On the Runtime Analysis of Generalised Selection Hyper-heuristics for Pseudo-Boolean Optimisation

Andrei Lissovoi
Department of Computer Science
University of Sheffield
Sheffield, S1 4DP, United Kingdom

Pietro S. Oliveto
Department of Computer Science
University of Sheffield
Sheffield, S1 4DP, United Kingdom

John Alasdair Warwicker
Department of Computer Science
University of Sheffield
Sheffield, S1 4DP, United Kingdom

ABSTRACT

Selection hyper-heuristics are randomised search methodologies which choose and execute heuristics from a set of low-level heuristics. Recent time complexity analyses for the LeadingOnes benchmark function have shown that the standard simple random, permutation, random gradient, greedy and reinforcement learning selection mechanisms show no effects of learning. The idea behind the learning mechanisms is to continue to exploit the currently selected heuristic as long as it is successful. However, the probability that a promising heuristic is successful in the next step is relatively low when perturbing a reasonable solution to a combinatorial optimisation problem. In this paper we generalise the classical selection-perturbation mechanisms so success can be measured over some fixed period of length τ, rather than in a single iteration. We present a benchmark function where it is necessary to learn to exploit a particular low-level heuristic, rigorously proving that it makes the difference between an efficient and an inefficient algorithm. For LeadingOnes we prove that the *generalised random gradient* mechanism approaches optimal performance while *generalised greedy*, although not as fast, still outperforms *random local search*. An experimental analysis shows that combining the two generalised mechanisms leads to even better performance.

CCS CONCEPTS

•Theory of computation → Optimization with randomized search heuristics;

KEYWORDS

Theory, Running time analysis, Hyper-heuristics, Selection heuristics

ACM Reference format:
Andrei Lissovoi, Pietro S. Oliveto, and John Alasdair Warwicker. 2017. On the Runtime Analysis of Generalised Selection Hyper-heuristics for Pseudo-Boolean Optimisation. In *Proceedings of GECCO '17, Berlin, Germany, July 15-19, 2017,* 8 pages.
DOI: http://dx.doi.org/10.1145/3071178.3071288

1 INTRODUCTION

Many successful applications of randomised search heuristics to real-world optimisation problems have been reported. Despite these successes, it is still difficult to decide which particular search heuristic is a good choice for the problem at hand, and what parameter settings should be used. In particular, while it is well understood that each heuristic will be efficient on some classes of problems and inefficient on others [21], very little guidance is available explaining how to choose an algorithm for a given problem. The high level idea behind the field of hyper-heuristics is to overcome this difficulty by evolving the search heuristic for the problem rather than choosing one in advance (or applying several arbitrary ones until a satisfactory solution is found). The overall goal is to automate the design and the tuning of the algorithm for the optimisation problem, and hence achieve a more generally applicable system.

Hyper-heuristics are usually classified into two categories: *selection hyper-heuristics* and *generation hyper-heuristics* [5]. The former repeatedly select from a set of low-level heuristics to create new solutions for the problem, while the latter generate new heuristics from components of existing heuristics. The low-level heuristics can be further classified as either *construction heuristics*, which build a solution incrementally, or *perturbation heuristics*, which start with a complete solution and try to iteratively improve the current solution.

Despite the numerous successes for NP-hard optimisation problems, including scheduling [6, 7, 10], timetabling [19], vehicle routing [3], cutting and packing [15], the theoretical understanding of hyper-heuristics is very limited. Some insights into the behaviour of selection heuristics have been achieved via landscape analyses [16–18]. Concerning their performance, Lehre and Özcan presented the first runtime analysis of selection-perturbation hyper-heuristics. They considered a simple random hyper-heuristic [6] that at each step randomly chooses between low-level heuristics and presented an example benchmark function class, called GapPath, where it is necessary to use more than one low-level heuristic to optimise the problem [14]. Similar results have also been presented by [11].

A comparative time complexity analysis of selection hyper-heuristics has recently been presented by Alanazi and Lehre [1]. They considered several common selection mechanisms, namely simple random, permutation, random gradient and greedy [6, 7] and analysed their performance on the standard LeadingOnes benchmark function when using a low-level heuristic set consisting of a 1-bit flip and a 2-bit flip operator (i.e., the same set previously considered in [14]). The runtime analysis results show that the four selection methods have the same asymptotic runtime, while experimental trials indicate that the runtimes are practically equivalent. Recently, additive reinforcement learning selection was also shown to often have roughly equivalent performance to simple random selection, including for the same problem setting (i.e., LeadingOnes selecting between 1-bit and 2-bit mutation operators) [2].

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
GECCO '17, Berlin, Germany
© 2017 Copyright held by the owner/author(s). 978-1-4503-4920-8/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3071178.3071288

In particular, the results indicate that selection mechanisms such as reinforcement learning and random gradient do not learn to exploit the more successful low-level heuristics and end up having the same performance as simple random selection.

The main idea behind the learning mechanisms considered above is to continue to exploit the currently selected heuristic as long as it is successful. Unlike construction heuristics, where iterating a greedy move on a currently successful heuristic may work for several consecutive construction steps, the probability that a promising heuristic is successful in the next step is relatively low when perturbing a reasonable solution to a combinatorial optimisation problem. In this paper we generalise the classical selection-perturbation learning mechanisms analysed in [1] so success can be measured over some fixed period of length τ, rather than in a single iteration. Since simple random and permutation do not use any feedback from the optimisation process (i.e., do not attempt to learn), we generalise the greedy and the random gradient learning mechanisms.

The rest of the paper is structured as follows. In the next section, we formally introduce the simple hyper-heuristic framework together with the classical selection mechanisms and the generalised ones. In Section 3, we prove that the generalised mechanisms are efficient on the GapPath function which was introduced in [14] to show that selection hyper-heuristics are necessary. We then introduce a generalised version of this function where one operator is useful more often than the other, and add trap points, which can only be avoided if the hyper-heuristic has learned to prefer 2-bit mutations over 1-bit mutations. This provides an example function class where the generalised mechanisms are efficient with overwhelming probability (w.o.p), while the classical mechanisms require infinite optimisation time (w.o.p).

In Section 4 we analyse the runtime of the generalised mechanisms on the LeadingOnes function, for which the classical mechanisms were analysed in [1]. The aim is to show that hyper-heuristics may outperform the single heuristics also on problems where the individual heuristics are efficient. We first provide a more precise analysis for the classical mechanisms, proving rigorously that they have equivalent expected runtimes up to lower order terms. In particular, they are less efficient than using the 1-bit mutation operator alone (i.e., Randomised Local Search (RLS)). Afterwards, we calculate the best possible runtime that may be achieved by a mixed strategy algorithm using the two operators and prove that the generalised random gradient algorithm achieves similar leading constants in its expected runtime. Concerning the generalised greedy algorithm, our theoretical results prove that it outperforms RLS, but is not as fast as generalised random gradient. In Section 5 we present experiments that confirm our theoretical results for realistic problem sizes and show that a selection mechanism that combines characteristics of both the greedy and the random gradient mechanisms has even better performance.

Due to space constraints, we omit some straightforward proofs, as well as some proofs for the Generalised Greedy mechanism, which use similar ideas to those used for Generalised Random Gradient.

Algorithm 1 Simple Selection Hyper-heuristic

```
Choose s ∈ S uniformly at random
while stopping conditions not satisfied do
    Choose h ∈ H according to the learning mechanism
    s' ← h(s)
    if f(s') > f(s) then
        s ← s'
```

2 PRELIMINARIES

Let S be a finite search space, H a set of low-level heuristics and $f : S \to \mathbb{R}$ a cost function. Algorithm 1 shows the pseudocode representation for a simple selection hyper-heuristic as used in previous theoretical analyses [1].

The following learning mechanisms have been commonly used in the literature to solve combinatorial optimisation problems [6, 7]: **Simple Random**, which selects a low-level heuristic independently with probability p_h in each iteration (usually $p_h = 1/|H|$, i.e., uniformly at random); **Permutation**, which generates a random ordering of low-level heuristics and returns them in that sequence when called by the hyper-heuristic; **Greedy**, which applies all available low-level heuristics in parallel and returns the best found solution; **Random Gradient**, which randomly selects a low-level heuristic, and keeps using it as long as it obtains improvements.

Alanazi and Lehre [1] analysed the runtime of these four classical mechanisms[1] for the LeadingOnes benchmark function, choosing between flipping either one, or two, randomly chosen bits of the bit string (1BitFlip and 2BitFlip respectively). We derive the exact runtimes in Section 4. It was shown experimentally in [1] that all mechanisms have the same performance as just choosing low-level heuristics at random. By making a heuristic selection decision in every iteration without taking past performance into account, the classical mechanisms do not have enough time to learn which operator is preferable at the current optimisation stage.

In this paper we generalise the classical mechanisms to allow a longer time period to decide whether a low-level heuristic is currently successful or not, aiming to maintain the intrinsic ideas of the classical learning mechanisms while generalising sufficiently to allow learning to take place. In particular, we consider:

Generalised Greedy (GG): all low-level heuristics are tested on the current candidate solution until an improvement is found (*Decision Stage*). The improvement (chosen uniformly at random if there are multiple) is then accepted, and the corresponding operator is run for a fixed period of length τ (*Exploitation Stage*).

Generalised Random Gradient (GRG): a low-level heuristic is chosen uniformly at random and run for a period of fixed length τ. When an improvement is found, a new period of length τ is immediately initialised. If the chosen operator fails to provide an improvement in τ iterations, a new operator is chosen at random.

3 LEARNING IS NECESSARY

In this section we consider the GapPath function (GP) previously introduced by Lehre and Özcan as an example to show that applying multiple low-level heuristics may be necessary [14]. We prove that

[1] Throughout the paper, bounds on the runtimes of the considered mechanisms refer to the number of fitness evaluations performed by Algorithm 1 using these mechanisms.

the generalised mechanisms are also efficient on the problem. We then generalise the GP function to make one operator useful more often than the other, and add trap points, which make the classical mechanisms fail (w.o.p), while the generalised mechanisms can find the global optimum efficiently (w.o.p). The GP function is defined as such [14]:

$$\mathrm{GP}(x) := \begin{cases} \mathrm{ZM}(x) & \text{if } \textsc{Ridge}(x) \bmod 3 = 1 \\ \mathrm{ZM}(x) + 2n\textsc{Ridge}(x) & \text{otherwise} \end{cases}$$

where $\mathrm{ZM}(x) := \sum_{i=1}^{n}(1-x_i)$, and $\textsc{Ridge}(x) := i$ if $x = 1^i 0^{n-i}$, and 0 otherwise.

The function consists of a short path which corresponds to all search points of the form $1^i 0^{n-i}$. The path contains gaps where the function values are inferior to the rest of the path. Such gaps consist of all path points where $i \equiv 1 \pmod 3$. As a result of these gaps, an algorithm that only uses either the 1BitFlip or the 2BitFlip operator will have infinite expected runtime, since it is necessary to alternate the operators in order to make progress on the path. The authors of [14] prove that the runtime of the simple random hyper-heuristic, initialised on the 0^n bit string and using only the 1BitFlip and 2BitFlip operators with probability p and $1-p$ respectively, is $\left(\frac{n^3-3n^2}{6(1-p)} + \frac{n^2}{3(1-p)p}\right)$ for any probability $p \in (0,1)$. In the following two theorems we show that the generalised mechanisms are also efficient on GapPath when choosing between the same two operators. Furthermore, our bounds suggest that GG is faster than GRG, which is confirmed experimentally for small values of n.

Theorem 3.1. *The expected runtime of the Generalised Greedy algorithm, when initialised on the 0^n bit string, for GapPath is $n^3/3 + 2n^2/3 + 2\tau n/3 - \tau$.*

Proof. The GP function is optimised by requiring multiple repetitions of 1-bit and 2-bit consecutive improvements. There is only one possible successful move at each point on the path and only the use of the correct operator will lead to an improvement.

If a 1-bit improvement is necessary, the success probability of the 1BitFlip operator is $1/n$, implying n expected 1-bit mutations before an improvement is found (by standard waiting time arguments). During this time, n 2-bit mutations will be performed in parallel, as Generalised Greedy mechanism applies both operators during the Decision Stage.

When a 2-bit improvement is necessary, the success probability of the 2BitFlip operator is $1/n \cdot 1/n \cdot 2$ (since two specific bits must be flipped, but can be chosen in either order) implying $n^2/2$ expected 2-bit mutations before an improvement is found. The same number of 1-bit mutations will be evaluated in parallel.

After an operator has succeeded, the Generalised Greedy mechanism will deterministically apply the successful operator (on its own) for another τ iterations (the Exploitation Stage), during which no improvement can occur (due to the nature of the GP function), before starting the next Decision Stage.

Initially, a 2-bit improvement is required. The expected number of fitness function evaluations needed to reach the first point on the path 110^{n-2} from the 0^n bit string is $2 \cdot n^2/2 = n^2$, which will be followed by τ non-improving steps. The next improvement will then occur by the 1BitFlip operator after an expected $2n$ steps, followed by τ unsuccessful 1-bit flips. Hence, the expected time to optimise the first three bits is $n^2 + \tau + 2n + \tau = n^2 + 2n + 2\tau$.

This three bit pattern occurs a total of $n/3$ times before the optimum is reached, and as the global optimum is reached just before the final exploitation stage starts, the total expected runtime is $E(T_{GG}) = n/3 \cdot (n^2 + 2(n+\tau)) - \tau = n^3/3 + 2n^2/3 + 2\tau n/3 - \tau$. □

Theorem 3.2. *The expected runtime of the Generalised Random Gradient hyper-heuristic, when initialised on the 0^n bit string, for GapPath is at most $n^3/3 + 2n^2/3 + 4\tau n/3 - \tau$.*

Proof. The Generalised Random Gradient mechanism chooses a mutation operator uniformly at random, and applies it for a period of τ iterations. If an improvement is produced, a new period of τ iterations using the same operator is started immediately.

Let T_1 and T_2 respectively be the number of iterations before a 1-bit or a 2-bit improvement is constructed when it is possible to do so. If the appropriate operator i is chosen (i.e. with probability 1/2), it may succeed with probability p_i in each iteration, or it may fail during all τ iterations with probability $(1-p_i)^\tau$ in which case the random choice is repeated. If the other operator is chosen, τ iterations are wasted before the random choice is repeated. Combined, using $q_i := 1 - p_i$ for brevity:

$$\begin{aligned} E(T_i) &= \frac{1}{2}\left(\left(\sum_{k=1}^{\tau} k q_i^{k-1} p_i\right) + q_i^\tau(\tau + E(T_i))\right) + \frac{1}{2}(\tau + E(T_i)) \\ &= \left(\sum_{k=1}^{\tau} k q_i^{k-1} p\right) + q_i^\tau E(T_i + \tau) + \tau \qquad (1) \\ &= \sum_{j=0}^{\infty}\left(\left(\sum_{k=1}^{\tau}(k + j\tau) q_i^{j\tau+k-1} p_i\right) + \tau q_i^{j\tau}\right) \qquad (2) \\ &= \left(\sum_{k=1}^{\infty} k q_i^{k-1} p_i\right) + \tau + \frac{\tau q_i^\tau}{1-q_i^\tau} = \frac{1}{p_i} + \tau + \frac{\tau q_i^\tau}{1-q_i^\tau}. \qquad (3) \end{aligned}$$

A simple rearrangement yields (1), and applying it recursively to the $q_i^{j\tau} E(T_i + j\tau)$ terms yields (2). As the inner sums are contiguous, a rearrangement in (3) simplifies the expression to an expectation of a geometrically-distributed variable and a sum of an infinite geometric series. As the expected number of periods of length τ where the correct operator is applied but does not produce an improvement is $\frac{q_i^\tau}{1-q_i^\tau}$, $\frac{\tau q_i^\tau}{1-q_i^\tau}$ is a lower bound on the number of iterations where a correct operator is applied before a success (i.e. $1/p_i$) and $\tau + \frac{\tau q_i^\tau}{1-q_i^\tau}$ is an upper bound on $1/p_i$. Substituting these bounds into Eq. (3) yields $2/p_i \le E(T_i) \le 2/p_i + \tau$.

After an improvement is constructed, the successful operator will run for an additional τ iterations while not being able to construct the next improvement. To reach the optimum from 0^n, improvements must be constructed by alternating 2-bit and 1-bit mutations $n/3$ times, while the final τ-iteration exploitation phase is unnecessary. The total expected optimisation time is therefore:

$$\frac{n}{3}(E(T_1) + E(T_2) + 2\tau) - \tau \le n^3/3 + 2n^2/3 + 4\tau n/3 - \tau.$$

□

3.1 Generalising the GapPath Function

Having shown that the generalised mechanisms are efficient on GP, we now introduce a function class on which it is necessary to learn. We will prove that the proposed generalised hyper-heuristics significantly outperform the standard mechanisms on these functions.

Let n be of the form $n = d(2k+1)$ for some $d, k \in \mathbb{N}$.

$$\mathrm{GGP}_k(x) := \begin{cases} \mathrm{ZM}(x) & \text{if } \textsc{Ridge}(x) \in S_k \\ \mathrm{ZM}(x) + 2n\textsc{Ridge}(x) & \text{otherwise} \end{cases}$$

where $S_k = \{c(2k+1) + 1 - 2\beta \mid c, \beta \in \mathbb{N} : c \leq d, \beta \leq k\}$, and $\mathrm{ZM}(x)$ and $\textsc{Ridge}(x)$ are defined as in Section 3.

This function is similar to the GapPath function, but requires $k \geq 3$ consecutive 2-bit successes after the 1-bit improvement is successfully executed at search points of the form $1^{(c-1)(2k+1)}0^{n-((c-1)(2k+1))}$ for $0 \leq c \leq d$. Hence learning to use 2-bit flips will lead to improved performance.

We modify the GGP_k function by increasing the fitness of a random non-GGP_k Ridge point within each group of k 2-bit flips, except for the first and last such points within each group. These "trap" points can be reached by a 1-bit mutation, while the next path point remains reachable only by a 2-bit mutation, and are local optima for the two mutation operators. If a trap point is ever constructed, the algorithm will not be able to reach the global optimum. We consider the success probabilities for the classical and generalised mechanisms on this modified function (denoted GGPT_k). Since there is a finite probability for any mechanism to construct a trap point, the expected runtimes will all be infinite by the law of total expectation.

Theorem 3.3. *Starting at 0^n, all of the classical selection mechanisms will fail to find the global optimum on $GGPT_k$ in finite time with probability at least $1 - n^{-\Omega(n)}$.*

Proof. We will show that when each mechanism is at a point from which a trap point is accessible, it will construct the trap point before the next path point with probability $1 - O(1/n)$. As there are at least $n/(2k+1) = \Omega(n)$ points from which a trap is accessible, the classical mechanisms construct a trap before reaching the global optimum with probability $1 - O(1/n)^{\Omega(n)} = 1 - n^{-\Omega(n)}$.

The Greedy mechanism applies both mutation operators in one iteration. The probability that a trap is constructed via a 1-bit mutation is $1/n$, while the probability that the next path point is constructed is $2/n^2$. If both are constructed, the mechanism accepts the trap point as it has higher fitness. The probability that a trap point is constructed before the next path point is therefore at least $\frac{1/n}{1/n+2/n^2} = 1 - \frac{2}{n+2} = 1 - O(1/n)$.

The Permutation mechanism, following a two-bit success leading to a point from which a trap is accessible, will perform 1-bit and 2-bit mutations in this order. The probability that a trap point is constructed over two iterations is thus $1/n$, while the probability that the next path point is constructed over two iterations is $(1 - 1/n) \cdot 2/n^2$. The probability that a trap point is constructed before the next path point is therefore $\frac{1/n}{1/n+(1-1/n)2/n^2} = \frac{1/n}{1/n+2/n^2} + \frac{2}{n^2+4n+2-4/n} = 1 - O(1/n)$.

The Simple Random mechanism constructs a trap point with probability $1/(2n)$, and the next path point with probability $1/n^2$. The probability that a trap point is constructed before the next path point is therefore at least $\frac{1/(2n)}{1/(2n)+1/n^2} = 1 - \frac{2}{n+2} = 1 - O(1/n)$.

The Random Gradient mechanism behaves equivalently to the Simple Random mechanism, unless it manages to construct an improvement by a two-bit mutation immediately following another two-bit improvement. The probability that an individual trap point is skipped in this manner is $1 - 2/n^2 = O(1/n)$, and thus the probability that the Random Gradient constructs the trap point before the next path point is $(1 - O(1/n))^2 = 1 - O(1/n)$. □

On the other hand, with sufficiently large τ, the generalised mechanisms can avoid the traps (w.o.p).

Theorem 3.4. *Starting at 0^n for $\tau = \Omega(n^3)$, the Generalised Random Gradient mechanism will find the global optimum of $GGPT_k$ (with constant k) in at most $\frac{4\tau n}{2k+1} + \frac{kn^3}{2(2k+1)} + O(n^2)$ steps with probability at least $1 - 2^{-\Omega(n)}$. However, for $\tau = O(n^2)$, the Generalised Random Gradient mechanism will fail to reach the global optimum of $GGPT_k$ in finite time with probability at least $1 - 2^{-\Omega(n)}$.*

Proof. The Generalised Random Gradient mechanism avoids making a decision inside the two-bit region if it is able to construct $k-1$ two-bit improvements within τ iterations each. The probability of success for a 2-bit mutation is $2/n^2$; the probability of no successes in τ steps is $(1 - 2/n^2)^\tau$ and the probability of (at least) one success in a period of τ steps is this value subtracted from 1. Thus, the probability that all $k-1$ two-bit improvements are found in time is $\left(1 - (1 - 2/n^2)^\tau\right)^{k-1}$.

For $\tau = cn^2$, where c is a constant, the probability of the required $k-1$ successful periods is $\left(1 - \left(1 - 2/n^2\right)^{cn^2}\right)^{k-1} = (1 - e^{-2c} - o(1))^{k-1} = \Theta(1)$, and so the probability of failing to do so is at least constant. If another random decision phase is performed inside the region of two-bit improvements, a trap is accessible via a 1-bit mutation from the point at which the decision occurs with probability at least $1/(k-1) = \Omega(1)$ (as traps are placed uniformly at random inside these regions), and is found by the Random Gradient mechanism with probability at least $1/2 \cdot (1 - 1/n)^{cn^2} \geq 1/4$. Thus, the probability that the no trap point is constructed in a single region of k two-bit improvements is at most a constant smaller than one, which means that over the $n/(2k+1)$ such regions, the probability that a trap point is constructed is at least $1 - 2^{-\Omega(n)}$.

For $\tau = cn^3$, where c is a constant, the probability that all $T = (k-1) \cdot n/(2k+1) = \Theta(n)$ two-bit successes are constructed within τ iterations is at least $\left(1 - \left(1 - 2/n^2\right)^{cn^3}\right)^T = (1 - e^{-2cn})^T \geq 1 - Te^{-2cn} = 1 - e^{-\Omega(n)}$ by applying a union bound. Thus, with overwhelming probability, the algorithm is not trapped, and the conditional expected optimisation time is

$$E(T_{GRG} \mid \text{not trapped}) \leq \frac{n}{2k+1}\left(n + 2\tau + \frac{k}{2}n^2 + 2\tau\right) = \frac{4\tau n}{2k+1} + \frac{kn^3}{2(2k+1)} + O(n^2),$$

with the 2τ terms corresponding to using the wrong operator for, in expectation, one period of τ iterations at the start of the phase,

and using the successful operator for τ iterations while the next improvement can only be constructed by the other operator. □

The following theorem can be proven using a similar approach.

Theorem 3.5. *Starting at 0^n for $\tau = \Omega(n^3)$, the Generalised Greedy mechanism will find the global optimum of $GGPT_k$ (with constant k) in at most $\frac{2\tau n}{2k+1} + \frac{n^3}{2k+1} + O(n^2)$ steps with probability at least $1 - 2^{-\Omega(n)}$. However, for $\tau = O(n^2)$, the Generalised Greedy mechanism will fail to reach the global optimum of $GGPT_k$ in finite time with probability at least $1 - 2^{-\Omega(n)}$.*

We point out that if the function values were inverted such that the trap points had optimal fitness, and the 1^n bit string did not, then the generalised mechanisms would fail to find any of the global optima (w.o.p), while the classical selection mechanisms would succeed (w.o.p). It is not surprising that learning mechanisms fail on a function especially designed to deceive them.

4 HYPER-HEURISTICS ARE FASTER

In this section we show that the generalised mechanisms can outperform low-level heuristics, even when the latter are efficient for the problem at hand. We consider the LeadingOnes function, $\mathrm{LO}(x) := \sum_{i=1}^{n} \prod_{j=1}^{i} x_j$, which counts the number of consecutive one-bits in a bit string before the first zero-bit.

It is well known that the standard (1+1) EA and RLS algorithms on LeadingOnes have expected runtimes of $\frac{e-1}{2}n^2 - o(n^2)$ and $0.5n^2$ fitness function evaluations respectively [4]. In the following we show that the generalised mechanisms are more efficient.

We first introduce a lower bound on the runtime of all algorithms which use only the 1BitFlip and 2BitFlip operators. Since the 1BitFlip operator has a success probability of $\frac{1}{n}$ while the 2BitFlip operator of $\frac{1}{n} \cdot \frac{n-i-1}{n} \cdot 2 = (2n - 2i - 2)/n^2$ where $i = \mathrm{LO}(x)$, it is easy to see that 2BitFlip is more effective for the first $n/2$ leading ones, while 1BitFlip is preferable afterwards. Hence, the expected runtime of an algorithm that uses these operators in such a way gives a trivial lower bound on all stochastic unbiased algorithms using the same operators. This expected runtime can easily be proved following the approach used for the (1+1) EA in [4].

Theorem 4.1. *The best possible expected runtime on LeadingOnes for a stochastic, unbiased algorithm using only the 1BitFlip and 2BitFlip operators is $\frac{1+\ln(2)}{4}n^2 + O(n) \approx 0.42329n^2 + O(n)$.*

4.1 Standard Mechanisms

In this section we show that the standard classical selection mechanisms all have the same runtime on LeadingOnes.

Theorem 4.2. *The expected runtime of the Simple Random mechanism on LeadingOnes for $p \in (0,1)$ is $\frac{1}{4(1-p)} \ln((2-p)/p)\, n^2 + o(n^2)$. If $p = 0$ the expected runtime is infinite. If $p = 1$, the expected runtime is $0.5n^2$.*

Proof. If $p = 0$, only the 2BitFlip operator is used. There is a finite probability of reaching the point $1^{n-1}0$, which cannot be improved by use of the 2BitFlip operator. Hence, by the law of total expectation, the expected runtime is infinite.

If $p = 1$, only the 1BitFlip operator is used, resulting in exactly RLS, which has expected runtime $0.5n^2$.

For $p \in (0,1)$, we can apply [4, Theorem 2] (since this mechanism leads to a stochastic, unbiased, Markovian, elitist algorithm) to bound $E(T_p) = \frac{1}{2}\sum_{i=1}^{n} A_{n-i}$, where A_{n-i} is the expected time needed to find an improvement given a solution with fitness i, and the algorithm is initialised with a search point chosen uniformly at random. In each iteration, the 1BitFlip operator is chosen with probability p; the 2BitFlip operator is chosen with probability $1-p$. Hence, $(A_{n-i})^{-1} = p \cdot \frac{1}{n} + (1-p) \cdot \frac{2n-2i-2}{n^2}$, and

$$A_{n-i} = \frac{1}{\frac{p}{n} + \frac{(1-p)(2n-2i-2)}{n^2}} = \frac{n^2}{2(1-p)(n-i-1) + np}$$

The total expected runtime $E(T_p)$ is, following [4, Theorem 2]:

$$\frac{1}{2}\sum_{i=0}^{n-1} \frac{n^2}{2(1-p)(n-i-1)+np} = \frac{n^2}{2}\sum_{k=1}^{n} \frac{1}{(2p-2)k + (2-p)n}$$

$$= \frac{n^2}{2(2p-2)}\left(\ln\left(\frac{n + \frac{(2-p)n}{2p-2}}{\frac{(2-p)n}{2p-2}}\right) + o(1)\right) = \frac{n^2 \ln((2-p)/p)}{4(1-p)} + o(n^2)$$

where in the equality before the last we used that $\sum_{k=1}^{n} \frac{1}{a\cdot k + b\cdot n} = \frac{1}{a}\left[\ln\left(\frac{n + bn/a}{bn/a}\right) + o(1)\right]$ with $a = 2p-2$ and $b = 2-p$. □

When $p = 0.5$, the Simple Random mechanism has expected runtime $\frac{\ln(3)}{2}n^2 + o(n^2)$. The runtime improves with increasing p, hence the optimal choice is $p = 1$.

Corollary 4.3. *The expected runtime of the Permutation, Greedy and Random Gradient mechanisms on LeadingOnes is $\frac{\ln(3)}{2}n^2 + o(n^2)$.*

Proof. For the Greedy and Permutation mechanisms, let $p_i = 1/n + (1-1/n)(2n-2i-2)/n^2$ be the probability that at least one improvement is constructed in respectively one or two iterations. To upper bound the optimisation time, we note that the difference between $2/p_i$ (i.e. the waiting time for an improvement to be constructed in terms of fitness evaluations) and the A_{n-i} waiting times used to prove Theorem 4.2 is at most constant, and thus the difference between these mechanisms and Simple Random expected times is limited to lower-order terms. We note that with probability at most $2/n^2$, both mutations considered are improvements; in such a case, we can consider an improvement step to be made in 0 iterations. As this occurs at most a constant number of times in expectation, and the maximum waiting time for any improving step is $O(n)$, the lower-bound differs from the upper bound by at most an $O(n)$ term. Thus, the expected runtime of these mechanisms is also $\ln(3)/2 \cdot n^2 + o(n^2)$.

For the Random Gradient mechanism, we note that the probability that an operator, when repeated following a success, is successful again is at most $2/n$. Since there are at most n improvements to be made throughout the search space, the expected number of repetitions which produce an improvement is at most 2. When not repeating a successful operator, the Random Gradient mechanism behaves identically to the Simple Random mechanism. Its expected runtime is therefore at least the runtime of the Simple Random mechanism less an $O(n)$ term, and at most the runtime

of the Simple Random mechanism plus n (as there are at most n iterations where the mechanisms differ in operator selection). Thus, its expected runtime is also $\ln(3)/2 \cdot n^2 + o(n^2)$. □

4.2 Generalised Mechanisms

We present a rigorous theoretical analysis of the generalised mechanisms on LeadingOnes.

Theorem 4.4. *The expected runtime of the Generalised Random Gradient algorithm on LeadingOnes with $\tau = cn$, where $c > 0$ is an appropriately chosen constant, is at most*

$$\frac{n^2}{2}\left(\sum_{j=1}^{w}\frac{2c + e^c M(1) + e^{2c(1-(j-1)/w)}M\left(\frac{0.5}{1-j/w}\right)}{(e^c + e^{2c(1-(j-1)/w)} - 2)w}\right) + o(n^2)$$

where $M(x) := \min(c, x)$, for any $w \in o(n)$.

We note that this bound decreases as w and c increase. For $\tau = 10n$ (i.e. $c = 10$), the bound is $\approx 0.48820n^2$ when $w = 5$, and $\approx 0.429529n^2$ when $w = 100$. Hence, the Generalised Random Gradient outperforms the classical selection mechanisms, as well as the well-known RLS and (1+1) EA on LeadingOnes. As w and c increase further, the resulting bounds on the leading constant are close to $\frac{1+\ln(2)}{4}n^2 \approx 0.423288n^2$, the optimal value derived in Theorem 4.1; for example, with $w = c = 1000$, the bound is $\approx 0.4232997n^2$.

Proof of Theorem 4.4. For the purpose of this proof, we partition the optimisation process into w stages based on the value of the LeadingOnes fitness function: during stage j, the LO value of the current solution is at least $(j-1)n/w$ and less than $j \cdot n/w$. After all w stages have been completed, the global optimum, with a LO value of n, has been found.

We upper-bound the expectation of the runtime T of the Generalised Random Gradient algorithm on LO by the sum of the expected values of T_j, the runtimes on each stage of the optimisation process. As our analysis of $E(T_j)$ requires the stage to start with a random choice of mutation operator, we bound $E(T) \leq \sum_{j=1}^{w}\left(E(T_j) + E(S_{j+1})\right)$ where S_{j+1} is a random variable denoting the expected number of iterations between the first solution in stage $j+1$ being constructed and the following random operator choice. We will later show that with proper parameter choices, the contribution of S_{j+1} terms can be bounded by $o(n^2)$, and therefore does not affect the leading constant in the overall bound.

Let us now consider T_j, the number of iterations the algorithm spends in stage j. Recall that a mutation operator is selected uniformly at random, and is allowed to run until it fails to produce an improvement for τ sequential iterations. Let N_j be a random variable denoting the number of random operator choices the algorithm performs during stage j, and $X_{j,1}, \ldots, X_{j,N_j}$ be the number of iterations the algorithm runs each chosen operator for. Then, $T_j = \sum_{k=1}^{N_j} X_{j,k}$, and by applying Wald's equation [9, 20], using $E(X_j)$ to denote an upper bound on all $E(X_{j,k})$ in stage j:

$$E(T_j) = \textstyle\sum_{k=1}^{N_j} E(X_{j,k}) \leq E(N_j)E(X_j). \tag{4}$$

To bound $E(N_j)$, the expected number of times the random operator selection is performed during stage j, we lower-bound the expected number of improvements found following operator selection, and apply the Additive Drift Theorem [12] to find the expected number of random operator selections occurring before a sufficient number of improvements have been found to enter the next stage.

Let F_1 and F_2 denote the events that the 1BitFlip and 2BitFlip operators fail to find an improvement during $\tau = cn$ iterations. For the 1BitFlip operator, this event occurs with probability $P(F_1) = (1-1/n)^{cn}$ throughout the process, which is within $[e^{-c} - 1/n, e^{-c}]$. For the 2BitFlip operator, recall that during stage j, the ancestor individual has at most $jn/w - 1$, and at least $(j-1)n/w$, one bits, and thus:

$$P(F_2) \leq (1 - 2 \cdot (1/n) \cdot (n - (jn/w - 1) - 1)/n)^{cn} \leq e^{-2c(1-j/w)},$$
$$P(F_2) \geq (1 - 2(1/n)(n - (j-1)n/w - 1)/n)^{cn}$$
$$> \left(1 - \frac{2(1-(j-1)/w)}{n}\right)^{cn} \geq e^{-2c(1-(j-1)/w)} - 1/n.$$

We note that a geometric distribution with parameter $p = P(F_1)$ can be used to model the number of improvements that the 1BitFlip operator finds prior to failing to find an improvement for τ iterations; the expectation of this distribution is $(1-p)/p = 1/p - 1$. Combined over both operators, the expected number of improvements D_j produced following a single random operator selection during stage j, $E(D_j)$, is greater than:

$$\frac{1}{2}\left(\frac{1}{P(F_1)} - 1\right) + \frac{1}{2}\left(\frac{1}{P(F_2)} - 1\right) \geq e^c/2 + e^{2c(1-j/w)}/2 - 1,$$

by inserting the upper bounds on $P(F_1)$ and $P(F_2)$. We use this expectation as the drift in the Additive Drift Theorem to upper-bound $E(N_j)$. Recall that each stage consists of advancing through at most n/w fitness values; as bits beyond the leading ones prefix and the first zero bit remain uniformly distributed, at most $n/(2w)$ improvements by mutation are required in expectation. If each step of a random process in expectation contributes $E(D_j)$ improvements by mutation, the expected number of steps required to complete stage j is at most:

$$E(N_j) \leq \frac{n/(2w)}{E(D_j)} \leq \frac{n}{(e^c + e^{c(2-2j/w)} - 2)w} \tag{5}$$

by the Additive Drift Theorem.

To bound $E(X_j)$, the expected number of iterations before a selected mutation operator fails to produce an improvement for τ iterations, we apply Wald's equation: let S be the number of improvements found by the operator before it fails, and $W_1, \ldots, W_S$ be the number of iterations it took to find each of those improvements. Then, once selected, the 1BitFlip operator runs for:

$$E(X_j \mid \text{1-bit}) = \tau + \sum_{k=1}^{S} E(W_k) = E(S)E(W_1 \mid S \geq 1, \text{1-bit})$$

where τ accounts for the iterations immediately before failure, and the sum for the iterations preceding each constructed improvement. Recall that $E(S) = 1/P(F_1) - 1$ by the properties of the geometric distribution, and observe that $E(W_1 \mid S \geq 1) = E(W_1 \mid W_1 \leq \tau) \leq \min(\tau, E(W_1))$. Using a waiting time argument for $E(W_1) = 1/P(F_1)$, we get (with a similar argument for the two-bit mutation operator):

$$E(X_j \mid \text{1-bit}) \leq \tau + (1/P(F_1) - 1)\min(\tau, n),$$
$$E(X_j \mid \text{2-bit}) \leq \tau + (1/P(F_2) - 1)\min\left(\tau, n/(2(1-j/w))\right).$$

Combining these conditional expectations with lower bounds on $P(F_1)$ and $P(F_2)$ yields

$$E(X_j) \leq 1/2 \cdot E(X_j \mid \text{1-bit}) + 1/2 \cdot E(X_j \mid \text{2-bit}) \leq \tau + \frac{\min(\tau, n)}{2(e^{-c} - 1/n)} + \frac{\min(\tau, n/(2-2j/w))}{2(e^{-2c(1-(j-1)/w)} - 1/n)}, \quad (6)$$

which for $\tau = cn$ can be simplified to:

$$E(X_j) \leq \frac{n}{2}\left(2c + \frac{\min(c,1)}{e^{-c} - 1/n} + \frac{\min(c, 1/(2-2j/w))}{e^{-2c(1-j/w+1/w)} - 1/n}\right) \leq \frac{n}{2}\Big(2c + e^c \min(c,1) + e^{2c(1-(j-1)/w)} \min\left(c, (2-2j/w)^{-1}\right)\Big) + O(1)$$

using $a/(b - c/n) = a/b + ac/(b^2 n - bc) = a/b + O(1/n)$, where a, b, and c are constants with respect to n, to limit the contributions of the $-1/n$ terms in denominators to lower-order terms.

Finally, we return to bounding the overall runtime of the algorithm. Recall that S_j denoted the number of iterations the algorithm spends in stage j while using the random operator chosen during stage $j-1$, and as $E(S_j) \leq \max\left(E(X_j \mid \text{1-bit}), E(X_j \mid \text{2-bit})\right) < 2E(X_j) = O(n)$, and $w = o(n)$, $\sum_{j=1}^{w} E(S_{j+1}) \in o(n^2)$. Substituting the bounds (5) and (6) into (4) yields the theorem statement:

$$E(T) \leq \sum_{j=1}^{w}\left(E(T_j) + E(S_{j+1})\right) \leq \left(\sum_{j=1}^{w} E(N_j)E(X_j)\right) + o(n^2) \leq o(n^2) + \frac{n^2}{2} \times \sum_{j=1}^{w} \frac{2c + e^c \min(c,1) + e^{2c(1-(j-1)/w)} \min\left(c, \frac{0.5}{1-j/w}\right)}{(e^c + e^{2c(1-(j-1)/w)} - 2)w}.$$

□

The following theorem can be proved using similar techniques and calculations, and applying the Variable Drift Theorem [13] instead of the Additive Drift Theorem.

Theorem 4.5. *The expected runtime of the Generalised Greedy algorithm on LeadingOnes with $\tau = cn$, where $c > 0$ is an appropriately chosen constant, is at most (up to $o(n^2)$ terms),*

$$\sum_{k=1}^{w}\left(\left(\frac{2w}{3w-2k} + c\right)n \int_{\frac{kn-n}{w}}^{\frac{kn}{w}} \frac{1}{\frac{2cn\cdot(5n^2-(8i+8)n+4(1+i)^2)}{n^2(3n-2i-2)} + 2} di\right)$$

for any $w = o(n)$.

This bound decreases for larger w, although our proof method imposes $w = o(n)$ as a requirement. For $w = 1$, the upper bound is $\approx 0.72474n^2$ when $\tau = n$ and $\approx 0.52169n^2$ when $\tau = 10n$. For $w = 100$, this improves to $\approx 0.51645n^2$ for $\tau = n$, and $\approx 0.48521n^2$ for $\tau = 10n$. That is, for a relatively low, linear, generalisation period of $\tau = n$, the Generalised Greedy mechanism improves upon the classical mechanisms. Increasing the generalisation period up to $\tau = 10n$ shows a significant improvement, even beating the single operator mechanism RLS. Interestingly, we note that as $c \to \infty$ (while remaining constant), the result from Theorem 4.5 provides a runtime of $n^2 \cdot \left(\frac{\ln(5)}{8} + \frac{\arctan(2)}{8}\right) \approx 0.47797n^2$, for any $w \in o(n)$.

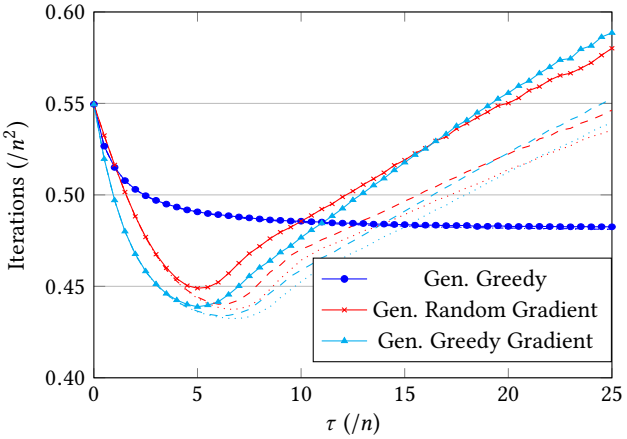

Figure 1: Average number of iterations required to find the LeadingOnes optimum for n = 10,000 (solid), n = 50,000 (dashed) and n = 100,000 (dotted), as τ increases.

5 EXPERIMENTAL RESULTS

In this section we discuss some experimental results for the Generalised Greedy (GG) and Generalised Random Gradient (GRG) mechanisms. We also consider a new mechanism called Generalised Greedy Gradient (GGG), which combines the decision stage of the GG mechanism and applies this operator choice to the descent stage of the GRG mechanism. The idea is to combine the learning aspects of both mechanisms with the aim of improving their runtime. All parameter combinations have been simulated 10,000 times.

Figure 1 shows the runtimes for the three mechanisms with n = 10,000, 50,000 and 100,000, illustrating the effect τ has on the runtime. We note that while the GG mechanism seems to be improving for increased τ consistently (seemingly converging towards $0.47797n^2$), the τ value for which the GRG and GGG mechanisms seem to reach their best performance increases with n. We see that for GRG and GG as n increases, the best τ value increases. At best, both mechanisms beat the experimental runtime observed in [8] for the recently presented k-bit mutation algorithm with self-adjusting k. This algorithm has an average experimental runtime of $0.450n^2$ for the given parameter choices in [8], with n = 10,000. We did not perform an experimental comparison with this algorithm as it is unclear how to set the various parameters for a fair comparison.

Figure 2 shows the effects of increasing the bit string length n for a fixed τ value ($5n$ and $10n$ respectively). While the GG mechanism seems to converge quite quickly at around $n = 10^5$ in both instances, the problem size required for the GRG and GGG mechanisms to converge increases with τ. As n increases, the experimental runtimes are below the bounds provided by Theorems 4.4 and 4.5. We postulate that larger τ values would see further improvements toward the optimum of Theorem 4.1 as n increases.

6 CONCLUSION

The theoretical analysis of hyper-heuristics is a growing field. Previous results have shown that the learning mechanisms applied in

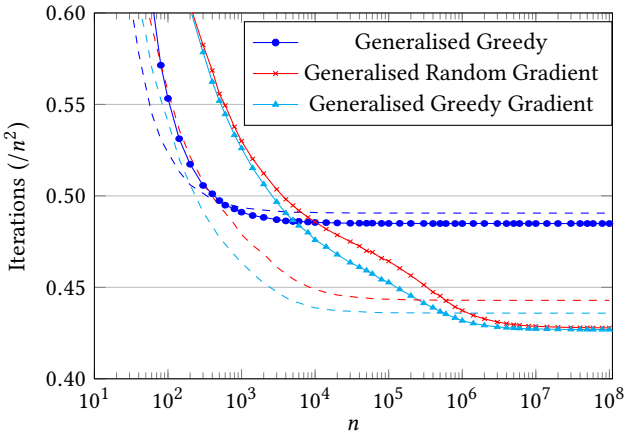

Figure 2: Average number of iterations required to find the LeadingOnes optimum for $\tau = 10n$ (solid) and $\tau = 5n$ (dashed), as n increases.

combinatorial optimisation applications show no signs of learning on LeadingOnes and perform similarly to choosing operators at random. We generalised the existing learning mechanisms to allow success to be measured over a longer period rather than in a single iteration, and proved that these generalised mechanisms are faster than all standard evolutionary and local search algorithms on LeadingOnes.

To apply the generalised algorithms, a value for the learning period τ is required. Although our results indicate that τ is a fairly robust parameter (i.e., for $n = 10{,}000$ all generalised mechanisms achieved faster experimental runtimes than that of the (1+1) EA for all tested values of τ between 1 and 1,000,000), setting it appropriately will lead to optimal performance. Concerning the Generalised Random Gradient mechanism, clearly τ must be large enough to have at least a constant expected number of successes within τ steps, if the algorithm has to learn about the operator performance. Obviously, setting large values of τ may lead to large runtimes, since switching operators requires $\Omega(\tau)$ steps. Similar considerations may also be made for the Generalised Greedy mechanism that experiments indicate is even more robust to the choice of τ. Concerning the negative effects that may occur for too large values of τ, the GapPath function can be considered as a worst-case scenario for the generalised mechanisms, because the operator has to change after every success. Nevertheless, we have proved that the algorithms are asymptotically just as efficient as the classical mechanisms for values of τ as large as $O(n^2)$.

The Generalised Greedy Gradient algorithm, which combines Generalised Greedy's decision stage with Generalised Random Gradient's repetition of successful exploitation phases, can achieve even better performance than either algorithm.

In future work we will analyse the mechanisms on a broader class of problems, including classical ones from combinatorial optimisation, and generalise the hyper-heuristics further by allowing the algorithms to automatically learn how to adapt τ.

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

REFERENCES

[1] Fawaz Alanazi and Per Kristian Lehre. 2014. Runtime analysis of selection hyper-heuristics with classical learning mechanisms. In *2014 IEEE Congress on Evolutionary Computation (CEC)*. Springer, 2515–2523. DOI: http://dx.doi.org/10.1109/CEC.2014.6900602

[2] Fawaz Alanazi and Per Kristian Lehre. 2016. Limits to Learning in Reinforcement Learning Hyper-heuristics. In *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2016)*. Springer International Publishing, Cham, 170–185. DOI: http://dx.doi.org/10.1007/978-3-319-30698-8_12

[3] Shahriar Asta and Ender Özcan. 2014. An apprenticeship learning hyper-heuristic for vehicle routing in HyFlex. In *IEEE Symposium on Evolving and Autonomous Learning Systems (EALS 2014)*. IEEE, 65–72.

[4] Süntje Böttcher, Benjamin Doerr, and Frank Neumann. 2010. Optimal Fixed and Adaptive Mutation Rates for the LeadingOnes Problem. In *Parallel Problem Solving from Nature (PPSN XI)*. Springer, Berlin, Heidelberg, 1–10. DOI: http://dx.doi.org/10.1007/978-3-642-15844-5_1

[5] 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.

[6] Peter Cowling, Graham Kendall, and Eric Soubeiga. 2001. A Hyperheuristic Approach to Scheduling a Sales Summit. In *Practice and Theory of Automated Timetabling III*. Springer, Berlin, Heidelberg, 176–190. DOI: http://dx.doi.org/10.1007/3-540-44629-X_11

[7] Peter Cowling, Graham Kendall, and Eric Soubeiga. 2002. Hyperheuristics: A Tool for Rapid Prototyping in Scheduling and Optimisation. In *Applications of Evolutionary Computing: EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN Kinsale, Ireland, April 3–4, 2002 Proceedings*. Springer, Berlin, Heidelberg, 1–10. DOI: http://dx.doi.org/10.1007/3-540-46004-7_1

[8] Benjamin Doerr, Carola Doerr, and Jing Yang. 2016. k-Bit Mutation with Self-Adjusting k Outperforms Standard Bit Mutation. In *Parallel Problem Solving from Nature (PPSN XIV)*. Springer International Publishing, Cham, 824–834. DOI: http://dx.doi.org/10.1007/978-3-319-45823-6_77

[9] Benjamin Doerr and Marvin Künnemann. 2013. How the (1+λ) evolutionary algorithm optimizes linear functions. In *Genetic and Evolutionary Computation Conference, GECCO '13, Amsterdam, The Netherlands, July 6-10, 2013*. 1589–1596. DOI: http://dx.doi.org/10.1145/2463372.2463569

[10] Jonathon Gibbs, Graham Kendall, and Ender Özcan. 2010. Scheduling english football fixtures over the holiday period using hyper-heuristics. In *Parallel Problem Solving from Nature (PPSN XI)*. Springer, Berlin, Heidelberg, 496–505.

[11] Jun He, Feidun He, and Hongbin Dong. 2012. Pure Strategy or Mixed Strategy?. In *Evolutionary Computation in Combinatorial Optimization: 12th European Conference (EvoCOP 2012)*. Springer, Berlin, Heidelberg, 218–229. DOI: http://dx.doi.org/10.1007/978-3-642-29124-1_19

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

[13] Daniel Johannsen. 2010. *Random combinatorial structures and randomized search heuristics*. Ph.D. Dissertation. Universität des Saarlandes, Postfach 151141, 66041 Saarbrücken. http://scidok.sulb.uni-saarland.de/volltexte/2011/3529

[14] Per Kristian Lehre and Ender Özcan. 2013. A Runtime Analysis of Simple Hyper-heuristics: To Mix or Not to Mix Operators. In *Proceedings of the Twelfth Workshop on Foundations of Genetic Algorithms XII (FOGA XII '13)*. ACM, New York, NY, USA, 97–104. DOI: http://dx.doi.org/10.1145/2460239.2460249

[15] Eunice López-Camacho, Hugo Terashima-Marin, Peter Ross, and Gabriela Ochoa. 2014. A unified hyper-heuristic framework for solving bin packing problems. *Expert Systems with Applications* 41, 15 (2014), 6876–6889.

[16] İbrahim Maden, Şima Uyar, and Ender Özcan. 2009. Landscape analysis of simple perturbative hyperheuristics. In *15th International Conference on Soft Computing*. Mendel, 16–22.

[17] Gabriela Ochoa, Rong Qu, and Edmund K. Burke. 2009. Analyzing the Landscape of a Graph Based Hyper-heuristic for Timetabling Problems. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO '09)*. ACM, New York, NY, USA, 341–348. DOI: http://dx.doi.org/10.1145/1569901.1569949

[18] Gabriela Ochoa, José Antonio Vázquez-Rodríguez, Sanja Petrovic, and Edmund Burke. 2009. Dispatching rules for production scheduling: a hyper-heuristic landscape analysis. In *IEEE Congress on Evolutionary Computation*. IEEE, 1873–1880.

[19] Ender Özcan, Mustafa Misir, Gabriela Ochoa, and Edmund K. Burke. 2010. A Reinforcement Learning - Great-Deluge Hyper-Heuristic for Examination Timetabling. *Applied Metaheuristic Computing* 1, 1 (2010), 39–59. DOI: http://dx.doi.org/10.4018/jamc.2010102603

[20] Abraham Wald. 1944. On Cumulative Sums of Random Variables. *Ann. Math. Statist.* 15, 3 (09 1944), 283–296. DOI: http://dx.doi.org/10.1214/aoms/1177731235

[21] David H Wolpert and William G Macready. 1997. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation* 1, 1 (1997), 67–82.