On the Runtime Analysis of Selection Hyper-Heuristics with Adaptive Learning Periods

Benjamin Doerr
Laboratoire d’Informatique (LIX)
École Polytechnique
Palaiseau, France

Pietro S. Oliveto
Department of Computer Science
University of Sheffield
Sheffield, United Kingdom

Andrei Lissovoi
Department of Computer Science
University of Sheffield
Sheffield, United Kingdom

John Alasdair Warwicker
Department of Computer Science
University of Sheffield
Sheffield, United Kingdom

ABSTRACT
Selection hyper-heuristics are randomised optimisation techniques that select from a set of low-level heuristics which one should be applied in the next step of the optimisation process. Recently it has been proven that a Random Gradient hyper-heuristic optimises the LEADINGONES benchmark function in the best runtime achievable with any combination of its low-level heuristics, up to lower order terms. To achieve this runtime, the learning period $\tau$, used to evaluate the performance of the currently chosen heuristic, should be set appropriately, i.e., super-linear in the problem size but not excessively larger. In this paper we automate the hyper-heuristic further by allowing it to self-adjust the learning period $\tau$ during the run. To achieve this we equip the algorithm with a simple self-adjusting mechanism, called $1-\sigma(1)$ rule, inspired by the 1/5 rule traditionally used in continuous optimisation. We rigorously prove that the resulting hyper-heuristic solves LEADINGONES in optimal runtime by automatically adapting $\tau$ and achieving a $1-\sigma(1)$ ratio of the desired behaviour. Complementary experiments for realistic problem sizes show the value of $\tau$ adapting as desired and that the hyper-heuristic with adaptive learning period outperforms the hyper-heuristic with fixed learning periods.

CCS CONCEPTS
- Theory of computation → Theory of randomized search heuristics;

KEYWORDS
Theory, Running time analysis, Selection hyper-heuristics, Parameter adaptation.

1 INTRODUCTION
Randomised search heuristics (RSH) have been successfully applied to numerous real world optimisation problems. While their main strength is their problem independence (they can be applied to virtually any optimisation problem), it is well understood that each RSH will only be efficient on some classes of problems and not on others [24]. The tedious and time consuming trial and error phases, used to determine which heuristic and parameter settings lead to good solutions for the problem at hand, are one of the main difficulties in the application of RSH.

The fields of automated algorithm design and hyper-heuristics (HHs) aim to automatically evolve the algorithm and related parameters for the problem, rather than making such choices manually. Successful applications of HHs have been reported for a variety of practical problems, including scheduling [6, 7], timetabling [23] and vehicle routing [3] (see [5] for an extended survey). However, a rigorous and foundational understanding of the behaviour and performance of HHs is largely lacking.

Recently some preliminary theoretical studies of selection HHs have appeared [1, 20]. Selection HHs consist of high level mechanisms which select from a set of low-level heuristics which one should be applied in the next step of the optimisation process. A comparative analysis [21] for the LEADINGONES benchmark problem revealed that the simple Permutation, Greedy and Random Gradient selection HHs from the literature [6, 7] have the same performance as that of simply choosing a random operator at each step, i.e., they all require $O((\ln(3)/2)n^2 + o(n^2))$ fitness evaluations in expectation to optimise LEADINGONES when choosing between Random Local Search with neighbourhood size 1 (RLS1) and neighbourhood size 2 (RLS2).

The reason for the disappointing performance was that the simple HHs at most consider whether an operator was successful in the previous step to decide whether to apply it again in the following generation [21]. For most discrete optimisation problems and most randomised search heuristics, in a typical run the probability that an operator is successful in a single iteration is very low. As a result, even if the current best operator is chosen, it is very unlikely that the simple HHs will continue using it, hence it will not exploit the better operator. Lissovoi et al. [21] generalised the Random Gradient and the Greedy HH to measure the performance of operators over a fixed period of time $\tau$, rather than in a single iteration,
and rigorously proved that both HHs outperform their constituent low-level heuristics (i.e., RLS1 and RLS2) for LeadingOnes provided that the learning period \( \tau \) is large enough. In a subsequent analysis [22], they proved that for learning periods \( \tau = o(n) \) and \( \tau \leq (1/2 - \varepsilon)n \ln n \), \( \varepsilon \) a constant, the generalised Random Gradient HH has the best possible runtime achievable by using the two low-level operators, namely \( 1 + (\ln(2))n^2 + o(n^2) \). The set of low-level heuristics is extended to allow to choose from more operators (i.e., any RLS_i operator may be chosen for \( 1 \leq i \leq k \) and \( k = o(n) \), then the generalised Random Gradient HH is faster, up to lower order terms, than the best possible runtime achievable by using \( k - 1 \) operators if \( \tau \) is set appropriately.

In this paper, we further extend the power of the generalised HHs by allowing the learning period \( \tau \) to automatically adapt its length throughout the run. Apart from removing the burden of having to identify an appropriate value for the learning period, the best parameter value may change at different stages of the optimisation process, hence no fixed value of \( \tau \) may be optimal.

We equip the generalised HH with an update scheme inspired by the 1/5 rule, where the value of \( \tau \) is decreased by a multiplicative factor if the currently selected low-level heuristic proves successful within a period of \( \tau \) steps, and increased by a multiplicative factor if the low-level heuristic proves unsuccessful. The factors are chosen such that \( \sigma \) decreases counteract exactly one increase, reflecting, in essence, a \( 1 - 1/\sigma \) rule. To simplify the algorithmic settings, the HH will consider a low-level heuristic successful if it obtains \( \sigma \) improvements in a learning period \( \tau \), rather than just one improvement as in previous work [21, 22]. Our main result is the proof that the presented generalised random gradient hyper-heuristic with self-adjusting learning period has optimal performance on LeadingOnes. In contrast to the parameter \( \tau \) in the previous work, the hyper-parameter \( \sigma \) here is not very critical – all we require for our proof is that \( \sigma \in \Omega(\log^4 n) \cap o(\sqrt{n}/\log n) \).

We complement our theoretical results with some experiments which show that the self-adjusting HH converges to optimality faster than the HH with fixed values of \( \tau \) i.e., it is faster for realistic problem sizes.

The rest of the paper is structured as follows. In the Preliminaries we provide a brief overview of previous theoretical results concerning parameter adaptation and formally introduce the selection hyper-heuristics considered in this paper. In Section 3 we present our runtime analysis of the Adaptive Random Gradient (ARG) hyper-heuristic for LeadingOnes. Section 4 presents the experimental results for realistic problem sizes. We conclude the paper with a discussion of how ARG would also be efficient with larger sets of low-level heuristics.

2 PRELIMINARIES

In the next subsection we provide an overview of the theoretical results concerning self-adjusting parameters in evolutionary computation. In Subsection 2.2 we introduce the hyper-heuristics that have been previously analysed theoretically in the literature and introduce the Adaptive Random Gradient hyper-heuristic considered in this paper.

### Algorithm 1 Simple Selection Hyper-Heuristic [1, 2]

1. Choose \( x \in S \) uniformly at random
2. while stopping conditions not satisfied do
   3. Choose \( h \in H \) according to the learning mechanism
   4. \( x' \leftarrow h(x) \)
   5. if \( f(x') > f(x) \) then
      6. \( x \leftarrow x' \)

### Algorithm 2 Generalised Random Gradient Hyper-Heuristic [21]

1. Choose \( x \in S \) uniformly at random
2. while stopping conditions not satisfied do
   3. Choose \( h \in H \) uniformly at random
   4. \( c_\tau \leftarrow 0 \)
   5. while \( c_\tau < \tau \) do
      6. \( c_\tau \leftarrow c_\tau + 1; x' \leftarrow h(x) \)
      7. if \( f(x') > f(x) \) then
         8. \( c_\tau \leftarrow 0; x \leftarrow x' \)

2.1 Parameter Adaptation

Since adapting the step size is essential in continuous optimisation, parameter adaptation is well-established within that field. Within discrete settings it has been gaining attention in recent years. When the \((1 + (\lambda, \lambda)) \) GA was proposed, it was clear that the best value for the offspring population size \( \lambda \) changed during the optimisation process [12]. It was subsequently proved that adapting \( \lambda \) using a 1/5 rule allows the algorithm to be asymptotically faster than any static parameter choice for OneMax and optimise the function in linear time [11]. The only earlier theoretical work on adaptive parameters was [19] where self-adjusting mutation rates and population sizes where shown to improve the parallel runtime of an island model for several test functions. Concerning the \( r \)-valued OneMax function, a self-adjustment of the \((1+1) \) EA mutation rate inspired by the 1/5 rule was proven to find the asymptotically best possible runtime [13].

A self-adaptation scheme, i.e., where the mutation rate is encoded in the genome, has also been recently considered [8]. Dang and Lehre presented an example function for which in some areas of the search space a high mutation rate is required, while in other areas it is detrimental. They proved that a generational stochastic selection EA with self-adaptation is efficient while using static parameters leads to an exponential expected runtime.

In [14] a learning mechanism was proposed to estimate the efficiency of different parameter values from their medium-term past performance. This mechanism obtains essentially the best possible runtime achievable for OneMax by mutating the optimal number \( k \) of bits in each step. If an offspring population is used, then the mutation rate may be adjusted based solely on information learned in the current iteration and the learning mechanism may be avoided [15].

2.2 Hyper-Heuristics

Let \( S \) be a finite search space, \( H \) a set of low-level heuristics and \( f : S \rightarrow \mathbb{R} \) a cost function. Algorithm 1 shows the pseudocode for
which applies all available low-level heuristics in $H$ achieves at least optimal heuristics fail. We define a heuristic to be successful through the learning period by introducing a simple self-adjusting mechanism inspired by the $\tau$ theorem. This is the best possible runtime achievable using $\tau$ and $\sigma$ proved that if the learning period is set such that $\tau$ is increased to a multiplicative factor $F^{1/\sigma}$ and a new operator is chosen at random. Otherwise, once the heuristic is successful, $\tau$ it is decreased by a smaller factor $F^{1/\sigma^2}$ and a new learning period is started with the successful heuristic. This self-adjusting rule strives to adapt the learning period such that a failure occurs approximately every $\sigma$ successes, and to maintain a success probability of $1 - 1/\tau$, i.e., a $1 - o(1)$ rule. In contrast to the one success after several failures of traditional 1/5 rule algorithms [10, 17], the innovation behind the $1 - o(1)$ rule is that it seeks many successes before a failure. The parameter $F$ should be a constant greater than 1 (previously 1.5 has been used in discrete settings [12]). The Adaptive Random Gradient Hyper-Heuristic (ARG) is formally described in Algorithm 3.

### Algorithm 3: Adaptive Random Gradient Hyper-Heuristic

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\tau \leftarrow \tau_0$</td>
</tr>
<tr>
<td>2</td>
<td>Choose $x \in S$ uniformly at random</td>
</tr>
<tr>
<td>3</td>
<td>while optimum not found do</td>
</tr>
<tr>
<td>4</td>
<td>Choose $h \in H$ uniformly at random</td>
</tr>
<tr>
<td>5</td>
<td>$c_x \leftarrow 0; c_t \leftarrow 0$</td>
</tr>
<tr>
<td>6</td>
<td>while $c_t &lt; \tau$ do</td>
</tr>
<tr>
<td>7</td>
<td>$c_t \leftarrow c_t + 1; x' \leftarrow h(x)$</td>
</tr>
<tr>
<td>8</td>
<td>if $f(x') &gt; f(x)$ then</td>
</tr>
<tr>
<td>9</td>
<td>$c_x \leftarrow c_x + 1; x \leftarrow x'$</td>
</tr>
<tr>
<td>10</td>
<td>if $c_t \geq \sigma$ then</td>
</tr>
<tr>
<td>11</td>
<td>$c_t \leftarrow 0; c_t \leftarrow 0$</td>
</tr>
<tr>
<td>12</td>
<td>$\tau \leftarrow \tau \cdot F^{-1/\sigma^2}$</td>
</tr>
<tr>
<td>13</td>
<td>$\tau \leftarrow \tau \cdot F^{1/\sigma}$</td>
</tr>
</tbody>
</table>

a simple selection hyper-heuristic as used in previous theoretical analyses [1, 2].

In such works the following learning mechanisms, commonly used in the literature to solve combinatorial optimisation problems [6, 7] were considered: 

1. **Random Selection** (RS): each low-level heuristic $h \in H$ is randomly selected in each iteration.
2. **Greedy** (G): the heuristic that produces the best improvement is chosen for the current iteration.
3. **Random Gradient** (GR): the best heuristic out of $\min(1, \lfloor H/\lceil n \rceil \rfloor)$ is chosen uniformly at random.
4. **Generalised Greedy Gradient** (GGG) [17]: out of $\lceil n/2 \rceil$ operators, one is randomly selected.

Lissovoi et al. [21] proved that, when selecting between RLS$_1$ and RLS$_2$, the four mechanisms all have the same performance if $\tau$ is increased to $\tau \cdot F^{-1/\sigma^2}$ and the same operator is applied again. If fewer than $\sigma$ improvements are produced in $\tau$ iterations, $\tau$ is increased to $\tau \cdot F^{1/\sigma}$, and a mutation operator is used to change the current search point (i.e., the 2-bit ancestor). We believe that our results would also apply when RLS$_2$ is defined to flip two distinct bits, although we do not prove this.

ARG picks an initial mutation operator uniformly at random, and applies it for a number of iterations. If the mutation operator produces at least $\sigma$ improvements within $\tau$ iterations, $\tau$ is decreased to $\tau \cdot F^{1/\sigma^2}$, and the same operator is applied again. If fewer than $\sigma$ improvements are produced in $\tau$ iterations, $\tau$ is increased to $\tau \cdot F^{1/\sigma}$, and a mutation operator is used to change the current search point (i.e., the 2-bit ancestor). We believe that our results would also apply when RLS$_2$ is defined to flip two distinct bits, although we do not prove this.

ARG picks an initial mutation operator uniformly at random, and applies it for a number of iterations. If the mutation operator produces at least $\sigma$ improvements within $\tau$ iterations, $\tau$ is decreased to $\tau \cdot F^{1/\sigma^2}$, and the same operator is applied again. If fewer than $\sigma$ improvements are produced in $\tau$ iterations, $\tau$ is increased to $\tau \cdot F^{1/\sigma}$, and a mutation operator is used to change the current search point (i.e., the 2-bit ancestor). We believe that our results would also apply when RLS$_2$ is defined to flip two distinct bits, although we do not prove this.

### 3 Runtime Analysis

In this section, we will prove that ARG using $H = \{RLS_1, RLS_2\}$ has optimal runtime for the LEADINGONES (LO) benchmark function defined on the set $\{0, 1\}^n$ of bit-strings of length $n$:

$$\text{LEADINGONES}(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j.$$  

We consider the runtime of the Adaptive Random Gradient (ARG) algorithm, with access to 1-bit and 2-bit mutation operators, RLS$_1$ and RLS$_2$, which respectively flip one and two bits of the bit-string, chosen uniformly at random with replacement, as in [1, 21]. We note that as the bits are picked with replacement, RLS$_2$ can, with probability $1/n$, flip the same bit twice, constructing a copy of the ancestor bit-string. We believe that our results would also apply when RLS$_2$ is defined to flip two distinct bits, although we do not prove this.

ARG picks an initial mutation operator uniformly at random, and applies it for a number of iterations. If the mutation operator produces at least $\sigma$ improvements within $\tau$ iterations, $\tau$ is decreased to $\tau \cdot F^{1/\sigma^2}$, and the same operator is applied again. If fewer than $\sigma$ improvements are produced in $\tau$ iterations, $\tau$ is increased to $\tau \cdot F^{1/\sigma}$, and a mutation operator is used to change the current search point (i.e., the 2-bit ancestor). We believe that our results would also apply when RLS$_2$ is defined to flip two distinct bits, although we do not prove this.

ARG picks an initial mutation operator uniformly at random, and applies it for a number of iterations. If the mutation operator produces at least $\sigma$ improvements within $\tau$ iterations, $\tau$ is decreased to $\tau \cdot F^{1/\sigma^2}$, and the same operator is applied again. If fewer than $\sigma$ improvements are produced in $\tau$ iterations, $\tau$ is increased to $\tau \cdot F^{1/\sigma}$, and a mutation operator is used to change the current search point (i.e., the 2-bit ancestor). We believe that our results would also apply when RLS$_2$ is defined to flip two distinct bits, although we do not prove this.

We refer to the operator with the greatest probability of producing an improvement at the current search point (i.e., the 2-bit mutation operator for $i < n/2$, and the 1-bit mutation operator for $i \geq n/2$) as the optimal operator, and use $p_{\text{opt}}(i)$ to denote its improvement probability. Conversely, we call the operator with the smaller improvement probability non-optimal, and use $p_{\text{non-opt}}(i)$ to denote its probability of producing an improvement in one iteration.

**Theorem 3.1.** The expected runtime of the Adaptive Random Gradient algorithm on LEADINGONES, for $\tau_0 = 1$, $\sigma = \Omega(\log^4 n)$, $\tau = \Theta(\log^2 n)$, and $F > 1$ a constant, is

$$E[\tau_{\text{ARG}}] \leq ((1 + \log(2))/4) n^2 + o(n^2).$$

**Proof.** We call a period of at most $\tau$ iterations in which the mutation operator produces $\sigma$ LEADINGONES improvements by mutation a successful phase, and a period of $\tau$ iterations in which the
mutation operator produces less than $\sigma$ LeadingOnes improvements a failed phase. We will bound $T_{\text{ARG}} = T_{\text{mid}} + T_S + T_{NS} + T_F$ by bounding each of the four contributing components:

- $T_{\text{mid}}$, the number of iterations spent in the middle region, where $|LO(x) - (n/2 - 1)| < \beta n$, and $LO(x)$ is the LeadingOnes value of the current solution. By Lemma 3.4, $E[T_{\text{mid}}] = o(n^2)$ for any $\beta \in o(1)$.
- $T_S$, the number of iterations spent in successful phases applying the optimal operator outside the middle region,
- $T_{NS}$, the number of iterations spent in successful phases applying the non-optimal operator outside the middle region,
- $T_F$, the number of iterations spent in failed phases outside the middle region.

To prove the theorem, we will bound $E[T_S] \leq E[T_{\text{OPT}}]$, and show that the expected values of the other contributing terms are at most $o(n^2)$.

We define $T_{\text{opt}}(i) := (1 + 4/\ln n) \cdot \sigma/p_{\text{pop}}(i)$, where $p_{\text{pop}}(i)$ is the improvement probability of the optimal operator, and $i$ is the LeadingOnes value of the current solution, and will make use of the following.

(1) With high probability, $\tau$ remains below $T_{\text{opt}}(i)$ throughout the optimisation process per Lemma 3.5.

(2) While $|LO(x) - (n/2 - 1)| > \beta n$ and $\tau < T_{\text{opt}}(i)$, the non-optimal operator fails a phase with probability $(1 - e^{-\Omega(\beta n)})$ per Lemma 3.6.

For $T_S$, we note that $E[T_S] \leq E[T_{\text{OPT}}]$, as the optimum would be found in expectation after $E[T_{\text{OPT}}]$ iterations of applying the optimal mutation operator, while ARG can additionally make progress toward the optimum using the non-optimal operator, as well as in periods which fail to produce improvements.

For $T_{NS}$ and $T_F$, bounds on the number of successful phases with the non-optimal operator as well as the total number of failed phases are needed. Let $L_S$ denote the event that $\tau$ remains below $T_{\text{opt}}(i)$ throughout the optimisation process, $N_S$ and $N_F$ denote the number of successful and failed phases (respectively) that occur before the global optimum is constructed, and $\tau_{\text{end}}$ be the value of $\tau$ when the global optimum is obtained. $N_F$ can be bounded by observing that, regardless of the order of the phases, the following balance relation is valid: $\tau_{\text{end}} = \tau_{\text{opt}}(N_F/\sigma - N_S/\sigma^2)$.

Conditional on $L_S$, we have $\tau_{\text{end}} < T_{\text{opt}}(n)\sigma n$, and hence

$$\log e T_{\text{opt}}(n) \geq N_F/\sigma - N_S/\sigma^2.$$  

As each successful phase provides at least $\sigma$ LeadingOnes improvements, we can bound $N_S \leq n/\sigma$, and then bound $N_F$:

$N_F \leq N_S/\sigma + \sigma \log e T_{\text{opt}}(n) = O(n/\sigma^2 + \sigma \log n).$

This bounds the number of iterations spent in failed phases:

$$E[T_F | L_S] \leq T_{\text{opt}}(n) \cdot N_F \leq (1 + o(1))\sigma n \cdot O(n/\sigma^2 + \sigma \log n) = o(n^2).$$

Except during the iterations spent in the middle region of the search space (which are counted by $T_{\text{mid}}$), the probability that a non-optimal operator produces $\sigma$ improvements within a single phase is at most $o(1)$ by Lemma 3.6. Conditional on $L_S$, there are at most $N_S + N_F = O(n/\sigma)$ phases in total, and hence at most $o(1) \cdot O(n/\sigma) = o(n/\sigma)$ successful phases with the non-optimal operator, which, combined with $\tau < T_{\text{opt}}(i) \leq T_{\text{opt}}(n)$ yields a bound on the number of iterations spent in successful phases using the non-optimal operator:

$$E[T_{NS} | L_S] \leq T_{\text{opt}}(n) \cdot o(n/\sigma) = o(n^2).$$

Combining the four contributing factors, by the linearity of expectation, the expected runtime of ARG is:

$$E[T_{\text{ARG}} | L_S] \leq E[T_S | L_S] + E[T_F | L_S] + E[T_{NS} | L_S] + E[T_{mid} | L_S],$$

noting that the bounds on the expected values of $T_S$ and $T_{mid}$ derived previously hold also when conditioned on $L_S$.

For an unconditional expectation, we use Lemma 3.3 to bound the expected runtime of ARG when $L_S$ fails to hold as $O(\sigma n^2)$. By Lemma 3.5, the probability of $\tau$ exceeding $T_{\text{opt}}(i)$ at any point before the global optimum is found can be made $n^{-c}$ small for any constant $c > 0$, and hence, for e.g. $c' = 2$:

$$E[T_{\text{ARG}}] = E[T_S] + E[T_F | L_S] + E[T_{NS} | L_S] + E[T_{mid} | L_S].$$

$$E[T_{\text{OPT}}] + O(n^2) + O(\sigma n^2) = E[T_{\text{OPT}}] + o(n^2),$$

as $\sigma = o(\sqrt{n/\log n}).$ \qed

Note that the proof above not only shows that ARG has (apart from lower-order terms) the optimal runtime achievable with the given set of operators, but also that, with probability $1 - o(1)$, only a fraction of $o(1)$ of the iterations use the non-optimal operator. Hence our self-adjusting mechanism in an extremely good manner manages to select the most suitable mutation operator.

Without proof, we note that analogous results also hold for larger sets of operators, e.g., for all $k$-bit flip mutation operators for $k$ from some constant-size set of alternatives (including 1 to ensure convergence). The reason is that again, after excluding a constant number of lower-order ranges of the objective space (as above the middle range), the best operator is sufficiently better than the others and the adjusting mechanism is sufficiently sensitive so that the $\tau$-value stays in a range in which essentially only the optimal operator can lead to successful phases. Also without proof, we note that also non-constant numbers of operators could be handled by our algorithm, again giving an optimal performance apart from lower order terms.

In the rest of this section we will prove the lemmas required to complete the proof of Theorem 3.1. Before this, we present the following technical lemma, which allows us to bound the probability that a process on the non-negative integers which in expectation decreases in value remains above its initial value after a certain amount of time. A similar result has been proven in [18]. As witnessed by the ln $(1/(\sigma + 1/\sigma))$ term, our bound is stronger when the expected movement away from zero is significantly smaller than the expected movement towards zero. We need this stronger result in Lemma 3.3.

**Lemma 3.2.** Let $X_0, X_1, \ldots$ be a random process on the non-negative integers. Assume that there are $a \in \mathbb{N}_{\geq 1}$ and $\rho \in (0, 1/\rho(a))$ such that for all $t$ and all $k \geq 1$, we have $Pr[X_{t+1} = X_t + 1 | X_t = k] \leq \rho$ and $Pr[X_{t+1} = X_t - 1 | X_t = k] = 1 - Pr[X_{t+1} = X_t + 1 | X_t = k]$, where
Assume further that \( X_0 = 0 \). Then for all \( k \in \mathbb{N} \) and all \( t \in \mathbb{N} \), we have
\[
\Pr[X_t \geq k] \leq \exp\left(-\frac{(k-1)(1-(\sigma+1)p)}{\sigma a + 1}\left(\ln\frac{1}{\sigma + 1} - 1\right)\right) K(a + 1),
\]
where \( K \) is an absolute constant.

\[ \Box \]

**Lemma 3.3.** Consider a run of the ARG, started with an arbitrary initial search point \( x \) and an arbitrary initial period length \( t_0 \leq n^3 \), on the LEADINGONES problem. Let \( T \) be the runtime, that is, the number of fitness evaluations performed up to the point when for the first time the optimal solution is evaluated. Then \( \E[T] = O(\sigma n^3) \).

**Proof.** While the fitness is less than \( n-1 \), each of the two operators has a probability of at least \( 1/n^2 \) of finding an improvement. Hence by a simple fitness level argument, the time \( t_0 \) to reach a fitness of at least \( n-1 \) satisfies \( \E[T_0] \leq n^3 \).

Throughout this first part of the optimisation process, we have that a period starting with a \( \tau \)-value of \( n^3 \) or more is successful with probability \( 1 - \exp(-\Theta(n/\sigma)) = 1 - \rho \). This is because the expected number of improvements is at least \( \tau/\sigma^2 \), whereas for a success we need only \( \sigma \) improvements. Hence a Chernoff bound for geometrically distributed random variables shows this claim.

Let \( i \) be minimal such that \( \tau^i := t_0^i \tau^{i/\sigma^2} \geq n^3 \). In other words, \( \tau^i \) is the smallest \( \tau \) value not smaller than \( n^3 \) which we could encounter in this run of the algorithm. For any time \( t \), let \( X_t = \max(0, \log_{\tau^{i/\sigma^2}}(t_0/\tau^i)) \). In other words, if \( t_0 \geq \tau^i \), then \( X_t = 0 \). Otherwise, \( X_t \) is such that \( \tau^i = (\tau^{i/\sigma^2})^{X_t} \); otherwise \( X_t = 0 \). By definition, we have \( X_0 = 0 \). Also, for all \( t \geq 0 \) and all \( k \geq 1 \), we have \( \Pr[X_{t+1} = X_t + \sigma | X_t = k] \leq \rho \) and \( \Pr[X_{t+1} = X_t - 1 | X_t = k] = 1 - \Pr[X_{t+1} = X_t + \sigma | X_t = k] \).

By Lemma 3.2, we have
\[
\Pr[X_t \geq k] \leq \exp\left(-\frac{(k-1)(1-(\sigma+1)p)}{\sigma a + 1}\left(\ln\frac{1}{\sigma + 1} - 1\right)\right) K(a + 1),
\]
where \( K \) can be chosen as an absolute constant (independent from \( a \) and \( c \), provided that \( c < 1/e \)).
Let $r_i$ be the probability that the optimum is found in $P_i$ and let $\hat{R}_i := \sum_{j=1}^{i-1} r_j$ be the probability that it is found in period $P_i$ or later. Let $\theta_0 \geq 1$ be minimal such that $r_i^\theta \geq n$. Note that for each $i \geq \theta_0$, we have $q_i \geq (1/2)(1 - 1/e)$. Consequently, $\hat{R}_i \leq \min\{1, (1 - (1/2)(1 - 1/e))^i\}$. This gives $E[T_i | T_{\theta_0}] \leq \sum_{i=1}^{\infty} E[i \cdot \hat{R}_i + \sum_{j=1}^{i} r_j^\theta] = T_{\theta_0} + \sum_{i=1}^{\infty} E[i \cdot \hat{R}_i^\theta]$. We estimate $\sum_{j=1}^{i} \hat{R}_i^\theta \leq \sum_{j=1}^{i} \frac{e^{(1-\theta_0)^j}}{\theta_0^j} \frac{\theta_0}{\theta_0} \leq \frac{e^{(1-\theta_0)^j}}{\theta_0^j} \frac{\theta_0}{\theta_0} \leq T_{\theta_0}(1 + 1/e)^{\theta_0} \leq T_{\theta_0}$. Hence, we have $E[T_i] = O(1)$. Let $\alpha \in (2, 3)$ be an arbitrary number and $\beta \in (0, 2/3)$. Then, the global optimum is found in $n^{\alpha - 1} \beta$ with probability $1 - n^{-\alpha \beta}$. To show this, the obvious guess strategy would be to return $T_{\theta_0}$ if no improvement is found, but we can do better. If $T_{\theta_0}^\theta \leq 1 - n^{-\alpha \beta}$, we know that $E[T_i] \leq O(1)$, and if $T_{\theta_0}^\theta > 1 - n^{-\alpha \beta}$, we know that $E[T_i] \leq T_{\theta_0}(1 + 1/e)^{\theta_0} \leq T_{\theta_0}$. Thus, we have $E[T_i] \leq O(1)$. The result follows.

**LEMMA 3.4.** The expected runtime of any combination of RLS and RLS2 for the $|LO(x)| = n/2 - 1 < \beta n$ region of the search space, where $\beta \in (0, 1)$, is at most $O(n^2)$.

**Proof.** This region contains $2\beta n = o(n)$ fitness values. The expected waiting time for an improvement within this region is smaller than $1/p_{\text{opt}}(3n/4 - 1) = 4n$. Hence, a simple fitness level argument gives that the region is optimised after $2\beta n \cdot 4n = o(n^2)$ iterations in expectation.

**LEMMA 3.5.** With high probability (at least $1 - n^{-c'}$ for any $c' > 0$), ARG finds the global optimum before $\tau \geq \tau_{\text{max}}(i)$ occurs, where $i$ is the LEADINGONES value of the ancestor individual in any given iteration.

**Proof.** We prove the claim by showing that at any time of the algorithm ($L_0(x) = i$) where we have $\tau \geq \tau_{1}(i) = (1 + 1/\ln(n)) \cdot \sigma/p_{\text{opt}}(i)$, and then failures increase $\tau$ over $\tau_{1}(i)$, the probability that the algorithm increases $\tau$ to over $\tau_{2}(i) = \tau_{1}(i) \cdot p_{\text{log}(\ln(n))}^{\tau_{1}(i)}/\sigma < \tau_{\text{max}}(i)$ or more without first reaching or going below $\tau_{1}(i)$ is at most $n^{-c'}$ for any $c' > 0$. We then prove that, with probability at least $1 - n^{-c'}$, this does not happen at any point throughout the optimisation process. We first show that $\tau_{2}(i) < \tau_{\text{max}}(i)$. Note that, by a Maclaurin series argument, $E[\log(n)]^2/\sigma = 1 + \frac{\ln(\log(n))^2}{\sigma} + o\left(\frac{\ln(\log(n))^2}{\sigma}\right) = 1 + O(\log(n)^2)$, and hence:

$$\tau_{2}(i) = \tau_{1}(i) \cdot \frac{p_{\text{log}(\ln(n))^2}}{\sigma} = (1 + O(\log(n)^2) \cdot (1 + \ln(n)) \cdot \sigma/p_{\text{opt}}(i) \leq (1 + \ln(n)) \cdot \sigma/p_{\text{opt}}(i) \leq (1 + 4/\ln(n)) \cdot \sigma/p_{\text{opt}}(i) = \tau_{\text{max}}(i).$$

To increase $\tau$ to $\tau > \tau_{2}(i)$ from $\tau \leq \tau_{1}(i)$, at least $\log(n)^2$ failures are necessary; hence, at least $\log(n)^2$ random operator choices are necessary. With probability at least $1 - n^{-\log(n)}$, at least one of these choices is with the optimal operator.

With overwhelming probability, the LEADINGONES value of the current solution is increased by no more than $3\sigma$ in a successful phase. Let $Y$ be a binomially-distributed random variable with parameters $r_{\text{max}}(i)$ and $p_{\text{opt}}(i + 3\sigma)$; the number of improvements produced by the optimal operator in a period of $\tau \geq \tau_{1}(i)$ iterations stochastically dominates $Y$. Applying a classic multiplicative Chernoff bound (see, e.g., Corollary 1.10 (c) in [9]) with $E[Y] \geq \mu := (1 + 1/\ln(n)) \cdot (1 - n^{-\beta n/2})/\sigma = \Theta(\sigma)$ and $\delta := n^{-\beta n/2} \log(n) = \Theta(1/\log(n))$, such that $\tau \leq \mu = \sigma$ yields

$$P[Y < \sigma] \leq P[Y \leq \sigma] \leq \exp\left(-\sigma \cdot \frac{n - 6\sigma \log(n) - 6\sigma}{(1 + \ln(n))(n - 6\sigma)} \cdot \frac{\sigma}{2n}\right) \leq \exp\left(-\Omega\left(\frac{\sigma}{\log(n)}\right)\right) = n^{-\Omega(1/\log(n))}.$$
\[\mu := E[Y] = (1 + 4/\ln(n)) \cdot \frac{\sigma}{\ln(n)^4}, \text{ and } \delta := \frac{2\ln(n)\beta^4}{\ln(n)^4} \text{ (such that } (1 + \delta)\mu = \sigma \text{) yields}\]

\[
\Pr[Y \geq \sigma] \leq \exp\left(-\frac{2\ln(n)(\beta^4 - 4)}{\ln(n) + 4} \cdot \left(1 + \frac{4}{\ln(n)}\right) - \frac{\sigma}{2\beta + 1} \right) \\
= \exp(-\Theta(\sigma\beta)) = o(1).
\]

A similar argument holds at \(i = n/2 - 1 + \beta n\) where a binomially-distributed random variable \(Y'\) with parameters \(\tau_{\text{max}}(n/2 - 1 + \beta n) = (1 + 4/\ln(n)) \cdot \sigma n \) and \(p_{\text{opt}}(n/2 - 1 + \beta n) = (1 - 2\beta)/n\) stochastically dominates the number of improvements produces by the non-optimal operator in \(\tau < \tau_{\text{max}}(n/2 - 1 + \beta n)\) iterations. A Chernoff bound with \(\mu := E[Y] = (1 + 4/\ln(n)) \cdot (1 - 2\beta) \cdot \sigma \) and \(\delta := \frac{2\beta(\ln(n) + 4\beta^4 - 2)}{(1 - 2\beta)/\ln(n) - (8\beta^4 + 4)}\) (such that \((1 + \delta)\mu = \sigma\) clearly yields the same result: \(\Pr[Y' \geq \sigma] = o(1)\).

\[\Box\]

### 4 EXPERIMENTAL SUPPLEMENTS

In the previous section we have proven that for large enough \(n\) the ARG hyper-heuristic has the optimal possible performance for \texttt{LeadingOnes}, up to lower order terms. We perform a set of experiments to assess the performance of ARG for realistic problem sizes. Our theoretical results rely on the adaptation parameter \(\sigma\) to grow slowly with the problem size, i.e., asymptotically between \(\Omega(\log^2 n)\) and \(o(\sqrt{n}/\log n)\). We experiment with various super-constant values within and outside the theoretical bounds. We introduce multiplicative constants \(c^*\) such that \(\sigma\) is sufficiently small compared to \(n\), for small problem sizes. We decide to arbitrarily set \(c^*\) such that for our smallest problem size \((n = 10^3)\), \(\sigma = 4\). We set the initial learning period trivially to \(\tau_0 = 1\). We then set \(F = 1.5\) as suggested by [10] and run all algorithms 100 times.

In Figure 1 we plot the average runtimes of the ARG hyper-heuristic as the problem size increases and compare their performance against the the GRG hyper-heuristic with fixed \(\tau\) values. We pick the best performing versions of GRG identified in [22] (i.e., \(\tau = 0.4n \ln n, 0.5n \ln n\) and the best, \(0.6n \ln n\), for \(n = 100,000\)). While the runtimes of all the ARG hyper-heuristics are comparable with the three GRG variants, some outperform them. These include all the ones with \(\sigma\) values within the theoretical bounds. Hence, for realistic problem sizes, better runtimes can be achieved by adapting \(\tau\) during the run.

Figure 2 shows the adaptation of \(\tau\) for five individual runs for \(n = 10^3\) of the ARG hyper-heuristic with \(\sigma = \sqrt{n}/\ln n\), as well as an average over 100 runs. As expected, the learning period \(\tau\) is quickly increased into the range where the optimal operator produces more than \(\sigma\) improvements per \(\tau\) iterations in expectation. The adapted learning period tracks the increasing waiting times for an improvement by the optimal operator in the first half the search space, while remaining stable in the second half where the waiting time does not change. \(\tau_{\text{max}}\) is included to show how the parameter \(\tau\) adapts as suggested within the proof of Theorem 3.1. The average runtime of ARG over 100 runs in this setting was 0.42634m².

Figure 3 shows the percentage of iterations where the optimal operator is used by ARG with \(\sigma = \sqrt{n}/\ln n\) for problem sizes \(n = 10^3\) and \(n = 10^4\), averaged over 100 independent runs. We divide the \(n + 1\) fitness values into 100 ranges and plot the percentage of iterations where the optimal operator is employed. We see that

![Figure 1: Average number of fitness function evaluations required by the ARG and GRG hyper-heuristics (with parameters \(\sigma\) and \(\tau\) respectively) to find the \texttt{LeadingOnes} optimum as the problem size \(n\) increases. RLS₁ and \(2_{opt}\) show the best runtime achievable by using respectively one low-level heuristic or both.](image1)

![Figure 2: Adapted value of \(\tau\) over time in five typical runs for \(\sigma = \sqrt{n}/\ln n, n = 10^3\), and an average over 100 runs.](image2)

ARG exhibits the behaviour predicted by Theorem 3.1, already for these problem sizes. Naturally, as \(n\) increases, the curves are smoother and the optimal operator is used more often. As expected, both operators are used in the middle section where they have similar improvement probabilities. Outside this area, the optimal operator is used most of the time as desired.

### 5 CONCLUSION

Recently it has been proven that the ARG selection hyper-heuristic runs in optimal time for \texttt{LeadingOnes} if the learning period \(\tau\) is set appropriately. In this paper we have presented an adaptive random gradient (ARG) hyper-heuristic that automatically adjusts the learning period throughout the run. ARG uses an innovative \(1 - o(1)\) self-adjusting rule that strives to adapt the learning period such that a failure occurs approximately every \(o(1)\) successes. The novelty consists of seeking many successes before a failure in contrast to a success after many failures of traditional adaptive algorithms [10, 17].
We believe that also non-constant numbers of operators could be handled by our algorithm, again giving optimal performance apart from lower order terms.

We have complemented the theory with experiments for realistic problem sizes. The results show that the parameter $\sigma$, indicating the ratio of successes to failures which maintains a stable $\tau$ value, is fairly robust. If it is set within the range of values predicted by our theoretical analysis, then ARG outperforms the best hyper-heuristics with fixed $\tau$ reported in the literature.

Future work should evaluate the performance of ARG for other problems including ones from combinatorial optimisation and real-world applications.

Acknowledgements. This work was initiated in May 2017 while Pietro S. Oliveto was a chercheur invité at École Polytechnique. This work was supported by a public grant as part of the Investissement d’avenir project, reference ANR-11-LABX-0056-LMH, LabEx LMH, in a joint call with Gaspard Monge Program for optimization, operations research and their interactions with data sciences, and by EPSRC under grant EP/M004252/1.

REFERENCES


