A Runtime Analysis of Simple Hyper-Heuristics: To Mix or Not to Mix Operators

Per Kristian Lehre
ASAP Research Group
School of Computer Science
University of Nottingham, UK
PerKristian.Lehre@nottingham.ac.uk

Ender Özcan
ASAP Research Group
School of Computer Science
University of Nottingham, UK
Ender.Ozcan@nottingham.ac.uk

ABSTRACT

There is a growing body of work in the field of hyper-heuristics. Hyper-heuristics are high level search methodologies that operate on the space of heuristics to solve hard computational problems. A frequently used hyper-heuristic framework mixes a predefined set of low level heuristics during the search process. While most of the work on such selection hyper-heuristics in the literature are empirical, we analyse the runtime of hyper-heuristics rigorously. Our initial analysis shows that mixing heuristics could lead to exponentially faster search than individual (deterministically chosen) heuristics on chosen problems. Both mixing of variation operators and mixing of acceptance criteria are investigated on some selected problems. It is shown that mixing operators is only efficient with the right mixing distribution (parameter setting). Additionally, some of the existing adaptation mechanisms for mixing operators are also evaluated.

Categories and Subject Descriptors

F.2 [**Theory of Computation**]: Analysis of Algorithms and Problem Complexity

General Terms

Theory, Algorithms

Keywords

Runtime Analysis, Hyper-heuristic, Stochastic Local Search

1. INTRODUCTION

Hyper-heuristic methodologies are learning mechanisms or search techniques that *mix* a prefixed set of user-defined heuristics (neighbourhood operators) or *generate* automatically new ones based on user-defined components [3]. This study focuses on the former type of selection hyper-heuristics. The key components in a single point based selection hyper-heuristic framework are identified as *heuristic selection* and *move acceptance* [25]. Usually, a high level hyper-heuristic aims to iteratively improve an initially generated complete solution by repeatedly selecting a low level perturbative neighbourhood operator, then using it to generate a new complete solution and finally deciding whether to accept or reject the new solution at each step. This heuristic selection process can be viewed as setting the probability of each low level operator for selection and application of a selected operator based on these probabilities to a candidate solution. A number of empirical studies indicate the success of selection hyper-heuristics which mix different heuristics when solving real world problems, ranging from Personnel Scheduling [7] to Vehicle Routing [27]. Additionally, Mısır [18] explored the behaviour of group decision making strategies which successfully mixed well known move acceptance operators and outperformed individual move acceptance operators when used as the selection hyper-heuristic components. More examples on the empirical success of hyper-heuristics can be found in [4].

Cowling et al. [7, 8] introduced hyper-heuristics as "heuristics to choose heuristics" and investigated the performance of *simple* hyper-heuristics across a set of real-world scheduling problem instances. The study focused on simple heuristic selection methods, such as, *Simple Random* and *Random Permutation*. These two methods do not learn, as they do not receive any feedback during the search process. Simple Random chooses a low level operator uniformly at random in each step, while Random Permutation uses a uniform random permutation of a given set of low level operators and successively uses a low level operator from that list. Two different types of acceptance strategies can be found in the literature: *deterministic* or *non-deterministic*. The authors used two deterministic acceptance operators which would make the same acceptance decision regardless of the given step during the search using the same current and new candidate solutions(s) in [7]: *All Moves* (AM) and *Only Improvements* (OI). AM accepts all solutions regardless of their quality (fitness), while OI accepts only improving moves and previous solution is used in the next step, when a non-improving move is made.

The learning heuristic selection methods can adapt during the search process improving their decision capability for the selection of *promising* neighbourhood operators. Nareyek [21] proposed *Reinforcement Learning* (RL) which scores each low level operator based on its individual performance. The probability of an operator being selected is updated depending on these scores. For example, roulette wheel strategy chooses a low level heuristic with a probability given by

FOGA'13, January 16-20, 2013, Adelaide, Australia.

the ratio of its individual score to the sum of scores of all low level operators. After the selected operator is applied to the current solution, the score of a low level operator is increased in the case of success indicating improvement in the quality of the new candidate solution and decreased in the case of a failure. There are many different selection hyper-heuristic components studied in the literature. Özcan et al. [26] investigated the memory length issues in reinforcement learning using a case study. Gibbs et al. [11] and Burke et al. [2] compared the performance of different reinforcement learning based heuristic selection methods when used within a selection hyper-heuristic framework for sports scheduling, examination timetabling, respectively. An overview of hyper-heuristics and more can be found in [28, 5, 4].

A significant trend in research on randomised search heuristics including evolutionary algorithms is the advent of rigorously proven results about their expected runtime [1, 22]. Runtime analyses show how the expected optimisation time of a search heuristic depends on its parameter settings and the characteristics of the optimisation problem at hand. Such analyses often provide deeper insights into the behaviour of search heuristics as compared to the empirical investigations.

There are some work on selection hyper-heuristics in the direction of gaining more insights regarding their behaviour via landscape analysis [17, 24, 23]. The theoretical work on selection hyper-heuristics is still very limited. Recently, He et al. [13] compared pure and mixed strategy (1+1) EAs using the so-called asymptotic hitting time as performance measure. A mixed strategy uses multiple mutation operators and chooses one based on a given fixed distribution, while a pure strategy uses a single mutation operator. The authors claimed that the asymptotic hitting time of a mixed strategy over a set of mutation operators O, is not worse than that of the worst pure strategy using one operator in O only.

The asymptotic hitting time considered by He et al. [13] is a different performance measure than the expected runtime, which is considered in this study. The theoretical work on the runtime analysis of selection hyper-heuristics is almost non existent. To the best knowledge of the authors, this study is one of the initial studies in the area. Our motivation is to illustrate that mixing (neighbourhood or acceptance) operators as components of a selection hyper-heuristic can be more efficient on certain problems. We perform runtime analyses of hyper-heuristics for choosing the right parameter setting for mixing low level operators. Moreover, the expected runtime of a simple reinforcement learning scheme and other mechanisms are compared.

1.1 Notation

The paper uses the following notation. For $n \geq 1$, define the set of integers $[n] := \{1, \ldots, n\}$, and $[0..n] := \{0\} \cup [n]$. Given a vector $v \in \mathbb{R}^n$ (e.g., a bitstring) and an integer $i \in [n]$, then v_i denotes the i-th element of v. The notation $X \sim D$ signifies that X is a random variable with distribution D. In particular, $D_p(a, b)$ with parameter $p \in [0, 1]$ is the distribution where $X \sim D_p(a, b)$ if and only if $\Pr[X = a] = p$ and $\Pr[X = b] = 1 - p$. The n-th harmonic number is denoted H_n. The runtime analysis uses standard notation (e.g., O, Ω and Θ) for asymptotic growth of functions (see, e.g., [6]).

2. MIXING NEIGHBOURHOOD OPERATORS

This section considers hyper-heuristics that mix neighbourhood operators (also called variation operators). The progress made in runtime analysis of evolutionary algorithms demonstrates the importance of completely understanding the simple algorithms and simple problems before proceeding to runtime analysis of more complex algorithms and more complex problems. We therefore consider a simple hyper-heuristic, Algorithm 1 below, which is a variant of the well-known (1+1) EA [9]. This algorithm is also similar to the randomised local search (RLS) variant used in Giel and Wegener [12].

Algorithm 1

1: $x \sim \textsc{Unif}(\{0,1\}^n)$
2: **while** termination criteria not satisfied **do**
3: Var $\sim D_{\bar{p}}(\textsc{Op}_1, \textsc{Op}_2, ..., \textsc{Op}_m)$ // Selects a neighbourhood operator.
4: $x' \sim \text{Var}(x)$
5: **if** $f(x') \geq f(x)$ **then** $x \leftarrow x'$.
6: **end while**

The RLS algorithm maintains a current solution x, from which a neighbouring candidate solution x' is generated in each iteration. The candidate solution replaces the current solution if it is not worse than the current solution. The (1+1) EA produces the candidate solution by flipping each bit in the current solution with probability $1/n$. In contrast, Algorithm 1 produces the candidate solution using one of the m neighbourhood operators. Based on the probability distribution $\bar{p}$, it selects and applies the operator $\textsc{Op}_i$ with probability p_i, for $i \in [m]$, where $\sum_{i=1}^{m} p_i = 1$.

We will use *fitness-based partitions*, which is a well known method for runtime analysis of randomised search heuristics.

Definition 1. *A tuple $(A_0, A_1, \ldots, A_\ell)$ is an f-based partition of a function $f : \{0,1\}^n \to \mathbb{R}$ if*

1. $\bigcup_{i=0}^{\ell} A_i = \{0,1\}^n$
2. $A_i \cap A_j = \emptyset$ *for* $\forall i, j$ *and* $i \neq j$
3. $f(A_i) < f(A_j)$ *for* $\forall i, j$ *and* $i < j$
4. $f(A_\ell) = \max_x f(x)$

The set A_i, $i \in [\ell]$ is referred to as the *i-th fitness level*. If the probability of leaving fitness level i is at least s_i, then the expected time to leave fitness level i is at most $1/s_i$.

Theorem 1. *Given any f-based partition, let s_i be the minimum probability for a (1+1) EA to leave A_i towards $A_{i+1} \cup \cdots \cup A_\ell$. The expected runtime of (1+1) EA on f is bounded from above by*

$$E\left(T_{(1+1)\ EA,f}\right) \leq \sum_{i=0}^{\ell-1} \frac{1}{s_i}$$

Theorem 1 can also be applied to the hyper-heuristic framework in Algorithm 1. Assuming that there is a fitness based partition for a given problem, Theorem 2 shows that the upper bound on the expected runtime of an algorithm mixing a set of operators as in Algorithm 1 can be computed based on the shortest expected running time for a solution to move from one fitness level to an upper level across all levels based on a given probability distribution.

THEOREM 2. *Given a function $f : \{0,1\}^n \to \mathbb{R}$ and an f-based partition $(A_0, \ldots, A_\ell)$, let T be the runtime of Algorithm 1 with parameters $p_i > 0, \forall i$ on the function f. For each $i \in [0..\ell-1]$, let r_i^k be the minimum probability for Algorithm 1 to leave fitness level A_i towards $A_{i+1} \cup \cdots \cup A_\ell$ using the mutation operator OP_k. Then, the expected runtime of Algorithm 1, is bounded from above by*

$$E\left(T_{Algorithm\ 1,f}\right) \leq \sum_{i=0}^{\ell-1} \min_{1\leq k\leq m} \frac{1}{p_k r_i^k} \leq \min_{1\leq k\leq m} \sum_{i=0}^{\ell-1} \frac{1}{p_k r_i^k} \quad (1)$$

PROOF. Use Theorem 1, with

$$s_i = \sum_{k=1}^{m} p_k r_i^k \geq \max_{1\leq k\leq m} p_k r_i^k,$$

for $i \in [0..m-1]$, and the theorem follows. □

Algorithm 2 (HH) is a special case of Algorithm 1, where $m = 2$:

Algorithm 2

1: $x \sim \text{UNIF}(\{0,1\}^n)$
2: **while** termination criteria not satisfied **do**
3: Var $\sim D_p(\text{RLSMOVEONEBIT}, \text{RLSMOVETWOBITS})$
// Selects a neighbourhood operator.
4: $x' \sim \text{Var}(x)$
5: **if** $f(x') \geq f(x)$ **then** $x \leftarrow x'$.
6: **end while**

With probability p, HH applies the 1-bitflip operator RLSMOVEONEBIT which flips one uniformly chosen bit position. With probability $1-p$, HH picks the 2-bitflip operator RLSMOVETWOBITS which flips two different and uniformly chosen bit-positions.

2.1 Analysis of the ONEMAX function

The following "warm-up" analysis describes the behaviour of a simple and standard hyper-heuristic which mixes heuristics on a familiar problem. We do not expect that a hyper-heuristic performs well in a case where the optimal setting is known, since hyper-heuristics are meant to be general methodologies that could be applied to a variety of problem domains.

The ONEMAX function is a well known benchmark function which counts the number of nonzero bits in the given bitstring

$$\text{ONEMAX}(x) := \sum_{i=1}^{n} x_i \quad (2)$$

THEOREM 3. *The expected runtime of Algorithm 2 on* ONEMAX *is*

$$E\left(T_{HH,\text{ONEMAX}}\right) \leq \begin{cases} (\frac{1}{p})n(\ln(n)+1) & \text{if } p > \frac{\ln(n)+1}{n+\ln n}, \text{ and} \\ \frac{1}{(1-p)}n^2 & \text{otherwise.} \end{cases}$$

PROOF. We apply Theorem 2 with the canonical partition $A_i := \{x \mid \text{ONEMAX}(x) = i\}$, for $i \in [0..n]$. In order for a solution to be improved to a higher fitness level, it is sufficient to flip a single 0-bit. The minimum probability for this event considering a candidate solution at the i-th fitness level is given by $r_i^1 = (n-i)/n$ or $r_i^2 = (n-i)(n-i-1)/n^2$ (assuming $i \leq n-2$, otherwise $r_i^2 = 0$) when RLSMOVEONEBIT or RLSMOVETWOBITS is selected, respectively. By Theorem 2,

$$\begin{aligned} E\left(T_{HH,\text{ONEMAX}}\right) &\leq \min\left\{\sum_{i=0}^{n-1}\frac{1}{pr_i^1}, \sum_{i=0}^{n-1}\frac{1}{(1-p)r_i^2}\right\} \\ &= \min\left\{\frac{1}{p}\sum_{i=0}^{n-1}\frac{n}{(n-i)}, \right. \\ &\qquad \left. \frac{1}{1-p}\sum_{i=0}^{n-2}\frac{n^2}{(n-i)(n-i-1)}\right\} \\ &= \min\left\{\frac{n}{p}H_n, \frac{n^2}{1-p}\sum_{j=1}^{n-1}\frac{1}{j(j+1)}\right\} \\ &\leq \min\left\{\frac{n}{p}(\ln(n)+1), \frac{n^2}{1-p}\left(1-\frac{1}{n}\right)\right\}. \end{aligned}$$

The first term on the right hand-side will be the minimum if

$$\begin{aligned} \frac{n}{p}(\ln(n)+1) &< \frac{n^2}{1-p}\left(1-\frac{1}{n}\right) \\ \frac{1}{p} - 1 &< \frac{n-1}{\ln(n)+1} \end{aligned}$$

and the theorem follows. □

2.2 Analysis of the GAPPATH function

As a simple benchmark, we use the GAPPATH function defined below.

$$\text{GAPPATH}(x) := \begin{cases} \text{ZM}(x) & \text{if } \text{RIDGE}(x) \equiv 1 \pmod 3 \\ \text{ZM}(x) + 2n\text{RIDGE}(x) & \text{otherwise.} \end{cases} \quad (3)$$

where $\text{ZM}(x) := \sum_{i=1}^{n}(1-x_i)$, and

$$\text{RIDGE}(x) := \begin{cases} i & \text{if } x = 1^i 0^{n-i} \text{ for } i \in [0..n] \\ 0 & \text{otherwise.} \end{cases}$$

In order not to unnecessarily complicate the analysis, we assume that $n = 3k$ for some integer $k \geq 1$.

The GAPPATH function is a variant of the SPI function (short path with increasing values on the path), which is a standard benchmark function in theoretical analysis of evolutionary algorithms [16, 15]. These functions feature a short path which corresponds to all search points on the form $1^i 0^{n-i}$. The function value of any search point outside the path is the number of 0-bits, and is inferior to the path members. The function GAPPATH differs from SPI, in that the path contains gaps where the function values are inferior to the rest of the path. The gaps correspond to all search points on the form $1^i 0^{n-1}$ where $i \equiv 1 \pmod 3$. A typical randomised search heuristic will optimise GAPPATH by first locating the search point 0^n, then by traversing along the path while "jumping" across the gaps.. Due to the gaps in the GAPPATH function, it is straightforward to see that using only a 1-bitflip operator, or only using a 2-bitflip operator will not be effective.

THEOREM 4. *If $p = 0$ or $p = 1$, then the expected runtime of Algorithm 2 on* GAPPATH *is infinite.*

PROOF. Setting $p = 0$ implies that the algorithm will only use the 2-bitflip operator. With probability $1/2^n$, the initial search point is $1^2 0^{n-2}$. All search points within Hamming distance 2 from this point have inferior function value. Hence, the algorithm will never accept any search point produced from the 2-bitflip operator. By the law of total probability, the expected runtime is therefore infinite. An analogous argument can be made for the case $p = 1$ with respect to the initial search point $1^3 0^{n-3}$ and the 1-bitflip operator. □

We will therefore consider mixing of operators, i.e., choosing $p \in (0, 1)$, and study how the choice of parameter p influences the expected runtime. We start by estimating the expected time to make a single improving step along the path.

LEMMA 1. *Let $p \in (0, 1)$. Assume that the current search point is $x = 1^j 0^{n-j}$, where $j \equiv i \pmod 3$ for $i \in \{0, 2\}$. Let T_i be the random time until a search point with strictly higher function value is obtained. Then, the expectation of T_i is*

$$E(T_i \mid p) = \begin{cases} \frac{n(n-1)}{2(1-p)}, & i = 0 \\ \frac{n}{p}, & i = 2 \end{cases}$$

PROOF. The probability that the 2-bitflip operator improves a search point where $\text{RIDGE}(x) \equiv 0 \pmod 3$ is $r_0 := (2/n)(1/(n-1))$, and the probability that a 1-bitflip operator improves a search point where $\text{RIDGE}(x) \equiv 2 \pmod 3$ is $r_2 := 1/n$. Hence,

$$E(T_0 \mid p) = \frac{1}{r_0(1-p)} = \frac{n(n-1)}{2(1-p)},$$
$$E(T_2 \mid p) = \frac{1}{r_2 p} = \frac{n}{p}.$$

□

It is now easy to calculate the expected runtime with a fixed mixing parameter p.

THEOREM 5. *The expected runtime of Algorithm 2 initialised with the search point 0^n on* GAPPATH *with fixed mixing parameter $p \in [0, 1]$ is $\left(\frac{n^3-3n^2}{6(1-p)} + \frac{n^2}{3(1-p)p}\right)$.*

PROOF. To obtain the optimum, it is necessary to transition from state $\text{RIDGE}(x) \equiv 0 \pmod 3$ to $\text{RIDGE}(x) \equiv 2 \pmod 3$ and back $n/3$ times. Hence, the expected runtime of the algorithm is

$$(n/3)\left(E(T_0 \mid p) + E(T_2 \mid p)\right)$$

where the expectations of T_0 and T_2 are given in Lemma 1. Hence,

$$\frac{n}{3}\left(\frac{n(n-1)}{2(1-p)} + \frac{n}{p}\right) = \frac{n^2}{3}\left(\frac{p(n-1)+2-p}{2(1-p)p}\right)$$

and the theorem follows. □

Theorem 5 implies that the expected runtime of Algorithm 2 is $O(\max\{n^2/p, n^3/(1-p)\})$. Moreover, if $p = n^{-c}$, where $c \geq 1$, then the expected runtime is $\frac{n^{c+2}}{3} + \frac{n^3}{6} + O(n^2)$. Similarly, if $p = 1 - n^{-c}$, then the expected runtime is $\frac{n^{c+3}}{6} - \frac{n^{c+2}}{6} + O(n^2)$.

We will revisit and use the GAPPATH function in Section 4.

3. MIXING ACCEPTANCE CRITERIA

This section considers hyper-heuristics that mix move acceptance operators. We show that mixing move acceptance operators can be efficient on problems where individual acceptance operators fail.

Algorithm 3

1: $x \sim \text{UNIF}(\{0,1\}^n)$
2: **while** termination criteria not satisfied **do**
3: $\quad x' \sim \text{FLIPRANDOMBIT}(x)$
4: $\quad \text{ACC} \sim D_p(\text{AM}, \text{OI})$ // Selects an acceptance operator.
5: $\quad$ **if** $\text{ACC}(x, x')$ **then** $x \leftarrow x'$
6: **end while**

Algorithm 3 follows the same outline as Algorithm 2, but differs in the choice of neighbourhood operator and move acceptance operator. Algorithm 3 always uses the 1-bitflip operator as neighbourhood operator, and it mixes two different move acceptance operators. With probability p, it chooses the $\text{AM}(x, x')$ acceptance operator (always move) which always accepts the candidate solution x'. With probability $1 - p$, it uses the $\text{OI}(x, x')$ acceptance operator (only improvement) which accepts the candidate solution x' if and only if $f(x') > f(x)$, ie., it is strictly better than the current candidate solution.

The pseudo-Boolean function RR_k, also called the *royal road function* [29], splits a bitstring of length n into $m := \lfloor n/k \rfloor$ blocks of length k. A block containing exactly i 0-bits, for $i \in [0..k]$ is called an *i-block*. The function value of a bitstring equals the number of 0-blocks in the bitstring.

$$\text{RR}_k(x) := \sum_{i=1}^{\lfloor n/k \rfloor} \prod_{j=1}^{k} x_{k(i-1)+j}.$$

The simplest case $k = 1$ equals to the well-known ONEMAX function. The problem becomes harder for larger values of k. The RR_k function is similar but not identical to, the Royal Road function defined in [19].

We will first show that it is necessary to mix the acceptance criteria, i.e. choosing $p \in (0, 1)$, in order to optimise RR_k efficiently. By setting $p = 0$, Algorithm 3 will only accept improving moves, which is problematic because RR_k with $k \geq 2$ contains large plateaus.

THEOREM 6. *If $p = 0$, then the expected optimisation time of Algorithm 3 on RR_k with $k \geq 2$ is infinite.*

PROOF. Any search point that is obtained by flipping any of the first k positions in the bitstring $x := 0^2 1^{n-2}$ has the same function value as x, and will not be accepted by the OI-operator. Any search point that is obtained by flipping any of the $n - k$ last bit positions in x will have inferior function value, and will not be accepted by the OI-operator. The initial search point is x with probability $1/2^n$. The expected runtime on RR_k is therefore infinite by the law of total probability. □

It is obvious that Algorithm 3 degenerates into a random walk over the set of bitstrings when $p = 1$. Mixing the acceptance operators is therefore necessary, and a natural question is to determine the right parameter setting for p. By intuition, if p is chosen too high, the algorithm will accept too many worsening moves. In fact, Theorem 8 which will be

presented later, shows that the expected runtime is super-polynomial, even for as small parameter settings as $p = \omega(\log(n)/n)$. The following theorem shows how to set p to guarantee polynomial expected runtime.

THEOREM 7. *For any integer $k \geq 2$, let $m := \lfloor n/k \rfloor$. If $0 < p \leq 1/(6mk^k)$, then the expected runtime of Algorithm 3 on RR_k is no more than $m(1+mk^k)/p$. If additionally $p = \Omega(1/mk^k)$, then the expected runtime is $O(m^3k^{2k})$.*

PROOF. Consider the vector-valued stochastic process $X_t \in \mathbb{N}^{k+1}$, where $X_t = v$ if the current search point in iteration t consists of v_i blocks with exactly i 0-bits for all $i \in [k]$. The drift of this process will be analysed with respect to the distance function $g(v) = \sum_{i=0}^{k} f_i v_i$, where for all $i \in [0..k]$

$$f_i := \begin{cases} 0 & \text{if } i = 0 \\ \frac{k}{2m} & \text{if } i = 1 \\ \frac{k}{2m} + \sum_{j=2}^{i} k^{k+2-j} & \text{if } 2 \leq i \leq k. \end{cases}$$

Note that $f_{i+1} - f_i = k^{k+1-i}$ and $f_{i+1} - f_{i-1} = k^{k+1-i}(1+k)$ for all $i \in [2, k-1]$. Obviously, the distance function g is non-negative and bounded from above by

$$B := k/2 + mk^{k+1} > mf_k.$$

Define the random variable

$$\Delta_t(v) := (g(X_{t-1}) - g(X_t) \mid X_{t-1} = v).$$

For each pair $i, j \in [k]$, where $i \neq j$, let the random variable $M_t(i,j)$ denote the number of i-blocks in iteration $t-1$ that was turned into j-blocks in iteration t. For each $i \in [k]$, let $M_t(i,i) = 0$. Now, define the conditional random variables

$$\Delta_t^{(i)}(v) := \left(\sum_{j=0}^{k} (f_i - f_j) M_t(i,j) \mid X_{t-1} = v \right)$$

Intuitively, the random variable $\Delta_t^{(i)}$ describes the drift contribution from i-blocks, so that the total drift can be decomposed as

$$\begin{aligned} \Delta_t(v) &= \sum_{i=0}^{k} f_i v_i - \sum_{i=0}^{k} f_i X_t(i) \\ &= \sum_{i=0}^{k} f_i v_i - \sum_{i=0}^{k} f_i \left(v_i + \sum_{j=0}^{k} M_t(j,i) - \sum_{j=0}^{k} M_t(i,j) \right) \\ &= \left(\sum_{i=0}^{k} f_i \sum_{j=0}^{k} M_t(i,j) \right) - \left(\sum_{i=0}^{k} f_i \sum_{j=0}^{k} M_t(j,i) \right) \\ &= \sum_{i=0}^{k} \sum_{j=0}^{k} (f_i - f_j) M_t(i,j) = \sum_{i=0}^{k} \Delta_t^{(i)}(v). \end{aligned}$$

The drift contribution from the 0-blocks is

$$\Delta_t^{(0)}(v) = -p \cdot (v_0/m)(k/2m) \geq -pk/(2m).$$

The drift contribution from the 1-blocks is

$$\Delta_t^{(1)}(v) = \frac{v_1}{km} \left(\frac{k}{2m} - (k-1)pk^k \right) \geq \frac{v_1}{2m} \left(\frac{1}{m} - 2pk^k \right).$$

The drift contribution from the i-blocks with $i \in [2, k-1]$ is

$$\begin{aligned} \Delta_t^{(i)}(v) &= \frac{pv_i}{m} \left(\frac{i(f_i - f_{i-1})}{k} + \frac{(k-i)(f_i - f_{i+1})}{k} \right) \\ &= \frac{pv_i}{m} \left(\frac{i(f_{i+1} - f_{i-1})}{k} - (f_{i+1} - f_i) \right) \\ &= \frac{pv_i}{m} k^{k+1-i} \left(\frac{i(1+k)}{k} - 1 \right) \\ &= \frac{pv_i}{m} k^{k+1-i} \left(\frac{i}{k} \right) \geq \frac{pv_i k}{m}. \end{aligned}$$

Finally, the drift contribution from the k-blocks is

$$\Delta_t^{(k)}(v) = p \frac{v_k}{m} k^2.$$

Two cases are now distinguished. In the case when $v_1 \geq 1$,

$$\begin{aligned} \Delta_t(v) \geq \Delta_t^{(1)}(v) + \Delta_t^{(0)}(v) &\geq \frac{v_1}{2m} \left(\frac{1}{m} - 2pk^k \right) - \frac{pk}{2m} \\ &\geq \frac{1}{2m} \left(\frac{1}{m} - p(2k^k + k) \right) \\ &> \frac{1}{2m} \left(\frac{1}{m} - \frac{1}{2m} \right) = \frac{1}{4m^2}, \end{aligned}$$

using the assumption $p \leq 1/(6mk^k) < 1/(2m(2k^k+k))$.

In the second case, where $v_1 = 0$, and $v_0 \leq m-1$, then

$$\begin{aligned} \Delta_t(v) &\geq \Delta_t^{(0)}(v) + \Delta_t^{(2)}(v) + \cdots + \Delta_t^{(k)}(v) \\ &\geq \frac{kp}{m} \left(v_2 + \cdots + v_k - \frac{1}{2} \right) \geq \frac{kp}{2m}. \end{aligned}$$

In both cases, $\Delta_t(v) \geq \Delta := kp/(2m)$ holds. Given that $g(X_t) \leq B$, it follows by the polynomial drift theorem (see eg. [14]) that the expected time until $g(X_t) = 0$ is no more than

$$B/\Delta \leq \left(k/2 + mk^{k+1} \right) (2m)/(kp) \leq m(1+mk^k)/p. \quad \square$$

We then show that if the parameter p in Algorithm 3 is chosen asymptotically larger than $\log(m)/(km)$, then the expected runtime on the RR_k problem is super-polynomial. This theorem also covers the special case $p = 1$, stating that Algorithm 3 becomes ineffective when the AM acceptance operator is used alone.

THEOREM 8. *If $p > g(m)/(km)$ for any function $g(m) \leq m/2$, then the expected runtime of Algorithm 3 on RR_k is $e^{\Omega(g(m))}$.*

PROOF. Assume optimistically that the number of 1-blocks never increases above $g(m)/2+1$, and that all other blocks are 0-blocks. Clearly, this will only make Algorithm 3 optimise RR_k faster. If the number of 1-blocks is no more than $g(m)/2$, then the probability of increasing the number of 1-blocks is at least $q := (3/4)p \leq (m - g(m)/2)p/m$ and the probability of decreasing the number of 1-blocks is no more than $r := (1/2)p \geq g(m)/(2km)$.

The algorithm is modelled by the stochastic process $X_t \in \mathbb{N}$ on the integers where $\Pr[X_t = X_{t-1} + 1] = q/(r+q)$ and $\Pr[X_t = X_{t-1} - 1] = r/(r+q)$. Assume that $X_0 = g(m)/2$. By the gambler's ruin problem (see [10]), it follows that the probability that the process reaches the point 0 before first reaching the point $g(m)/2 + 1$ is

$$\frac{\left(\frac{q}{r}\right) - 1}{\left(\frac{q}{r}\right)^{g(m)/2} - 1} = \frac{\frac{1}{2}}{\left(\frac{3}{2}\right)^{g(m)/2} - 1} = e^{-\Omega(g(m))}.$$

It is therefore clear, that if the algorithm at some point has $g(m)/2$ 1-blocks, then the probability that the algorithm reaches 0 1-blocks before reaching $g(m)/2+1$ 1-blocks is also $e^{-\Omega(g(m))}$. By a Chernoff bound (see eg. [20]), the number of 0-blocks in the initial search point is less than $m-g(m)/2-1 \geq (3/4)m-1$ with probability $1-e^{-\Omega(m)}$. Hence, the expected number of times the current search point has $m-g(m)/2-1$ 0-blocks before reaching the optimum is $e^{\Omega(g(m))}$, which concludes the proof. □

The result in Theorem 8 is complemented with a plot of the drift field corresponding to the number of 1-blocks and 2-blocks in the case $k=2$. With i 1-blocks, and j 2-blocks, these drifts are respectively

$$D_1(i,j) = \frac{(m-i-j)p}{m} + \frac{jp}{m} - \frac{ip}{2m}$$
$$D_2(i,j) = \frac{ip}{2m} - \frac{jp}{m}$$

When the mixing parameter p is too high, the algorithm accepts too many worsenings, and the current search point drifts towards an equilibrium point which is far from the global optimum at origin, as shown in Fig. 1.

4. ADAPTATION OF THE MIXING PARAMETER

The previous two sections have shown that mixing operators can be more efficient than each individual operator. However, the results also show that the efficiency of the mixing approach depends on having an appropriate mixing distribution. Theorems 7 and 8 tell us that the expected runtime (i.e., exponential vs polynomial) depends critically on the mixing parameter p, and that the right choice of p is problem-dependant (in the case of RR_k, it depends on parameter k).

We therefore consider mechanisms for online adaptation of the mixing parameter p in the context of neighbourhood operators. For simplicity, we assume a scenario where Algorithm 2 can choose between two parameter settings, $p = p_{\mathrm{hi}}$ or $p = p_{\mathrm{lo}}$, where $0 \leq p_{\mathrm{lo}} < 1/2 \leq p_{\mathrm{hi}} \leq 1$. Setting $p = p_{\mathrm{hi}}$ implies that the 1-bitflip operator will be chosen most often, while setting $p = p_{\mathrm{lo}}$ implies that the 2-bitflip operator will be chosen most often. The *permutation* mechanism deterministically chooses $p = p_{\mathrm{hi}}$ in even iteration numbers, and $p = p_{\mathrm{lo}}$ in odd iteration numbers. The *simple reinforcement learning* mechanism sets $p = p_{\mathrm{hi}}$ if the last improving step was made with the 1-bitflip operator, and $p = p_{\mathrm{lo}}$ if the last improving step was made with the 2-bitflip operator. The simple reinforcement approach is also commonly referred to as *random permutation gradient* hyper-heuristic in the literature [7, 4].

THEOREM 9. *The expected runtime of Algorithm 2 on* GAPPATH *with simple reinforcement learning with parameters p_{hi} and p_{lo}, starting with $p = p_{hi}$ from the search point 0^n is*

$$\frac{n}{3}\left(\frac{n(n-1)}{2(1-p_{hi})} + \frac{n}{p_{lo}}\right)$$

PROOF. Starting from $\textsc{Ridge}(x) = 0$, the first improving step will be made by the 2-bitflip operator. In all subsequent iterations, the algorithm will choose the parameter setting $p = p_{\mathrm{hi}}$ when $\textsc{Ridge}(x) \equiv 0 \pmod 3$, and $p = p_{\mathrm{lo}}$ when

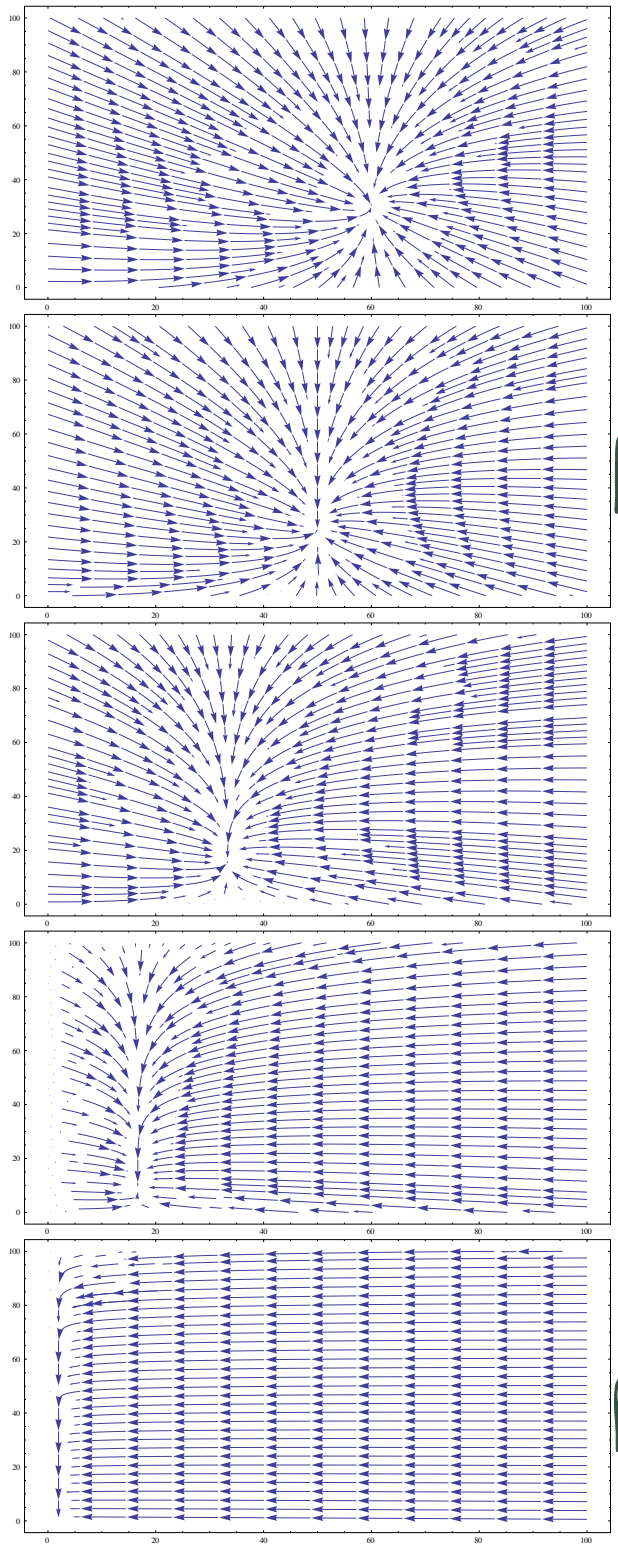

Figure 1: The drift of the number of 1-blocks (horizontal) and the number of 2-blocks (vertical) in the current search point of Algorithm 3 on the Royal Roads function (RR_k) when $k=2$, $m=100$, and (from top to bottom) $p=3/4$, $p=1/2$, $p=1/4$, $p=1/10$, and $p=1/100$. The global optimum 1^n is located at origin.

RIDGE$(x) \equiv 2 \pmod 3$. Hence, the expected optimisation time is by Lemma 1

$$\frac{n}{3}\left(E\left(T_0 \mid p = p_{\text{hi}}\right) + E\left(T_2 \mid p = p_{\text{lo}}\right)\right) = \frac{n}{3}\left(\frac{n(n-1)}{2(1-p_{\text{hi}})} + \frac{n}{p_{\text{lo}}}\right)$$

□

Finally, we have the following theorem for the permutation mechanism.

THEOREM 10. *The expected runtime of Algorithm 2 on* GAPPATH *with the permutation mechanism where* $p_{lo} = 1 - p_{hi} = 1/n$ *is no more than* $(1/3)n^3 + (1/3)n^2 + O(1)$ *when starting with the search point* 0^n.

PROOF. Let r be the probability of making an improving step, assuming that the correct mutation operator is selected. Let q_{hi} and q_{lo} be the probabilities of selecting the correct mutation operator given that $p = p_{\text{hi}}$, respectively $p = p_{\text{lo}}$. Since there are only two operators to choose from, it holds that

$$q_{\text{hi}}q_{\text{lo}} = (1 - p_{\text{lo}})p_{\text{lo}} = \frac{1}{n}\left(1 - \frac{1}{n}\right).$$

Starting from an even iteration number, the probability of making at least one improvement within the next two iterations is

$$q_{\text{hi}}r + (1 - q_{\text{hi}}r)q_{\text{lo}}r = r(1 - q_{\text{hi}}q_{\text{lo}}r) = r(1 - O(r/n)),$$

So the expected number of iterations to make an improvement is no more than

$$(2/r)(1 + O(r/n)) = 2/r + O(1/n).$$

In the case where RIDGE$(x) \equiv 0 \pmod 3$, the probability of making an improvement with the 2-bitflip operator is $r = (2/n)(1/(n-1))$. Hence, it holds that

$$E\left(T_0\right) \leq n(n-1) + O(1/n).$$

In the case where RIDGE$(x) \equiv 2 \pmod 3$, the probability of making an improvement with the 1-bitflip operator is $r = 1/n$. Hence it holds that

$$E\left(T_2\right) \leq 2n + O(1/n).$$

Following the same argument as above, the expected runtime on the problem is no more than

$$(n/3)\left(E\left(T_0\right) + E\left(T_2\right)\right) \leq (n/3)\left(n^2 + n + O(1/n)\right) = n^3/3 + n^2/3 + O(1).$$

□

Starting from a random solution will require an additional running time to obtain 0^n (see Section 2.1). This process is exactly the same as solving ONEMAX in which the 0-bits are counted instead of 1-bits. In order to emphasise the remaining expected time, this additional runtime is omitted in Table 1.

5. CONCLUSION

Hyper-heuristic studies have been mostly empirical. This paper provides one of the first runtime analyses of selection hyper-heuristics. We have shown that mixing different neighbourhood or move acceptance operators can be more efficient than using stand-alone individual operators in some cases. However, the analysis also shows that the performance of the mixing operators rely critically on having the right mixing distribution, which is problem dependent. Table 1 shows the result of an analysis of some well-known mechanisms under certain settings. Most notably, a previous successful mechanism which reinforces the operator performs poorer than the static mixing distributions. This is just a first step towards more rigorous analysis of hyper-heuristics. Future work should consider more complex hyper-heuristics, and more complex problem scenarios.

Acknowledgements

This work was supported in part by EPSRC under grant no. grant EP/F033214/1.

6. REFERENCES

[1] A. Auger and B. Doerr. *Theory of Randomized Search Heuristics: Foundations and Recent Developments*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2011.

[2] E. Burke, G. Kendall, M. Mısır, and E. Özcan. Monte carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, 196:73–90, 2012.

[3] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward. A classification of hyper-heuristic approaches. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research and Management Science*, pages 449–468. Springer, 2010.

[4] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu. Hyper-heuristics: A survey of the state of the art. Technical Report NOTTCS-TR-SUB-0906241418-2747, School of Computer Science, University of Nottingham, 2010.

[5] K. Chakhlevitch and P. I. Cowling. Hyperheuristics: Recent developments. In *Adaptive and Multilevel Metaheuristics*, volume 136 of *Studies in Computational Intelligence*, pages 3–29. Springer, 2008.

[6] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. McGraw Hill, New York, NY, 2nd edition, 2001.

[7] P. Cowling, G. Kendall, and E. Soubeiga. A hyper-heuristic approach to scheduling a sales summit. In *Practice and Theory of Automated Timetabling III : Third International Conference, PATAT 2000*, volume 2079 of *LNCS*. Springer, 2000.

[8] P. Cowling, G. Kendall, and E. Soubeiga. Hyperheuristics: A tool for rapid prototyping in scheduling and optimisation. In S. Cagoni, J. Gottlieb, E. Hart, M. Middendorf, and R. Goenther, editors, *Applications of Evolutionary Computing: Proceeding of Evo Workshops 2002*, volume 2279 of *Lecture Notes in Computer Science*, pages 1–10, Kinsale, Ireland, April 3-4 2002. Springer-Verlag.

Mechanism	Expected Runtime
Simple permutation	$(1/3)n^3 + (1/3)n^2 + O(1)$
Static parameter $p = 1/n$	$(1/2)n^3 + O(n^2)$
Static parameter $p = 1 - 1/n$	$(1/6)n^4 - (1/6)n^3 + O(n^2)$
Simple reinforcement $p_{\text{lo}} = 1 - p_{\text{hi}} = 1/n$	$(1/6)n^4 + (1/6)n^3$
Static $p = 1$ or $p = 0$	∞

Table 1: Impact of parameter update mechanism on the runtime of Algorithm 2 on the GapPath function. The mechanisms are ordered w.r.t. increasing runtime.

[9] S. Droste, T. Jansen, and I. Wegener. On the analysis of the (1+1) Evolutionary Algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

[10] W. Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, New York, 1968.

[11] J. Gibbs, G. Kendall, and E. Özcan. Scheduling english football fixtures over the holiday period using hyper-heuristics. In *Proceedings of the 11th international conference on Parallel problem solving from nature: Part I*, PPSN'10, pages 496–505, Berlin, Heidelberg, 2010. Springer-Verlag.

[12] O. Giel and I. Wegener. Maximum cardinality matchings on trees by randomized local search. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, GECCO '06, pages 539–546, New York, NY, USA, 2006. ACM.

[13] J. He, F. He, and H. Dong. Pure strategy or mixed strategy? - an initial comparison of their asymptotic convergence rate and asymptotic hitting time. In J.-K. Hao and M. Middendorf, editors, *Evolutionary Computation in Combinatorial Optimization - 12th European Conference, EvoCOP 2012, Málaga, Spain, April 11-13, 2012. Proceedings*, volume 7245 of *Lecture Notes in Computer Science*, pages 218–229. Springer, 2012.

[14] J. He and X. Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35, 2004.

[15] T. Jansen. On the analysis of dynamic restart strategies for evolutionary algorithms. In J. J. M. Guervós, P. Adamidis, H.-G. Beyer, J. L. F.-V. Martín, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature PPSN VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 33–43. Springer Berlin / Heidelberg, 2002.

[16] T. Jansen and I. Wegener. Evolutionary algorithms - how to cope with plateaus of constant fitness and when to reject strings of the same fitness. *IEEE Transactions on Evolutionary Computation*, 5(6):589–599, 2001.

[17] I. Maden, A. S. Uyar, and E. Özcan. Landscape analysis of simple perturbative hyper-heuristics. In *Mendel 2009: 15th International Conference on Soft Computing*, pages 16–22, 2009.

[18] M. Mısır. Group decision making in hyper-heuristics. Master's thesis, Department of Computer Engineering, Yeditepe University, 2008.

[19] M. Mitchell, S. Forrest, and J. H. Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. In F. J. Varela and P. Bourgine, editors, *Proceedings of the First European Conference on Artificial Life, 1991*, pages 245–254, Paris, 11–13 1992. MIT Press.

[20] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[21] A. Nareyek. Choosing search heuristics by non-stationary reinforcement learning. In M. G. C. Resende, J. P. de Sousa, and A. Viana, editors, *Metaheuristics*, pages 523–544. Kluwer Academic Publishers, Norwell, MA, USA, 2004.

[22] F. Neumann and C. Witt. *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.

[23] G. Ochoa, R. Qu, and E. K. Burke. 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, pages 341–348, New York, NY, USA, 2009. ACM.

[24] G. Ochoa, J. A. Vázquez-Rodríguez, S. Petrovic, and E. Burke. Dispatching rules for production scheduling: a hyper-heuristic landscape analysis. In *Proceedings of the Eleventh conference on Congress on Evolutionary Computation*, CEC'09, pages 1873–1880, Piscataway, NJ, USA, 2009. IEEE Press.

[25] E. Özcan, B. Bilgin, and E. E. Korkmaz. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis*, 12:3–23, 2008.

[26] E. Özcan, M. Mısır, G. Ochoa, and E. Burke. A reinforcement learning – great-deluge hyper-heuristic for examination timetabling. *International Journal of Applied Metaheuristic Computing (IJAMC)*, 1(1):39–59, 2010.

[27] D. Pisinger and S. Ropke. A general heuristic for vehicle routing problems. *Computers and Operations Research*, 34:2403– 2435, 2007.

[28] P. Ross. Hyper-heuristics. In E. K. Burke and G. Kendall, editors, *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, chapter 17, pages 529–556. Springer, 2005.

[29] T. Storch and I. Wegener. Real royal road functions for constant population size. *Theoretical Computer Science*, 320(1):123–134, 2004.