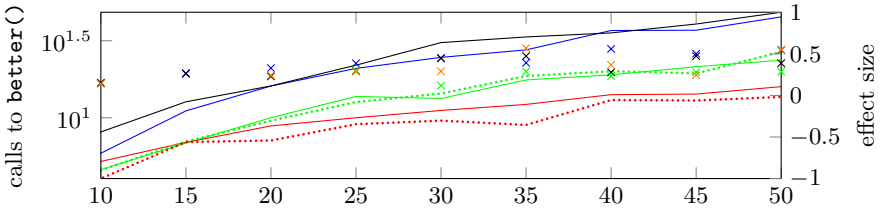
### 5.2 Results

The results from configuring benchmark functions are shown in Figures 2a, 2b, and 2c. Green lines indicate the random search operator (without replacement), black lines indicate the random search operator (with replacement), blue lines indicate the $\ell$-step operator, and red lines indicate the harmonic-step operator. Solid lines correspond to ParamRLS and dotted lines to ParamILS.

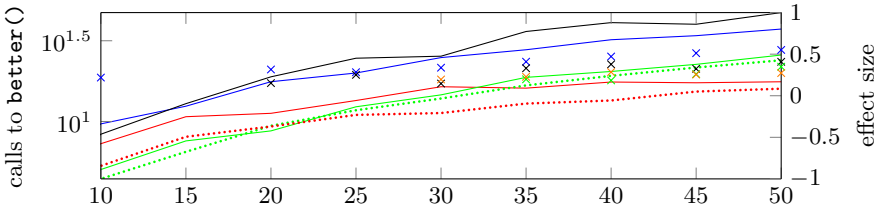
In each configuration scenario, and for both configurators, the harmonic-step operator located the optimal configuration faster than both the $\ell$-step and random operators. For both configurators, the polylogarithmic growth of the



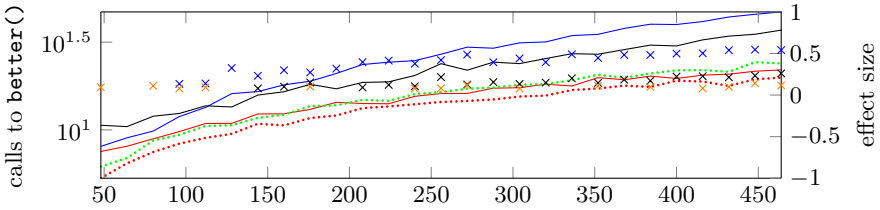

(a) Configuring the (1+1) EA for RIDGE with $\kappa = 2500$ and $r = 1500$.




(b) Configuring the (1+1) EA for LEADINGONES with $\kappa = 2500$ and $r = 1500$.




(c) Configuring $\text{RLS}_k$ for ONEMAX with $\kappa = 200$ and $r = 1500$.




(d) Configuring the $\alpha$ and $\rho$ parameters of SAPS.

Fig. 2: Mean number of calls to `better()` before sampling the optimal configuration. Green lines indicate the random search operator (without replacement), black lines indicate the random search operator (with replacement), blue lines indicate the $\ell$-step operator, and red lines indicate the harmonic-step operator. Solid lines correspond to ParamRLS and dotted lines to ParamILS. Crosses show effect size of difference at points where statistically significant for ParamHS versus: $\ell$-step (blue); random (without replacement) (green); random (with replacement) (black); and harmonic-step ParamILS vs. default ParamILS (orange).

time taken to locate the optimal configuration of the harmonic-step operator can be seen, compared to the linear growth of the time taken by the $\ell$-step and random local search operators. The difference between the operators is more pronounced when there is a plateau of neighbouring configurations all exhibiting the same performance (as in RIDGE). We also verified that these improvements in performance occur also if few runs per comparison are used.

Similar benefits from using the harmonic-step operator can be seen in the results for configuring SAPS for MAX-SAT. Figure 2d shows that it is faster to locate a near-optimal configuration for SAPS when using the harmonic step operator than when using the other operators.

Figure 2 also shows crosses where the difference between the performance of the harmonic-step operator and the other operators is statistically significant at a significance level of 0.95 (according to a two-tailed Mann-Whitney U test [21]). Their position reflects the effect size (in terms of Cliff's delta [2]) of this comparison (values closer to 1 indicate a larger difference). Orange crosses show the difference between ParamILS using harmonic-step and that using random (without replacement). The differences between ParamHS and ParamRLS using $\ell$-step, random (without replacement) and random (with replacement) are shown by blue, green, and black crosses, respectively. In every configuration scenario, for the larger parameter space sizes almost all comparisons with all other operators were statistically significant.

## 6 Conclusions

*Fast* mutation operators, that aim to balance the number of large and small mutations, are gaining momentum in evolutionary computation [9, 3, 5, 4]. Concerning algorithm configuration we demonstrated that ParamRLS and ParamILS benefit from replacing their default mutation operators with one that uses a harmonic distribution. We proved considerable asymptotic speed-ups for smooth unimodal and approximately unimodal (i.e., rugged) parameter landscapes, while in the worst case (e.g., for deceptive landscapes) the proposed modification may only slow down the algorithm by at most logarithmic factor. We verified experimentally that this speed-up occurs in practice for benchmark parameter landscapes that are known to be unimodal and approximately unimodal, as well as for tuning a MAX-SAT solver for a well-studied benchmark set. Indeed other recent experimental work has suggested that the search landscape of algorithm configurations may be simpler than expected, often being unimodal or even convex [12, 22]. We believe that this is the first work that has rigorously shown how to provably achieve faster algorithm configurators by exploiting the envisaged parameter landscape, while being only slightly slower if it was to be considerably different. Future theoretical work should estimate the performance of the harmonic mutation operator on larger parameter configuration problem classes, while empirical work should assess the performance of the operator for more sophisticated configurators operating in real-world configuration scenarios.

*Acknowledgements* This work was supported by the EPSRC (EP/M004252/1).

## References


1. Süntje Böttcher, Benjamin Doerr, and Frank Neumann. Optimal fixed and adaptive mutation rates for the LeadingOnes problem. In *Parallel Problem Solving from Nature – PPSN XI*, pages 1–10. Springer Berlin Heidelberg, 2010.
2. Norman Cliff. Dominance statistics: Ordinal analyses to answer ordinal questions. *Psychological bulletin*, 114(3):494, 1993.
3. Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. Fast artificial immune systems. In *Parallel Problem Solving from Nature – PPSN XV*, pages 67–78, 2018.
4. Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. Artificial immune systems can find arbitrarily good approximations for the NP-hard number partitioning problem. *Artificial Intelligence*, 247:180–196, 2019.
5. Dogan Corus, Pietro S. Oliveto, and Donya Yazdani. When hypermutations and ageing enable artificial immune systems to outperform evolutionary algorithms. *Theoretical Computer Science*, 832:166–185, 2020.
6. Martin Dietzfelbinger, Jonathan E. Rowe, Ingo Wegener, and Philipp Woelfel. Precision, local search and unimodal functions. *Algorithmica*, 59(3):301–322, 2011.
7. Benjamin Doerr, Carola Doerr, and Timo Kötzing. Static and self-adjusting mutation strengths for multi-valued decision variables. *Algorithmica*, 80:1732–1768, 2018.
8. Benjamin Doerr, Carola Doerr, and Jing Yang. Optimal parameter choices via precise black-box analysis. *Theoretical Computer Science*, 801:1–34, 2020.
9. Benjamin Doerr, Huu Phuoc Le, Régis Makhmara, and Ta Duy Nguyen. Fast genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017*, pages 777–784. ACM, 2017.
10. George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. On the impact of the cutoff time on the performance of algorithm configurators. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019*, pages 907–915. ACM, 2019.
11. George T. Hall, Pietro S. Oliveto, and Dirk Sudholt. Analysis of the performance of algorithm configurators for search heuristics with global mutation operators. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2020*. ACM, 2020. To appear.
12. Kyle R. Harrison, Beatrice M. Ombuki-Berman, and Andries P. Engelbrecht. The parameter configuration landscape: A case study on particle swarm optimization. In *IEEE Congress on Evolutionary Computation, CEC 2019*, pages 808–814. IEEE, 2019.
13. Changwu Huang, Yuanxiang Li, and Xin Yao. A survey of automatic parameter tuning methods for metaheuristics. *IEEE Transactions on Evolutionary Computation*, 24(2):201–216, 2020.
14. Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36(1):267–306, 2009.
15. Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization – 5th International Conference, LION 5*, pages 507–523. Springer, 2011.
16. Frank Hutter, Manuel López-Ibáñez, Chris Fawcett, Marius Lindauer, Holger H. Hoos, Kevin Leyton-Brown, and Thomas Stützle. AClib: A benchmark library for algorithm configuration. In *Learning and Intelligent Optimization – 8th International Conference, LION 8*, pages 36–40. Springer, 2014.

17. Frank Hutter, Dave A. D. Tompkins, and Holger H. Hoos. Scaling and probabilistic smoothing: Efficient dynamic local search for SAT. In *International Conference on Principles and Practice of Constraint Programming*, pages 233–248. Springer, 2002.
18. Robert Kleinberg, Kevin Leyton-Brown, and Brendan Lucier. Efficiency through procrastination: Approximately optimal algorithm configuration with runtime guarantees. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017*, pages 2023–2031. AAAI Press, 2017.
19. Robert Kleinberg, Kevin Leyton-Brown, Brendan Lucier, and Devon Graham. Procrastinating with confidence: Near-optimal, anytime, adaptive algorithm configuration. In *Advances in Neural Information Processing Systems 32, NeurIPS 2019*, pages 8881–8891. Curran Associates Inc., 2019.
20. Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.
21. Henry B. Mann and Donald R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, pages 50–60, 1947.
22. Yasha Pushak and Holger H. Hoos. Algorithm configuration landscapes: More benign than expected? In *Parallel Problem Solving from Nature – PPSN XV*, pages 271–283. Springer, 2018.
23. Gellért Weisz, András György, and Csaba Szepesvári. LeapsAndBounds: A method for approximately optimal algorithm configuration. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, pages 5254–5262. PMLR, 2018.
24. Gellért Weisz, András György, and Csaba Szepesvári. CapsAndRuns: An improved method for approximately optimal algorithm configuration. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019*, pages 6707–6715. PMLR, 2019.