Journal of Heuristics
https://doi.org/10.1007/s10732-018-09404-7

An analysis of heuristic subsequences for offline hyper-heuristic learning

W. B. Yates[1] · E. C. Keedwell[1]

Received: 20 May 2018 / Revised: 5 December 2018 / Accepted: 14 December 2018

Abstract

A selection hyper-heuristic is used to minimise the objective functions of a well-known set of benchmark problems. The resulting sequences of low level heuristic selections and objective function values are used to generate a database of heuristic selections. The sequences in the database are broken down into subsequences and the mathematical concept of a *logarithmic return* is used to discriminate between "effective" subsequences, which tend to decrease the objective value, and "disruptive" subsequences, which tend to increase the objective value. These subsequences are then employed in a sequenced based hyper-heuristic and evaluated on an unseen set of benchmark problems. Empirical results demonstrate that the "effective" subsequences perform significantly better than the "disruptive" subsequences across a number of problem domains with 99% confidence. The identification of subsequences of heuristic selections that can be shown to be effective across a number of problems or problem domains could have important implications for the design of future sequence based hyper-heuristics.

Keywords Machine learning · Hyper-heuristics · Offline learning

1 Introduction

Search and optimisation methods such as *metaheuristics* (see Blum and Roli 2003) and *hyper-heuristics* (see Burke et al. 2013) are typically employed to solve computationally hard optimisation problems. Such algorithms move through a search space by applying sequences of perturbations to a set of existing solutions. Although the mech-

✉ W. B. Yates
wy254@exeter.ac.uk

E. C. Keedwell
E.C.Keedwell@exeter.ac.uk

[1] College of Engineering, Mathematics and Physical Sciences, University of Exeter, Exeter EX4 4QF, UK

Published online: 04 January 2019

anisms for selecting a solution to perturb, and the frequency with which perturbation operations are applied can vary greatly between algorithms, there exists an identifiable sequence of perturbations for a single solution or population of solutions. For some algorithms, the perturbations and therefore the sequence is fixed. For example, an *evolutionary algorithm* will execute a mutation operation and crossover operation at a given rate for every iteration of the algorithm. In this case, there is little to be said about the sequences of perturbations which are generated by these types of algorithm. However, in the case of a selection hyper-heuristic which can select from a wide range of potential perturbations in any order it chooses, the issue of context and the notion of sequence becomes important. For example, a disruptive perturbation such as a partial randomisation of a solution when paired with an exploitative perturbation such as a local search in that order might yield improved solutions. The randomisation moves the solution to a new, unexplored region of space and the local search finds the best solution in that neighbourhood. However, the reverse (local search followed by randomisation) is likely to be a poor strategy, with the majority, if not all, of the work carried out by the local search being discarded. From this small example it becomes clear that the correct ordering of a sequence is imperative if search efficacy is to be achieved.

This paper is concerned with ordered subsets or *subsequences* of low level heuristics. The role of subsequences in the search and optimisation process is investigated via the generation and analysis of a large database of perturbation operations across a number of problems and problem domains. The results of this analysis demonstrate that it is possible to statistically separate "effective" subsequences of perturbations, which tend to decrease the objective function value which is to be minimised, from "disruptive" subsequences of perturbations, which tend to increase the objective value, across most, but not all, problem domains.

Such effective subsequences can be used to improve hyper-heuristics performance either directly, by embedding them in a suitable hyper-heuristic design, or indirectly as the inputs to an appropriate hyper-heuristic learning algorithm. Furthermore, by comparing effective subsequences across different problem domains it is possible to investigate the potential for cross-domain learning.

The statistical approach presented here establishes that subsequences of perturbations are important structural parts of the heuristic search space and that successful selection hyper-heuristics must consider the context in which a low level heuristic is to be executed before making their selection.

1.1 Hyper-heuristics

A *heuristic* is a process or method that learns to employ loosely defined rules or trial and error in order to solve a given problem. Heuristics are applied to computationally *hard* problems where no known effective algorithmic solution exists. Typically such problems are presented as optimisation problems where the goal is to minimise an *objective function* f defined on a space X of solutions.

In their seminal work on the job scheduling problem Fisher and Thompson (1963) and Crowston et al. (1963) demonstrate that an unbiased random combination of job scheduling heuristics outperforms any individual job scheduling heuristic and that it is possible to employ “probabilistic learning” to improve performance further. The idea of applying a combination or permutation of heuristics to a problem to find a better solution gives rise to the concept of higher level heuristics such as metaheuristics (see Blum and Roli 2003) and more recently hyper-heuristics (see Burke et al. 2013). Metaheuristics and hyper-heuristics differ in that most metaheuristics search the space X of problem solutions, whereas hyper-heuristics search the space S of heuristics. In practice this means that a metaheuristic can have access to problem specific information, while a hyper-heuristic is subject to the limitations of the *domain barrier* and is unable to access problem specific information. The domain barrier requires the hyper-heuristic to perform well in the absence of problem specific information and therefore, it is hoped, to be “re-useable” across different problem domains with minimal changes.

Hyper-heuristics are intended to either *generate* or *select* low level heuristics. A generation hyper-heuristic generates new heuristics by discovery, or by modifying or combining existing low level heuristics. Selection hyper-heuristics, such as those developed in this paper, must select and apply a heuristic chosen from a set of low level heuristics. The goal of both types of hyper-heuristics is to improve the search process through learning and/or optimisation.

1.2 Offline learning

Many selection hyper-heuristics employ learning algorithms to improve optimisation performance. Typically, the learning algorithm optimises a hyper-heuristic’s internal structure and parameters in order to improve the selection of heuristics based on the current (generic) state of a problem. Such learning can be categorised as either *online* or *offline* (see Burke et al. 2013).

Online learning is based on the low level heuristic selections and resulting objective function values computed during the execution of a hyper-heuristic. The objective is to improve optimisation performance on the problem at hand. Many algorithms have been employed successfully for online learning such as genetic algorithms (Fang et al. 1993), hill climbing (Gratch and Chien 1996), the choice function (Cowling et al. 2001), Tabu search (Burke et al. 2003), fuzzy systems (Asmuni et al. 2005), reinforcement learning (Özcan et al. 2010), and adaptive selection probabilities (Soria-Alcaraz et al. 2014).

In contrast, offline learning is performed on a database of low level heuristic selections and objective function values computed by a hyper-heuristic on a fixed number of benchmark problems. The objective is to *generalise* across the benchmark training problems leading to improvements in optimisation performance on unseen test problems. A variety of machine learning algorithms have also been proposed for offline learning such as classifier systems (Ross et al. 2002), case based reasoning (Burke et al. 2006), messy genetic algorithms (Terashima-Marín et al. 2008), and the *ILSParam* parameter tuning algorithm employed in Soria-Alcaraz et al. (2014). In Ross et al. (2002) a classifier system is applied to the 1D Bin Packing problem. The system is trained on a number of benchmark problems and learns a set of rules which associate characteristics of a current problem state with specific heuristics. Rules are selected according to the problem state and the associated heuristics are applied sequentially. When evaluated on a set of unseen problems, the evolved rule-set was able to produce an optimal solution for over 78% of the test cases, and in the rest it produced a solution very close to optimal. In Burke et al. (2006) case based reasoning is applied to exam timetabling problems. Previous problems and their "good" solutions (called source cases) are collected and stored. A similarity based retrieval process compares the source cases with the problem at hand, and selects heuristics that were employed successfully in similar situations. The results demonstrate that the "knowledge of heuristics" discovered offline helped in selecting the "best" heuristics during the problem solving process, and yielded higher quality solutions. In Terashima-Marín et al. (2008) the authors present a messy genetic algorithm that produces a population of hyper-heuristics to solve the dynamic variable ordering problem which occurs in the field of constraint satisfaction. Each of the chromosomes of the genetic algorithm represents a selection hyper-heuristic consisting of a set of condition-action rules that encode a problem's state and an associated low level heuristic. After a training phase, when evaluated on unseen examples, these hyper-heuristics solved many of the test problems very efficiently, and in a few cases, produced better results than the best single heuristic for the problem instance. In Soria-Alcaraz et al. (2014) offline and online learning are combined to optimise an Iterated Local Search (ILS) hyper-heuristic for the course timetabling problem. ILS uses heuristic selection probabilities to choose low level heuristics, and these selection probabilities are estimated offline using the *ILSParam* algorithm, and then adapted online. The best-performing hyper-heuristic produced competitive results when compared to the state-of-the-art on the well known ITC-2007,[1] benchmark test set, even producing a new best-known solution.

These four examples demonstrate that offline learning is able to improve hyper-heuristic performance by generalising over the problems of a domain. However, the methodologies employed differ markedly and are dictated by the choice of learning algorithm and/or the choice of problem state space representation. The framework for offline learning presented here does not depend on either of these factors.

1.3 Heuristic subsequences

Most of the online and offline learning research cited above aims to improve the selection of *single* heuristics (or heuristic pairs). Some research (see for example Ortiz-Bayliss et al. 2013; Kheiri and Keedwell 2015, 2017; Yates and Keedwell 2017, 2018) has argued that heuristic selections should be understood as part of a *sequence* of selections. As noted in Sect. 1, the concept of heuristic sequences is intuitive, certain heuristic orderings make sense whereas others do not, and it should be noted that such orderings can be discovered automatically online (see Kheiri and Keedwell 2015, 2017), or offline (see Yates and Keedwell 2017, 2018). In addition, adopting a sequence based approach can reduce the number of objective function evaluations required- in the introductory example there is no need to evaluate the result of the randomisation. This is beneficial because, for many real world problem domains and problem sizes, evaluating the objective function is computationally expensive.

[1] The 2007 International Timetabling Competition (http://www.cs.qub.ac.uk/itc2007/).

There are several examples in the literature of empirical research concerning subsets of low level heuristics such as Petrovic and Epstein (2008), Mısır et al. (2012) and Soria-Alcaraz et al. (2017), and from a purely theoretical perspective (Lehre and Özcan 2013). For example, in Petrovic and Epstein (2008) subsets of heuristics are randomly selected from a large pool of available heuristics. Each subset is evaluated on a number of benchmark problems, and a learning algorithm is used to determine weights for the elements of the subset. These weights determine an individual heuristic's probability of selection. In Mısır et al. (2012) various subsets of heuristics, heuristic selection mechanisms and acceptance strategies are evaluated on a number of patient admission scheduling problems. The experimental results demonstrate that different subsets of low level heuristics produce very different optimisation performances across a number of hyper-heuristic designs. In Soria-Alcaraz et al. (2017) non-parametric statistical tests and fitness landscape measurements are used to rank and select subsets of low level heuristics according to their performance on a set of benchmark problems.

In each case, the objective is to discover small subsets of effective heuristics that work well together and improve optimisation performance for the problems of a domain. However, the concept of heuristic sequence is absent, as the order in which the heuristics in a subset are applied is determined randomly. In contrast, Ortiz-Bayliss et al. (2013) construct ordered sets or subsequences of low level heuristic pairs to solve constraint satisfaction problems. Each possible permutation of the heuristics in a subsequence is evaluated on a set of benchmark problems, and the best performing permutation is selected. This permutation is then applied repeatedly to optimise test problems. Here the concept of heuristic sequence is explicit. However the methodology presented is only suitable for the analysis of short subsequences of small numbers of heuristics.

In this research, subsequences of heuristic selections are analysed statistically in order to distinguish between effective and disruptive subsequences in a manner that does not depend on the number of heuristics or the length of the subsequences.

1.4 Overview

A single selection hyper-heuristic is run on a number of benchmark problems in order to generate a database of low level heuristic selections and objective function values. The sequences of heuristic selections are broken down into subsequences and the mathematical concept of a *logarithmic return* is used to construct a statistical framework for the categorisation of these subsequences. Specifically, a statistical method is used to discriminate between effective and disruptive subsequences. These subsequences are then employed in a distinct sequence based selection hyper-heuristic and are evaluated on unseen examples of the benchmark problems. This allows for the differential performance of heuristic subsequences to be demonstrated empirically.

The results are compared with those produced by the SSHH hyper-heuristic (see Kheiri and Keedwell 2015). The SSHH hyper-heuristic is a sequenced based selection hyper-heuristic which employs online learning of low level heuristics and their parameters. The results for SSHH are included to provide a comparison between an online learning hyper-heuristic and the offline methodology presented here.

The identification of subsequences of heuristic selections that can be shown to be effective across a number of problems has important implications for the design of future sequence based hyper-heuristics. Effective subsequences can be used to improve hyper-heuristic performance either directly, by embedding them in a suitable hyper-heuristic design, or indirectly as the inputs to an appropriate offline learning algorithm.

Furthermore, by comparing effective subsequences across different problem domains it is possible to investigate the potential for cross-domain learning by, for example, attempting to identify a set of effective cross-domain subsequences.

The objective of this research is to assess the scope and limitations of the proposed statistical framework, and demonstrate empirically its predictive capability on a range of operational research problems.

2 Experimental methodology

In this section, the experimental methodology employed in this paper is presented.

2.1 Hard problems and the HyFlex framework

All the algorithms developed in this paper are trained and tested on the Hyper-heuristics Flexible framework (or HyFlex, see Ochoa et al. 2012). HyFlex is a set of benchmark problems that has been used in a number of studies (see for example Walker et al. 2012; Drake et al. 2012; Mısır et al. 2013; Drake et al. 2015; Kheiri and Keedwell 2015; Dempster and Drake 2016). HyFlex[2] is an implementation of six computationally hard problem domains:

1. 1D bin packing (BP),
2. permutation flowshop (PFS),
3. Boolean satisfiability (SAT),
4. personnel scheduling (PS),
5. the travelling salesman problem (TSP), and
6. the vehicle routing problem (VRP).

The first four domains are used for experimentation here. The TSP and VRP domains are not used in this study and are reserved for future experiments.

Each HyFlex problem domain contains 10 distinct problems of varying complexity. HyFlex hides all problem specific information such as the solution representations, the solution constructions, and the low level heuristic implementations. Each domain has 4 general *classes of heuristic*:

1. parameterised mutation (M) which perturbs a solution randomly,
2. crossover (C) which constructs a new solution from two or more existing solutions,
3. parameterised ruin and recreate (R) which destroys a given solution partially and then rebuilds the deleted parts, and

[2] HyFlex, Cross-domain Heuristic Search Challenge, CHeSC 2014, was used in this study. However this library is no longer available, and those wishing to replicate this study should use CHeSC 2011 (http://www.asap.cs.nott.ac.uk/chesc2011/).

Table 1 The low level heuristics for each domain

Dom.	Heuristics
BP	{M0, R1, R2, M3, L4, M5, L6, C7}
PFS	{M0, M1, M2, M3, M4, R5, R6, L7, L8, L9, L10, C11, C12, C13, C14}
SAT	{M0, M1, M2, M3, M4, M5, R6, L7, L8, C9, C10}
PS	{L0, L1, L2, L3, L4, R5, R6, R7, C8, C9, C10, M11}

4. parameterised hill climbing or local search (L) that incorporates an iterative improvement process and returns a non-worsening solution.

The number and implementation of the *low level heuristics* in each class differs between problem domains (see Table 1). It should be emphasised that the heuristic M0 in the BP domain is an entirely different heuristic to M0 in the PFS domain. However, the general underlying principles of each heuristic class should remain similar across domains. For example, a mutation operation should make random changes, while a local search operation will search the surrounding space.

The local search heuristics differ from the other classes in two respects. Firstly, they may require more that one objective function evaluation, and secondly, they cannot produce a solution of worse fitness. The 4 HyFlex domains considered in this study employ "first-improvement" local search heuristics. These heuristics iteratively apply neighbourhood functions which typically permute two or more elements of a solution in some way. For example, in the BP domain, one of the neighbourhood functions is to take the largest piece from the lowest filled bin, and exchange it with a smaller piece from a randomly selected bin. During a local search, at each iteration, a neighbour is generated using the neighbourhood function, and it is accepted immediately if it has superior or equal fitness. If the neighbour is worse, then the change is not accepted. The number of iterations, and therefore the number of objective function evaluations employed by these heuristics is determined by the "depth of search" parameter.

For each problem domain there is also a real valued, non-negative, *objective function* denoted f. The objective function induces an ordering on the solutions $x \in X$ of a problem, such that given any two solutions x_1 and x_2,

$$f(x_1) < f(x_2)$$

implies that x_1 is a better solution than x_2.

2.2 The DBGen hyper-heuristic

Subsequences of heuristics are extracted from a database of low level heuristic selections and objective function values. This database in generated by a selection hyper-heuristic denoted DBGen which is designed to sample the space of possible heuristic selections.

A selection hyper-heuristic consists of a set of low level heuristics, a *selection* strategy, and an *acceptance* strategy (see Burke et al. 2013). The selection strategy selects a particular low level heuristic s and its parameters $p = (q_1, \ldots, q_k)$. Given a solution x the low level heuristic computes a new solution $x' = s(x, p)$. The acceptance strategy, which is a (possibly stochastic) Boolean function, then decides whether to accept or reject the new solution. If the new solution is accepted, x' becomes the current solution, and $f(x')$ becomes the current objective function value, otherwise x' is discarded and x continues as the current solution. Iterating this process, starting from some given initial solution x_0, for n selections or iterations gives rise to sequences of

1. low level heuristic selections $s = s_1, \ldots, s_n$,
2. low level heuristic parameters $p = p_1, \ldots, p_n$,
3. solutions $x_0, x_1, \ldots, x_n$, and
4. objective function values $f(x_0), f(x_1), \ldots, f(x_n)$ also denoted $o_0, o_1 \ldots, o_n$.

These sequences represent a trace of the hyper-heuristic's execution on a given problem x_0 over n time steps.

The unbiased, random, single selection hyper-heuristic DBGen used to generate the database of heuristic selections and objective function values is shown in listing (1). The function *selectHeuristic()* (line 9) selects a single heuristic class at random from the set { C, L, R, M }. The function *apply()* (line 10) takes the heuristic class and chooses, again at random, a low level heuristic and its parameters, from the available heuristics of that class. The low level heuristic is then applied to the current solution, and if the class is C, to the current crossover solution. An objective function evaluation (line 11) and an acceptance check (lines 12–19) are then performed. The function *ran()* (lines 12 and 16) returns a uniformly distributed pseudorandom number in the interval (0, 1). If a new solution's objective value is less than the current solution's objective value or *ran()* < 0.5 then it is accepted. Otherwise the new solution is rejected. The random term allows new solutions to be accepted regardless of their objective function 50% of the time. Accepting states that may lead to a large increase in objective function value forces the DBGen hyper-heuristic to explore the space of low level heuristic selections instead of optimising the problem efficiently.

When crossover heuristics are available, the choice of crossover mechanism also affects hyper-heuristic performance (see Drake et al. 2015). The DBGen crossover mechanism is a simplification of the crossover management scheme employed by AdapHH (see Drake et al. 2015) which uses a population of five potential crossover solutions including the current best solution. The DBGen crossover mechanism employs a population of one solution (lines 12–15).

2.3 An offline subsequence database

The DBGen hyper-heuristic is run on the HyFlex problems in order to generate a database of low level heuristic selections and objective function values. Specifically, DBGen is executed 40 times on each of the 40 HyFlex problems for 150 iterations; a total of 1600 runs and 240,000 heuristic selections and objective function values. To put these figures into perspective, as the set of low level heuristic classes {C, L, R, M} contains four symbols there are $4^{150} \approx 2.0370 \times 10^{90}$ unique sequences of length 150.

Algorithm 1 The DBGen hyper-heuristic in pseudocode.

```
ITERATIONS $\leftarrow$ 150
new-sol $\leftarrow$ initialiseSolution()
new-obj $\leftarrow$ $f$(new-sol)
cross-sol $\leftarrow$ initialiseSolution()
cross-obj $\leftarrow$ $f$(cross-sol)
while (ITERATIONS $>$ 0) do
    cur-sol $\leftarrow$ new-sol
    cur-obj $\leftarrow$ new-obj
    Heuristic h $\leftarrow$ selectHeuristic()
    new-sol $\leftarrow$ apply(h, new-sol, cross-sol)
    new-obj $\leftarrow$ $f$(new-sol)
    if (new-obj $<$ cross-obj or ran() $<$ 0.5) then
        cross-sol $\leftarrow$ new-sol
        cross-obj $\leftarrow$ new-obj
    end
    if (new-obj $\geq$ cur-obj and ran() $\geq$ 0.5) then /* reject the new solution */
        new-sol $\leftarrow$ cur-sol
        new-obj $\leftarrow$ cur-obj
    end
    ITERATIONS $\leftarrow$ ITERATIONS $-$ 1
end
```

Each DBGen run is seeded by a single unique number defined by

$$seed = 4000 + (40\,p) + r$$

where $p = 0, \ldots, 39$ is the problem index and $r = 0, \ldots, 39$ is the run index.

The number of 40 runs was chosen so as to ensure that robust statistics could be calculated for each problem. The number of iterations or run length of 150 was chosen for computational feasibility, although results for much longer runs also show similar effects (see Sect. 4.6).

Each distinct sequence or run is then broken down into consecutive subsequences of length $n = 2, 3, \ldots, 149$. For example, given a sequence $\{\texttt{MCLLR}\}$ of length 5 note that the sets of subsequences of lengths 2, 3, and 4 are

$$\{\texttt{MC}, \texttt{CL}, \texttt{LL}, \texttt{LR}\}\ \{\texttt{MCL}, \texttt{CLL}, \texttt{LLR}\} \text{ and } \{\texttt{MCLL}, \texttt{CLLR}\}$$

respectively. The finite set of all such subsequences is denoted S.

2.4 Logarithmic returns

Each problem domain has its own objective function f, and the range of f may differ between problems and problem domains. Without a priori knowledge of the objective functions, the objective function values from different problems or problem domains cannot be compared directly. Instead, normalised subsequences of objective function values are compared. Consider a subsequence of objective function values $o_0, o_1, \ldots, o_n$ observed after applying a subsequence $s \in S$ of n low level heuristics to some initial solution x_0. The log objective values[3] are

[3] Objective functions that can produce 0 values must be suitably transformed so as to remove them.

$$\log(o_0), \log(o_1), \ldots, \log(o_n).$$

The *log returns* of this series are simply the sequential differences of the log objective values

$$\log(o_1) - \log(o_0),\ \log(o_2) - \log(o_1),\ \ldots, \log(o_n) - \log(o_{n-1})$$

and such subsequences are invariant to scaling.

Each low level heuristic s_i in $s = s_1, \ldots, s_n$ is associated with a log return. The sum of the n log returns is equal to the log return over the whole subsequence, since all but the first and last log objective values cancel out. In symbols

$$\sum_{i=1}^{n} \big(\log(o_i) - \log(o_{i-1})\big) = \log\left(\frac{o_n}{o_0}\right).$$

This is not the case for subsequences of *decimal returns*.[4] Although subsequences of decimal returns are scale invariant, they cannot be easily added or subtracted because the denominators of each return may differ. Furthermore, decimal returns are not symmetric, that is, a change d of x, followed by a change $-d$ of x does not return the original value x.

The *log return* α of a subsequence s of length n is defined by

$$\alpha(s) = \log_{10}\left(\frac{o_n}{o_0}\right).$$

The *unit log return* β of a subsequence s of length n is defined by

$$\beta(s) = \frac{1}{n}\alpha(s).$$

The length of a subsequence is important because for many real world optimisation applications the execution times of the low level heuristics and objective function evaluations can be non-trivial. The unit log return β induces an ordering on the subsequences $s \in S$. Given any two subsequences s_1 and s_2,

$$\beta(s_1) < \beta(s_2)$$

implies that s_1 is a better subsequence than s_2.

[4] The formulae to convert between $\log_{10}$ returns r and decimal returns d, are

$$r = \log_{10}(d+1) \quad \text{and} \quad d = 10^r - 1.$$

2.5 Selecting subsequences with the γ-ratio

The unit log return of a *set* of N subsequences is

$$\beta(\{s_1, \ldots, s_N\}) = \sum_{i=1}^{N} \beta(s_i).$$

Let S^+ be the set of all subsequences with $\alpha(s) > 0$, and let S^- be the set of all subsequences with $\alpha(s) \leq 0$, and note that $S = S^+ \cup S^-$. The positive and negative parts of β can be separated out by $\beta = \beta^+ - \beta^-$ where

$$\beta^+(U) = \beta(U \cap S^+) \quad \text{and} \quad \beta^-(U) = -\beta(U \cap S^-)$$

for every subset U of S.

A subsequence s may occur a number of times in the database and each occurrence will have a different subsequence of objective function values depending on the problem, the run, and the position in a run where s arises. The set

$$U_s = \{s^1, \ldots, s^{N_s}\}$$

is the set of all occurrences of a subsequence, where N_s is the number of occurrences of s. The function $\beta^+(U_s)$ measures the propensity of subsequence s to increase the objective value, while $\beta^-(U_s)$ measures the propensity of subsequence s to decrease the objective value. The larger the measure, the larger the propensity for change in the objective value. The word propensity is used deliberately in order to emphasise that applying a subsequence with a large β^+ (or β^-) could still lead to a decrease (or increase) in the objective function value. Indeed, it is quite possible for a subsequence to have a large (or small) measure under both β^+ and β^- and still produce a small (or large) change. The probability (estimate) that s produces a negative (unit) log return is

$$P(\alpha(s) \leq 0) = \frac{N_s^-}{N_s}$$

where N_s^- is the number of occurrences of s where $\alpha(s) \leq 0$.

The functions β, β^+, β^-, and $P(\alpha(s) \leq 0)$ can be used to categorise and select subsequences from S. In this study the γ*-ratio*, defined by

$$\gamma(U_s) = \frac{\beta^-(U_s)}{\beta^+(U_s) + 1}$$

is used to select subsequences from the offline database. Values of $\gamma(U_s) > 1$ indicate an effective subsequence while values of $\gamma(U_s) < 1$ indicate a disruptive sequence. The γ-ratio is used in preference to the *mean unit log return*

$$\overline{\beta}(\{s_1, \ldots, s_N\}) = \frac{1}{N} \sum_{i=1}^{N} \beta(s_i)$$

for selecting subsequences as it produces consistently better results.

For notational convenience $\gamma(U_s)$, $\overline{\beta}(U_s)$, and $P(\alpha(s) \leq 0)$ are abbreviated to $\gamma(s)$, $\overline{\beta}(s)$, and $P(s \leq 0)$ in the following sections.

2.6 The EvalHH hyper-heuristic

The EvalHH selection hyper-heuristic is used to evaluate subsequences and is shown in listing (2). It is a sequence based hyper-heuristic, and it employs a fixed set of k heuristic subsequences of varying lengths.

Algorithm 2 The EvalHH hyper-heuristic in pseudocode.

```
1.  ITERATIONS ← 150
2.  THRESHOLD ← 1.05
3.  new-sol  ← initialiseSolution()
4.  new-obj  ← $f$(new-sol)
5.  cross-sol ← initialiseSolution()
6.  cross-obj ← $f$(cross-sol)
7.  while (ITERATIONS > 0) do
8.      cur-sol ← new-sol
9.      cur-obj ← new-obj
10.     Subsequence ss ← selectSubsequence()
11.     for ($i$ ← 0 to length(ss)) do
12.         Heuristic h ← ss[i]
13.         new-sol ← apply(h, new-sol, cross-sol)
14.         ITERATIONS ← ITERATIONS − 1
15.     end
16.     new-obj ← $f$(new-sol)
17.     if (new-obj < cross-obj * THRESHOLD) then
18.         cross-sol  ← new-sol
19.         cross-obj ← new-obj
20.     end
21.     if (new-obj < cur-obj) then
22.         THRESHOLD ← 1.05
23.     else if (new-obj < cur-obj * THRESHOLD) then
24.         THRESHOLD ← THRESHOLD − 0.01
25.         if (THRESHOLD < 1) then
26.             THRESHOLD ← 1
27.         end
28.     else /∗ reject the new solution ∗/
29.         new-sol ← cur-sol
30.         new-obj ← cur-obj
31.     end
32. end
```

The function *selectSubsequence()* (line 10) selects a subsequence at random from this set. The function *apply()* (line 13) is then called for each heuristic class in the subsequence in order to choose, again at random, a low level heuristic and its parameters from the available heuristics of that class. This low level heuristic is then applied to the current solution, and if the class is C, to the current crossover solution. At the end of a subsequence, an objective function evaluation (line 16) and an acceptance check are performed (lines 17–31). If a new solution's objective value is less than the current solution's objective value or the current solution's objective value multiplied by THRESHOLD, then it is accepted (lines 17–27). Otherwise the new solution is rejected (lines 28–31). The threshold allows solutions with a small increase in objective function value (up to 5%) to be accepted. A low threshold forces the EvalHH hyper-heuristic to optimise the problem instead of exploring the space of low level heuristic selections.

The acceptance mechanism employed here is adapted from SSHH. The crossover mechanism used in the SSEval and SSHH hyper-heuristics is similar to the mechanism used by DBGen and operates on a population of one solution. As a result, any bias due to the crossover mechanism should be similar for all three hyper-heuristics.

The EvalHH hyper-heuristic is employed for evaluation purposes in order to ensure that any observed differences in performance are not dependant on the structure of DBGen which generated the subsequences. It should be emphasised that the EvalHH hyper-heuristic is not an attempt to produce a superior or novel hyper-heuristic algorithm. Rather EvalHH is intended to serve as a test bed and a "level playing field", in order to evaluate the performance of a number of subsequence sets which are selected using the γ-ratio.

2.7 Measuring hyper-heuristic performance

In this study, the performance of a hyper-heuristic is measured against 5 criteria:

1. the overall change in objective function value,
2. the number of heuristic selections required to find the best solution $x_{\min}$,
3. the number of objective function evaluations required to find $x_{\min}$,
4. the time required to find the best solution $x_{\min}$, and
5. the overall run time.

The change in objective function is measured by the *final log return* α_f. The final log return of a hyper-heuristic run or sequence s is the log return between the initial solution of the sequence x_0 and the best final solution $x_{\min}$ found during the run, which has objective value $o_{\min}$. In symbols

$$\alpha_f(s) = \log_{10}\left(\frac{o_{\min}}{o_0}\right).$$

The *mean final log return* of a set of N sequences is

$$\overline{\alpha}_f(\{s_1, \ldots, s_N\}) = \frac{1}{N}\sum_{i=1}^{N} \alpha_f(s_i).$$

The function $\overline{\alpha}_f$ is the mean of log values. In general, the mean of the logs is not equal to the log of the mean.[5] The anti-log of the mean of the logs is equivalent to the *geometric mean*. In symbols

[5] In fact, it is the median of logs that is equal to the log of the median.

$$\log^{-1}\left(\frac{1}{N}\sum_{i=1}^{N}\log(x_i)\right) = \sqrt[N]{x_1 \cdot x_2 \dots x_N}$$

assuming the values x_i all have the same sign. The geometric mean is used so that no range dominates the average. Although the use of sequences of log returns normalises the ranges of different objective functions, the log return values can still differ significantly, as some problems are harder to optimise than others. For this reason, in this study, $\overline{\alpha}_f$ is used in preference to the arithmetic mean of the decimal returns, although this value is also quoted (as a percentage) for comparison purposes.

As has been noted above, the number of objective function evaluations are important because, for certain problem sizes and problem domains, evaluating the objective function is computationally expensive. In addition, each objective function evaluation provides the hyper-heuristic with information regarding the progress of the optimisation process. More objective function evaluations provide more information which often leads to improved performance.

The number of selections and the time required to find the best solution are correlated. However, as the low level heuristics have different time complexities, a larger number of selections does not automatically imply a longer time to best solution.

The overall run time provides a simple measure of a hyper-heuristic's time complexity over an entire run. The overall run time combines low level heuristic execution time, objective function evaluations time, and the time necessary to perform other computations such as selection, acceptance or online learning.

3 Experimental setup

The γ-ratio is used to select sets of effective and disruptive subsequences of heuristics from the offline database. These subsequence sets are used to parameterise the EvalHH hyper-heuristic which is executed on unseen HyFlex problems in order to evaluate the subsequences. The EvalHH results are then compared with results produced by the SSHH hyper-heuristic.

The experiments presented here are designed to explore the effect of calculating γ for low level heuristics, and then for heuristic classes at a domain and cross-domain level. For clarity, the experiments are labeled according to the following convention. The results of selecting subsequences using a γ calculated over the low level heuristics of a particular domain have a suffix -DH where the D denotes domain statistics and the H denotes low level heuristics. When γ is calculated for the heuristic classes of a particular domain the suffix is -DC. Finally, when γ is calculated across all domains in the database the suffix is always -GC as it is not possible to to calculate cross domain statistics for individual low level heuristics, only classes.

In Sect. 4.1 an effective subsequence set of low level heuristics is selected using a γ-ratio that is calculated over each domain. In Sect. 4.2 this experiment is repeated for subsequences of heuristic classes. The intention is to assess the scope and limitations of the proposed statistical framework, and demonstrate empirically its predictive capability.

A primary objective of this study is to determine the existence of effective cross-domain subsequences of heuristic classes. With this in mind, in Sect. 4.3, an effective and a disruptive subsequence set of heuristic classes are selected using a γ-ratio that is calculated over the whole database. Section 4.4 presents a detailed analysis of these results for the Bin Packing domain.

In each case, the EvalHH hyper-heuristic results are compared with the results produced by the SSHH hyper-heuristic. The SSHH hyper-heuristic has been tested on the HyFlex problems (taken from the CHeSC 2011 competition) and compared with a number of other hyper-heuristics. The published results demonstrate that SSHH is able to outperform the then best-in-class hyper-heuristic AdapHH (see Mısır et al. 2013) on these problems.

These experiments employ short subsequences of length 2 and 3. There are three reasons for this. Firstly, it is logically necessary to demonstrate that the proposed statistical framework works with short subsequences before considering subsequences of longer lengths. Secondly, short subsequences occur much more frequently than longer subsequences in the offline learning database. As a result, the statistics calculated within and across problem domains are more reliable. Thirdly, as some work has already been carried out with heuristic pairs (see for example Mısır et al. 2012), it was decided to include subsequences of length 3. In Sect. 4.5 the issue of subsequence length is examined by selecting and evaluating an effective cross-domain subsequence set, where the subsequences have unrestricted length.

All of the preceding experiments employ a run length of 150 heuristic selections, and for some problems this can be considered to be a small number over which to evaluate a hyper-heuristic. The final experiment addresses the issue of run length.

The abbreviations used throughout the results sections together with their descriptions are summarised in Table 2.

The experiments were conducted on a Mac Pro computer with a 3.5 GHz, 6 core, Intel Xeon E5 processor, and 16 GB of 1866 MHz memory. Each hyper-heuristic

Table 2 Experimental abbreviations and descriptions

Abbreviation	Description
DBGen	Hyper-heuristic used to generate the learning database
SSHH	Sequenced based hyper-heuristic used for comparisons
EvalHH	Hyper-heuristic used to evaluate subsequence sets
TOP-DH	Subsequences of low level heuristics with the largest γ (by domain)
TOP-DC	Subsequences of heuristic classes with the largest γ (by domain)
TOP-GC	Subsequences of heuristic classes with the largest γ (all domains)
BOT-GC	Subsequences of heuristic classes with the smallest γ (all domains)
SINGLE	The individual heuristic classes {L, C, R, M}
RAND	Randomly generated subsequences of heuristic classes
LONG-GC	Arbitrary length subsequences of heuristic classes
{L}	Singleton heuristic class
{LL}	Singleton heuristic class subsequence

evaluation takes approximately 4 h running on a restricted number of threads. This is to ensure that the processor always has spare capacity so that the recorded timings are reliable.

4 Results

In this section, the experimental results of this paper are presented.

4.1 Subsequences of low level heuristics

Each problem domain has its own set of low level heuristics (see Table 1). In this section, the γ-ratio is used to select subsequences of low level heuristics.

For each problem instance in a domain, the top 10 subsequences with the largest γ-ratio are selected using the query

$$\textit{select}\ \ s\ \textit{from subseq where length} \leq 3\ \textit{order by}\ \gamma\ \textit{-ratio descend limit}\ 10$$

where γ is calculated using a *leave-one-out* cross-validation methodology. Recall that there are 10 problem instances in each of the four HyFlex domains. For each target problem in a domain, γ is calculated from the subsequences of low level heuristics from the remaining nine problems. The subsequences are then evaluated on the target problem. This ensures that the subsequences are always evaluated on a problem that is "unseen". This methodology gives rise to 40 subsequence sets, one for each problem in each domain. As the γ statistics are quite stable, the selected subsequences for the problems in a domain are very similar, usually differing by at most one or two subsequences. This subsequence set is denoted TOP-DH. The subsequence sets and their γ-ratios (calculated over all the problem instances) for each domain are shown in Table 3.

Table 3 The TOP-DH subsequences of low level heuristics for each domain

BP	$\gamma(s)$	PFS	$\gamma(s)$	SAT	$\gamma(s)$	PS	$\gamma(s)$
R2R2	43.8493	M4L7	4.0900	M1M0	16.2481	L4L4	295.7140
M3R2	28.9555	M4L8	3.6249	M1M1	16.1602	L4L3	271.2990
R2L4	28.6750	M4L10	3.2020	L7M1	16.0079	L4L2	269.6110
L4R2	26.1469	R6R6	3.0929	M0M1	15.7271	L3L4	262.8780
R1R2	24.3380	M4L9	3.0856	L7M0	15.4519	L2L4	225.9580
R2L6	19.7258	R5M4	3.0016	M0M0	15.1376	L3L3	224.4890
C7R2	19.6726	M4R6	2.9105	M0L7	14.9501	L1L4	214.2970
R2C7	19.5043	R6M4	2.8195	L7L7	14.9351	L0L2	205.0780
R2M3	19.3604	R5R5	2.6015	M1L7	14.3604	L2L3	201.9660
L6R2	18.4136	R5R6	2.5843	L8M0	11.3984	L0L3	200.1810

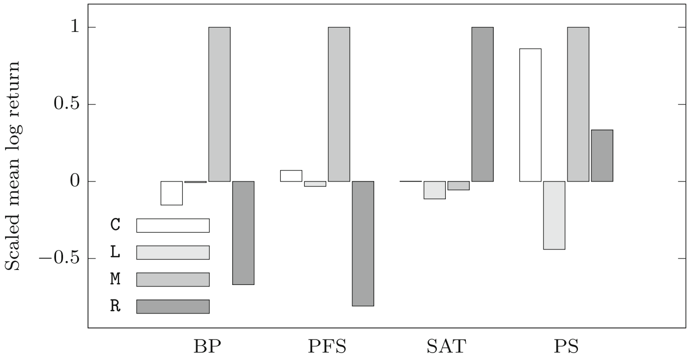


Fig. 1 The scaled mean log returns $\bar{\alpha}$ of the low level heuristics C, L, M, and R for each domain. In each domain the $\bar{\alpha}$ values have been scaled by the largest absolute $\bar{\alpha}$ value into the interval $[-1, 1]$

Table 4 The mean unit log return $\overline{\beta}(s)$, the negative $\beta^{-}(s)$ and positive $\beta^{+}(s)$ component sums, the probability of a negative log return $P(s \leq 0)$, the γ-ratio, and the number N_s of occurrences of the subsequence set TOP-DH of low level heuristics for each domain

Dom.	$\overline{\beta}(s)$	$\beta^{-}(s)$	$\beta^{+}(s)$	$P(s \leq 0)$	γ-ratio	N_s
BP	– 0.0283	257.4574	0.3608	0.9876	189.1929	9098
PFS	– 0.0109	33.4832	0.8420	0.8379	18.1776	2986
SAT	– 0.0238	150.3768	0.0000	1.0000	150.3768	6314
PS	– 0.4848	2371.4710	0.0000	1.0000	2371.4710	4892

As might be expected, the best performing subsequences differ markedly by domain. The PS subsequences consist exclusively of local search heuristics whereas the SAT subsequences combine mutation and local search. The PFS and BP subsequences both utilise ruin and recreate, mutation, and local search. The underlined subsequences in Table 3 violate the principle of exploration followed by exploitation. Interestingly, this principle does not appear to be preserved for the BP or PS domains. In the BP domain, local search is followed by the ruin and recreate heuristic R2. The BP ruin and recreate heuristics are *destroy x highest bins* and *destroy x lowest bins*. These heuristics remove all the pieces from the x highest or lowest filled bins where x is an integer determined by the “intensity of mutation” parameter. They then repack the pieces using the *best-fit* heuristic. For low values of x these heuristics preserve most of the existing solution and, as the number of bins increases, their effect becomes more exploitative than exploratory. In fact, on the BP domain, the ruin and recreate heuristics produce, on average, a larger reduction in objective function value than the local search heuristics (see Fig. 1).

The overall γ-ratio of each subsequence set shown in Table 3 is shown in Table 4 for each domain. As the γ-ratios are greater than 1 each set has a negative mean unit log return $\overline{\beta} \leq 0$. These values suggest that these subsequences will be effective on the HyFlex problems.

Table 5 shows the results of the EvalHH hyper-heuristic using the subsequence set TOP-DH (with a leave-one-out methodology) and the SSHH hyper-heuristic, averaged

Table 5 The mean final log return $\bar{\alpha}_f$ for the hyper-heuristic EvalHH using the subsequence sets TOP-DH, and the hyper-heuristic SSHH

Dom.	TOP-DH	SSHH
BP	– **0.3565**	– 0.3009
PFS	– **0.0074**	– 0.0049
SAT	– **0.9509**	– 0.6908
PS	– 1.7708	– **1.7770**
All	– **0.7714**	– 0.6934

The domain statistics are calculated over 400 runs
Winning scores are shown in bold

over 40 runs of each HyFlex problem; a total of 1600 runs. The low level heuristic parameters are chosen at random.

Each run of 150 selections is seeded by a single unique number

$$seed = 401 + (40\,p) + r$$

where $p = 0, \ldots, 39$ is the problem index and $r = 0, \ldots, 39$ is the run index. These seeds are distinct to the seeds used to generate the offline database.

The hyper-heuristic EvalHH using the subsequence set TOP-DH outperforms the SSHH hyper-heuristic overall, and on the BP, PFS and SAT domains (see Table 5).

In the evolutionary computation literature statistical tests are widely used to compare and rank the performance of algorithms (see for example Derrac et al. 2011). In this paper, the non-parametric one tailed Wilcoxon signed-rank test is used to validate the proposed methodology by establishing *stochastic orderings* on two hyper-heuristics A and B. The null hypotheses of the Wilcoxon test is that the median difference between pairs of observations is zero, and this is tested at a significance level of 0.01 on sample sizes of 400 or 1600, The null hypothesis is rejected if the p-value is less than 0.01. In this case, the alternative hypothesis that the median difference between pairs of observations is less than zero is accepted with 99% confidence. This implies that the random variable $\alpha_f(\text{A})$ is "smaller" than the random variable $\alpha_f(\text{B})$, and thus hyper-heuristic A is more effective than hyper-heuristic B. The $\alpha_f(\text{A})$ and $\alpha_f(\text{B})$ values can be paired because the initial seed and therefore the initial solution for each problem $p = 0, \ldots, 39$ and run $r = 0, \ldots, 39$ is the same for both hyper-heuristics. The results of the Wilcoxon test are shown in Table 6.

For each domain, the null hypothesis is rejected as the p-value is less than 0.01 (shown in bold) and the alternative hypothesis $\alpha_f(\text{TOP-DH}) < \alpha_f(\text{SSHH})$ is accepted with 99% confidence. Overall the median difference is significant and the hyper-heuristics can be stochastically ordered so that

$$\alpha_f(\text{TOP-DH}) < \alpha_f(\text{SSHH})$$

with 99% confidence.

The result that a simple hyper-heuristic such as EvalHH using fixed sets of subsequences (specifically the TOP-DH subsequences) is able to outperform SSHH, a published hyper-heuristic which employs online learning, demonstrates the utility of

Table 6 The Wilcoxon test results for α_f(TOP-DH) and α_f(SSHH)

Dom.	$\widehat{d}$	$\bar{d}$	SD	SEM	p-value	Conf. Int.
BP	– 0.0405	– 0.0556	0.1231	0.0062	0.0000	$[-\infty, -0.0253]$
PFS	– 0.0024	– 0.0026	0.0021	0.0001	0.0000	$[-\infty, -0.0022]$
SAT	– 0.2584	– 0.2601	0.1032	0.0052	0.0000	$[-\infty, -0.2457]$
PS	– 0.0013	0.0063	0.1848	0.0092	0.0032	$[-\infty, -0.0013]$
All	– 0.0685	– 0.0780	0.1631	0.0041	0.0000	$[-\infty, -0.0555]$

The sample median difference $\widehat{d}$, the sample mean difference $\bar{d}$, the standard deviation SD, the standard error of the mean SEM, the p-value, and the interval within which the population median difference falls with 99% confidence

Table 7 The TOP-DC subsequences of heuristics classes ordered by descending γ-ratio from left to right

Dom.	Subsequences
BP	{RR, RL, LR, RRR, CR, RLR, RC, RRL, LRR, RLL}
PFS	{RR, RL, LR, CR, RC, MR, RRL, MRR, RLR, CRR}
SAT	{LL, ML, LM, MML, MLM, LMM, MMM, LML, MLL, LLM}
PS	{LL, LLL, RLL, LRL, RL, RRL, CLL, MLL, LLR, RLR}

the subsequence based approach and the statistical framework proposed in this study. Of course, the online method must learn which heuristics to select during execution, which is not required of the offline method, but it shows that there is scope to improve the performance of the SSHH hyper-heuristic using offline learning techniques.

4.2 Subsequences of heuristic classes

The number and implementation of low level heuristics varies between domains. As a result, γ-ratios for subsequences of low level heuristics can only be calculated for a particular domain. For the case where there are several domains under consideration the γ-ratio must be calculated for heuristic classes. Before examining the cross-domain case, in this section, the γ-ratio is used to select effective subsequences of heuristic classes within each domain.

For each domain, the top 10 subsequences of length 2 and 3 with the largest γ-ratio are selected using a γ that is calculated over the subsequences of heuristic classes of that domain, using the leave-one-out methodology described in the previous section.

The subsequence sets TOP-DC, calculated over *all* the problem instances for each domain, are shown in Table 7.

Notice that some of the subsequences of heuristic classes shown in Table 7 can also be observed in the subsequences of low level heuristics shown in Table 3. For example, the most effective subsequence of low level heuristics in the BP domain is R2R2 while the most effective subsequence of heuristic classes in the BP domain is RR. The γ-ratio of each subsequence set shown in Table 7 is shown in Table 8. As the γ-ratios are greater than 1 each set has a negative mean unit log return $\overline{\beta} \leq 0$.

Table 8 The mean unit log return $\overline{\beta}(s)$, the negative $\beta^{-}(s)$ and positive $\beta^{+}(s)$ component sums, the probability of a negative log return $P(s \leq 0)$, the γ-ratio, and the number N_s of occurrences of the subsequence set TOP-DC of heuristics classes for each domain

Dom.	$\overline{\beta}(s)$	$\beta^{-}(s)$	$\beta^{+}(s)$	$P(s \leq 0)$	γ-ratio	N_s
BP	– 0.0241	452.5768	6.3965	0.8915	61.1881	18,495
PFS	– 0.0057	72.7367	9.4039	0.7724	6.9913	11,076
SAT	– 0.0140	556.9779	12.6391	0.9580	40.8369	38,922
PS	– 0.2766	9897.0400	927.9030	0.8820	10.6545	32,429

Table 9 The mean final log return $\overline{\alpha}_f$ for the hyper-heuristic EvalHH using the subsequence set TOP-DC, and the hyper-heuristic SSHH

Dom.	TOP-DC	SSHH
BP	– 0.2896	– **0.3009**
PFS	– **0.0053**	– 0.0049
SAT	– **0.8275**	– 0.6908
PS	– **1.8378**	– 1.7770
All	– **0.7109**	– 0.6934

The domain statistics are calculated over 400 runs
Winning scores are shown in bold

Each set of subsequences is evaluated using the EvalHH hyper-heuristic on the HyFlex problems of that domain. The low level heuristics and heuristic parameters are chosen at random. Table 9 shows the results of the EvalHH hyper-heuristics using the subsequence set TOP-DC, and the SSHH hyper-heuristic, over 40 runs of 150 selections for each HyFlex problem.

The subsequence set TOP-DH outperforms the subsequence set TOP-DC overall, and on the BP, PFS and SAT domains (see Tables 5, 9). This is unsurprising as information regarding specific low level heuristics is lost when using heuristic classes. However, on the PS domain the TOP-DC subsequences outperform the TOP-DH subsequences. One reason for this is that the TOP-DH subsequences for PS contain only L class heuristics while the TOP-DC subsequences for PS contain several heuristic types. A lack of heuristic diversity, that is, the lack of other heuristic classes in a set, can impair performance as noted in Fisher and Thompson (1963) and Crowston et al. (1963).

The hyper-heuristic EvalHH using the subsequence set TOP-DC outperforms the SSHH hyper-heuristic overall, and on the PFS, SAT and PS domains (see Table 9). The Wilcoxon test is used to establish whether the median differences are statistically significant. The results are shown in Table 10.

For the PFS, SAT, and PS domains the null hypothesis is rejected as the p-value is less than 0.01 (shown in bold) and the alternative hypothesis α_f(TOP-DC) $<$ α_f(SSHH) is accepted with 99% confidence. For the BP domain the α_f(TOP-DC) and α_f(SSHH) hyper-heuristics are not stochastically comparable. Overall the median difference is significant and the hyper-heuristics can be ordered stochastically so that

$$\alpha_f(\text{TOP-DC}) < \alpha_f(\text{SSHH})$$

Table 10 The Wilcoxon test results for α_f(TOP-DC) and α_f(SSHH)

Dom.	$\widehat{d}$	$\bar{d}$	SD	SEM	p-value	Conf. int
BP	0.0097	0.0113	0.0027	0.0054	0.9845	$[-\infty,\ 0.0223]$
PFS	– 0.0004	– 0.0004	0.0020	0.0001	0.0000	$[-\infty, -0.0002]$
SAT	– 0.1384	– 0.1367	0.0968	0.0048	0.0000	$[-\infty, -0.1273]$
PS	– 0.0413	– 0.0608	0.1231	0.0062	0.0000	$[-\infty, -0.0318]$
All	– 0.0357	– 0.0467	0.0003	0.0028	0.0000	$[-\infty, -0.0291]$

The sample median difference $\widehat{d}$, the sample mean difference $\bar{d}$, the standard deviation SD, the standard error of the mean SEM, the p-value, and the interval within which the population median difference falls with 99% confidence

with 99% confidence.

This result is notable because even though information regarding specific heuristics is lost when using subsequences of heuristic classes, the offline methodology is still superior to SSHH which performs online learning on low level heuristics and parameters.

4.3 Cross-domain subsequences of heuristic classes

A primary objective of this study is to explore the potential of subsequences to be effective across problem domains thus demonstrating cross-domain generalisation. In this section four sets of heuristic classes are evaluated:

1. the top 10 subsequences of length 2 and 3 with the largest γ-ratios denoted TOP-GC,
2. the bottom 10 subsequences of length 2 and 3 with the smallest γ-ratios denoted BOT-GC,
3. the set RAND consisting of subsequences of length 2 and 3 that are randomly generated at each iteration of the optimisation process, and
4. the set SINGLE containing the individual heuristic classes $\{\texttt{L}, \texttt{C}, \texttt{R}, \texttt{M}\}$.

The RAND and SINGLE subsequence sets are included to provide results for direct comparison with TOP-GC and BOT-GC. The random subsequences RAND act as a control variable in order to assess the relationship between the TOP-GC and BOT-GC subsequences. The individual heuristic set SINGLE causes the EvalHH hyper-heuristic to behave as a single selection hyper-heuristic, and allows for a comparison between single selection and sequenced based methods. The results for the DBGen and SSHH hyper-heuristics are also included to provide baseline comparisons.

The effective set of subsequences TOP-GC contains the 10 subsequences

$$\{\texttt{LL}, \texttt{LLL}, \texttt{RLL}, \texttt{LRL}, \texttt{RL}, \texttt{RRL}, \texttt{MLL}, \texttt{CLL}, \texttt{LLR}, \texttt{MMM}\}.$$

As the γ-ratio is greater than 1 the subsequence set TOP-GC has a negative mean unit log return $\overline{\beta} \leq 0$ (see Table 11). Notice that all but two of the subsequences ends in an L. This is to be expected for two reasons. Firstly, this result supports discussions earlier

Table 11 The mean unit log return $\overline{\beta}(s)$, the negative $\beta^{-}(s)$ and positive $\beta^{+}(s)$ component sums, the probability of a negative log return $P(s \leq 0)$, the γ-ratio, and the number N_s of occurrences for the TOP-GC and BOT-GC subsequences

s	$\overline{\beta}(s)$	$\beta^{-}(s)$	$\beta^{+}(s)$	$P(s \leq 0)$	γ-ratio	N_s
TOP-GC	– 0.1322	10087.5530	948.8757	0.8698	10.6199	69,119
BOT-GC	0.1411	301.8259	3386.0429	0.6085	0.0891	21,860

in this paper and in the literature that exploration followed by the exploitation of a local search operation is the preferable ordering of these heuristics (see Kheiri and Keedwell 2015). Secondly, the fact that the local search heuristic is only able to improve the log return of a sequence is important. If the solution is already optimal with regard to the local search landscape, the local search will return the initial solution. Therefore, by ending with local search, progress made by the subsequence can be exploited with no potential for generating a worse solution.

In contrast, the disruptive set of subsequences BOT-GC contains the 10 subsequences

$$\{\,\texttt{CC},\,\texttt{CRC},\,\texttt{RCC},\,\texttt{CCC},\,\texttt{CCR},\,\texttt{RC},\,\texttt{RCM},\,\texttt{CRM},\,\texttt{CR},\,\texttt{CMC}\,\}$$

with the smallest γ-ratios in the database. As the γ-ratio is less than 1 the subsequence set has a positive mean unit log return $\overline{\beta} > 0$ (see Table 11). Notice that no subsequence contains an `L` class heuristic. The BOT-GC subsequences contain a large number of crossover operations and again confirms what might be expected in that crossover operations are likely to be disruptive to existing good solutions (particularly when executed repeatedly) and are unlikely to deliver significant performance improvements. These operations are frequently combined with the ruin and recreate operation which would make for a highly disruptive pairing that would eliminate any information gained from a search of the local space.

The Table 12 shows the results of the DBGen and SSHH hyper-heuristics and the EvalHH hyper-heuristic using the subsequence sets TOP-GC, SINGLE, RAND, and BOT-GC, over 40 runs of 150 selections on each of the HyFlex problems. During evaluation the low level heuristics and their parameters are chosen at random.

The Wilcoxon test is used to establish whether the median differences observed are statistically significant. The results are shown in Table 13.

For each pair of hyper-heuristics the null hypothesis is rejected as the p-value is less than 0.01 and the alternative hypothesis $\alpha_f(\text{A}) < \alpha_f(\text{B})$ is accepted with 99% confidence. Thus the median differences are significant and the hyper-heuristics can be stochastically ordered so that

$$\alpha_f(\text{TOP-GC}) < \alpha_f(\text{SINGLE}) < \alpha_f(\text{RAND}) < \alpha_f(\text{BOT-GC})$$

with 99% confidence (for each comparison).

Table 12 and this statistical comparison illustrates the differences in the various selection mechanisms. The SSHH hyper-heuristic provides the best performance here,

Table 12 The mean final log return $\bar{\alpha}_f$, the mean percent return, the mean number of selections to a minimum, the mean number of objective function evaluations, the mean time to a minimum (ms), and the mean run time (ms), for the hyper-heuristics DBGen and SSHH, and the hyper-heuristic EvalHH using the subsequence sets TOP-GC, SINGLE, RAND and BOT-GC

	$\overline{\alpha}_f$	Percent %	Min. Sel.	Obj. Eval.	Min. T	Total T
DBGen	– 0.6243	– 49.0924	92.8738	93.8513	24,873	43,998
SSHH	– 0.6934	– 55.4024	111.8219	55.7438	36,824	51,643
TOP-GC	– 0.6868	– 53.0720	110.5556	39.1931	40,241	58,633
SINGLE	– 0.6643	– 52.9842	107.0994	107.0994	18,608	30,507
RAND	– 0.6328	– 51.5645	108.1688	40.3231	20,781	30,931
BOT-GC	– 0.1781	– 26.4057	86.6506	31.8488	5862	12,404

The statistics are calculated over 1600 runs

Table 13 The hyper-heuristic pair, the sample median difference $\widehat{d}$, the sample mean difference $\bar{d}$, the standard deviation SD, the standard error of the mean SEM, the p-value, and the interval within which the population median difference falls with 99% confidence

Hyper-heuristics	$\widehat{d}$	$\bar{d}$	SD	SEM	p-value	Conf. Int.
TOP-GC-SINGLE	– 0.0075	– 0.0226	0.1166	0.0029	0.0000	[∞, – 0.0040]
SINGLE-RAND	– 0.0173	– 0.0385	0.1439	0.0036	0.0000	[∞, – 0.0111]
RAND-BOT-GC	– 0.2714	– 0.4547	0.7050	0.0176	0.0000	[∞, – 0.2387]
TOP-GC-RAND	– 0.0393	– 0.0540	0.1252	0.0031	0.0000	[∞, – 0.0307]
TOP-GC-BOT-GC	– 0.3300	– 0.5087	0.7433	0.0186	0.0000	[∞, – 0.3030]

which is understandable given that it is the only online learning technique and so is able to adapt itself to the different requirements of each of the problem domains. Interestingly, the TOP-GC subsequences are the next best performing approach and are better than both SINGLE and RAND. This is notable because it demonstrates the increased performance available from using well-chosen subsequences over single heuristic selections or random subsequences. Predictably, the BOT-GC subsequences perform badly on this set of test runs.

The mean number of heuristic selections to find a minimum are similar for SSHH, SINGLE, TOP-GC, and RAND (see Table 12). However when the mean number of objective evaluations to a minimum is considered, TOP-GC and RAND use less evaluations than SSHH and SINGLE. In particular, when EvalHH is parameterised with the set SINGLE it operates as a single selection hyper-heuristic, and each selection is followed by an objective function evaluation. As a result the mean number of selections is equal to mean number of objective evaluations. In this case, the extra objective function evaluations performed when using SINGLE give rise to a superior performance to RAND. This is because a single selection hyper-heuristic has more opportunities to accept solutions with low(er) objective function values than a sequence based one.

The TOP-GC subsequence set is the most time expensive. This is due to the large proportion of `L` class heuristics in the TOP-GC subsequences, as `L` heuristics have a higher time cost than `C`, `R` or `M` heuristics (see Table 14). The SINGLE and RAND sub-

Table 14 The mean execution time (in milliseconds) of the heuristics classes C, L, M, and R calculated from the sequences generated by DBGen on the HyFlex problems overall and for each domain

LLH	All (ms)	BP (ms)	PFS (ms)	SAT (ms)	PS (ms)
C	27.5080	169.1298	0.0408	0.1773	7.6091
L	832.7797	8.2271	41.7763	1.2088	2202.5602
M	8.5262	2.9007	3.4166	16.6931	1.4122
R	256.2476	3.2653	466.0749	0.3983	489.5818

Table 15 A domain by domain comparison of the mean final log return $\bar{\alpha}_f$ of BOT-GC and TOP-GC

Dom.	BOT-GC	TOP-GC
BP	− **0.3079**	− 0.2419
PFS	− 0.0040	− **0.0051**
SAT	− 0.1246	− **0.6567**
PS	− 0.2760	− **1.8437**
All	− 0.1781	− **0.6868**

The domain statistics are calculated over 400 runs
Winning scores are shown in bold

sequences generate a lower number of L heuristic selections and thus have a lower time cost. Unsurprisingly, BOT-GC which contains no L heuristics, and many C heuristics is the most time efficient.

4.4 An analysis of the bin packing problem

The TOP-GC subsequences outperform the BOT-GC subsequences when compared over runs, problems, and most significantly, domains. Specifically TOP-GC "wins" 1119 runs out of 1600, 34 problems out of 40, and 3 domains out of 4. Table 15 show the mean final log returns $\overline{\alpha}_f$ for the subsequence sets TOP-GC and BOT-GC broken down by problem domain. Notice that the BOT-GC subsequences outperform the TOP-GC subsequences on the Bin Packing problem. In fact, the 6 problems that BOT-GC "wins" against TOP-GC are all examples of the Bin Packing problem.

The statistics for the TOP-GC and BOT-GC subsequences are shown in Table 11. These values are calculated from subsequences that have been drawn from all four problem domains. The γ-ratios and the mean unit log returns $\overline{\beta}$ suggest these sets should produce a good and poor performance respectively when evaluated on the HyFlex problems. However on the BP domain, not only has BOT-GC outperformed TOP-GC but it has produced results comparable to SSHH. In order to explain the discrepancy, the γ and $\overline{\beta}$ values are recalculated from subsequences drawn only from the BP domain. The results are shown in Table 16.

Notice that the BOT-GC subsequence set now has a γ-ratio greater than 1 and a negative mean unit log return $\overline{\beta} \leq 0$. Although the TOP-GC subsequences still has a $\gamma > 1$ and $\overline{\beta} \leq 0$, when compared to BOT-GC they have smaller magnitudes.

Table 16 The mean unit log return $\overline{\beta}(s)$, the negative $\beta^{-}(s)$ and positive $\beta^{+}(s)$ component sums, the probability of a negative log return $P(s \leq 0)$, the γ-ratio, and the number N_s of occurrences for the TOP-GC, BOT-GC, and TOP-DC subsequences on the Bin Packing domain

s	$\overline{\beta}(s)$	$\beta^{-}(s)$	$\beta^{+}(s)$	$P(s \leq 0)$	γ-ratio	N_s
TOP-GC	– 0.0073	155.5739	49.5213	0.8440	3.0794	14,477
BOT-GC	– 0.0162	81.3211	15.8588	0.8085	4.8236	4037
TOP-DC	– 0.0241	452.5768	6.3965	0.8915	61.1881	18495

These quantities provide an explanation as to why the BOT-GC subsequences perform better than the TOP-GC subsequences on the BP domain, and they imply that statistics calculated at the domain level are more reliable than those calculated across different domains. In order to understand why domain statistics produce better results than cross-domain statistics consider Fig. 1 which shows the scaled mean log returns $\bar{\alpha}$ for the low level heuristic classes C, L, M, and R calculated over the 400 sequences of each domain.

The results indicate that the effectiveness of the low level heuristics varies by domain, as one might expect but that the relationship between heuristics and the BP domain is diametrically opposed to their behaviour in other domains. For example, in the BP and PFS domains the L heuristic is less effective than the R heuristic. In the SAT and PS domains the situation is reversed. This suggests an explanation as to why the BOT-GC mean unit log returns differ so markedly between the BP domain and the means calculated over all four domains. In the BP problem domain, the L heuristic is less effective than the C and R heuristics. As BOT-GC employs more C and R heuristics and less L heuristics than TOP-GC, the performance of the BOT-GC subsequences is superior.

The analysis in this section has shown that the interface between heuristics and problem domain are complex. As the effectiveness of the heuristic classes varies across the problem domains, what has been learned about a class on one domain cannot necessarily be transferred to another. However, it should also be noted that three of the four domains behaved similarly and so good subsequence selections on one of these would transfer to the other three domains. An important finding is that this type of statistical analysis can quantify this difference in domains and bespoke optimisers can be created to cope with the unique demands of an outlier domain such as Bin Packing.

4.5 Subsequence length

This study has concentrated on short subsequences of length 2 and 3. In this section the performance of a set of subsequences of arbitrary length chosen using the γ-ratio is examined.

The 10 subsequences with the largest γ-ratio are selected using a γ that is calculated over the subsequences of whole database. This subsequence set is denoted LONG-GC and is shown in Table 17. It should be noted that the database contains subsequences

Table 17 The mean unit log return $\overline{\beta}(s)$, the negative $\beta^{-}(s)$ and positive $\beta^{+}(s)$ component sums, the probability of a negative log return $P(s \leq 0)$, the γ-ratio, and the number N_s of occurrences for the LONG-GC subsequences

s	$\overline{\beta}(s)$	$\beta^{-}(s)$	$\beta^{+}(s)$	$P(s \leq 0)$	γ-ratio	N_s
LL	– 0.2276	4075.5700	0.0000	1.0000	4075.5701	17,904
LLL	– 0.2241	1580.0700	0.0000	1.0000	1580.0699	7052
LLLL	– 0.2073	635.8680	0.0000	1.0000	635.8680	3068
LLLLL	– 0.1866	260.6290	0.0000	1.0000	260.6290	1397
LLLLLL	– 0.1685	114.2300	0.0000	1.0000	114.2300	678
RLLLL	– 0.1749	135.5010	1.5577	0.8982	52.9769	766
LLLLLLL	– 0.1482	49.9410	0.0000	1.0000	49.9410	337
LRLLL	– 0.1717	129.8070	2.4087	0.8895	38.0816	742
RLLL	– 0.1842	317.3610	7.4553	0.8864	37.5339	1682
LRLLLL	– 0.1593	56.1363	0.7107	0.9023	32.8153	348
LONG-GC	– 0.2161	7355.1133	12.1324	0.9887	560.0748	33,974

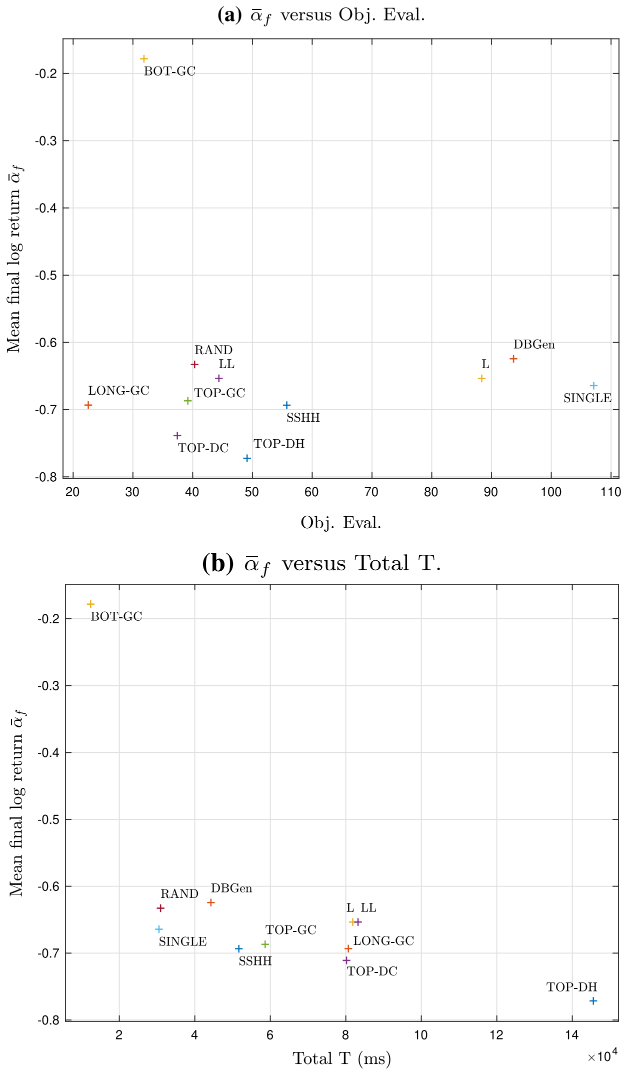
Table 18 The mean final log return $\bar{\alpha}_f$, the mean percentage change, the mean number of selections to a minimum, the mean number of objective function evaluations, the mean time to a minimum (ms), and the total run time (ms). The statistics are calculated over 1600 runs

	$\overline{\alpha}_f$	Percent %	Min. Sel.	Obj. Eval.	Min. T	Total T
DBGen	– 0.6243	– 49.0924	92.8738	93.8513	24,873	43,998
TOP-DH	– 0.7714	– 59.2435	97.5250	49.1031	62,632	141,847
TOP-DC	– 0.7109	– 47.2025	96.0275	37.4350	65,249	99,218
SSHH	– 0.6934	– 55.4024	111.8219	55.7438	36,824	51,643
LONG-GC	– 0.6932	– 52.3682	107.3506	22.5325	52,455	80,678
TOP-GC	– 0.6868	– 53.0720	110.5556	39.1931	40,241	58,633
SINGLE	– 0.6643	– 52.9842	107.0994	107.0994	18,608	30,507
{L}	– 0.6536	– 46.4370	88.3656	88.3656	37,807	81,847
{LL}	– 0.6535	– 46.4447	89.3356	44.3863	38,767	83,240
RAND	– 0.6328	– 51.5645	108.1688	40.3231	20,781	30,931
BOT-GC	– 0.1781	– 26.4057	86.6506	31.8488	5862	12,404

of up to length 25. Notice that the γ-ratios and the mean unit log returns $\overline{\beta}$ have greater magnitudes than those of the TOP-GC subsequence set (see Table 11).

The subsequence set LONG-GC is dominated by the L heuristic class, and the question arises as to how a hyper-heuristic using only this heuristic would perform. With this in mind, EvalHH is also run with the singleton subsequence sets {L} and {LL}. Table 18 contains the results for the hyper-heuristic SSHH and the the hyper-heuristic EvalHH parameterised with the subsequence sets of the preceding sections.

The LONG-GC subsequence set has a slightly better mean final log return than TOP-GC and uses the lowest number of objective function evaluations due to the length of its subsequences, but is one of the most computationally expensive sets



(a) $\overline{\alpha}_f$ versus Obj. Eval.

(b) $\overline{\alpha}_f$ versus Total T.

Fig. 2 The mean final log return $\overline{\alpha}_f$ plotted against **a** the mean number of objective function evaluations Obj. Eval. required to find a minimum, and **b** the mean overall run time Total T, for the HyFlex domains

due to the dominance of the L heuristic class. The most computationally expensive set of subsequences is TOP-DH. The overall timings for TOP-DH are dominated by the results on the PS domain, and these subsequences consist entirely of L class heuristics (see Table 3). However TOP-DH is more expensive than {L} and {LL} as its subsequences tend to be the most expensive in the other domains as well (see Table 14).

Table 18 shows that a well chosen set of subsequences of heuristics such as LONG-GC or TOP-GC outperform the single heuristic selections SINGLE, demonstrating

Table 19 A domain by domain comparison of the mean final log return $\overline{\alpha}_f$ of TOP-DH and BOT-DH

Dom.	TOP-DH	BOT-DH
BP	– **0.6515**	– 0.0290
PFS	– **0.0102**	– 0.0004
SAT	– **1.1283**	– 0.2459
PS	– **1.7873**	– 0.0485
All	– **0.8943**	– 0.0809

Winning scores are shown in bold

the benefits of a sequence based approach. Furthermore, combinations of several individual heuristics such as SINGLE outperform the single heuristics {L} and {LL} reproducing again the result observed in Fisher and Thompson (1963) and Crowston et al. (1963). Finally the single heuristics {L} and {LL} are superior to the random and badly chosen subsequences sets RAND and BOT-GC.

The results of plotting the mean final log returns $\overline{\alpha}_f$ against the number of objective function evaluations and the overall run time are shown in Fig. 2. They illustrate the efficiency trade-offs for the various hyper-heuristic runs.

4.6 Run length

The previous experiments all use a run length of 150 low level heuristic selections. For many problems, especially those with computationally efficient heuristics, 150 is a relatively small number over which to evaluate a hyper-heuristics's performance. Thus, the question arises as to whether the observed gains in performance are still present after a more realistic execution time. In this section, the TOP-DH subsequences, and a disruptive set of low level heuristic subsequences BOT-DH are evaluated 10 times on each of the 40 HyFlex problems for 10 min of wall clock time, using a leave-one-out methodology; a total of 400 runs. The results are shown in Table 19. The runs produce much larger numbers of selections, ranging from 941 selections for problem 13 in the PS domain, up to 6,577,523 selections for problem 8 in the BP domain.

The results show that the difference in performance has increased on each domain with time and the number of selections. This demonstrates that the subsequence based approach and the statistical framework employed in this study is also applicable to longer run lengths.

5 Conclusions

This study has presented a novel statistical framework for the analysis of subsequences of low level heuristics based on the concept of logarithmic returns. Log returns are used for the categorisation and selection of subsequences of heuristic selections based on their associated objective function values. Log returns can also be applied to the measurement and analysis of hyper-heuristic performance. This framework has been used to demonstrate that

1. the expected exploration-exploitation behaviour in sequences is seen in some, but not all domains,
2. offline learning can outperform online learning of heuristic subsequences,
3. the combination of heuristics into subsequences outperforms individual heuristic selections, and
4. generalisation across domains is possible for 3 out of the 4 domains tested.

Specifically, the unit log return function β is used to compare subsequences that have different lengths or have objective values that have different ranges. The mean final log return $\overline{\alpha}_f$ is used to compare hyper-heuristic performance over a number of problems and problem domains, and provides a better measure of performance than the arithmetic mean.

The γ-ratio is used to select sets of subsequences of heuristic selections from the offline database. The subsequence set TOP-DH of low level heuristics, and the set TOP-DC of heuristic classes, are selected according to a γ-ratio that is calculated for each particular domain. When the EvalHH hyper-heuristic is parameterised with these sets it significantly outperforms the SSHH hyper-heuristic on the HyFlex problems. The SSHH hyper-heuristic is known to perform well on HyFlex, and this result demonstrates the utility of the proposed framework.

In order to determine the existence of effective cross-domain subsequences, statistics are also calculated for heuristic classes across the whole database. The sets TOP-GC and BOT-GC contain the subsequences with the largest and the smallest γ-ratios respectively. The results of using these subsequence sets in EvalHH demonstrate that the effective subsequences TOP-GC perform better than the disruptive sequences BOT-GC on 3 out of the 4 problem domains, and that this improvement in performance is also statistically significant.

The BOT-GC subsequence set is more effective than the TOP-GC subsequences on the Bin Packing domain. This discrepancy can be explained by recalculating the γ-ratio for these sets on the BP domain, and observing that the γ value for BOT-GC is now larger than the γ value for TOP-GC. The set TOP-DC contains the subsequences with the largest γ-ratio for each domain. On the BP domain, the TOP-DC subsequences perform almost as well as the BOT-GC subsequences, and outperform TOP-GC. The change in γ for BOT-GC is due to the differing performances of the low level heuristics on different domains, and these differences can be quantified by calculating the mean log returns $\overline{\alpha}$ of the heuristic classes in each domain.

The issue of subsequence length is examined by selecting subsequences of unrestricted lengths. The resulting LONG-GC subsequence set contains subsequences of up to length 7 and outperforms the TOP-GC subsequence set, albeit slightly, with 99% confidence.

Lastly, the issue of run length is addressed by evaluating the TOP-DH and DIS–LLH subsequences for 10 min of wall clock time. The results show that the differences in performance observed in the previous experiments are still present after significantly larger numbers of heuristic selections.

This work has demonstrated that subsequences can be reliably extracted in an offline manner from a database of heuristic operations that have both effective and disruptive characteristics. It has also been shown that if those subsequences are well-chosen, they can provide significant improvements in performance. The approach has also shown that differences between problem domains and the interface between domain and heuristic class can be quantified by using statistics which is an important consideration for generalist algorithms such as these.

Effective subsequences of low level heuristics can be used to directly construct a selection hyper-heuristics, or used as training patterns for an offline learning algorithm for a specific problem. Furthermore subsequences of heuristic classes can, in some cases, be useful in constructing optimisers for problems from novel domains.

These findings underpin research in hyper-heuristics that have proposed large numbers of algorithms (many of which have been successful) without investigating the fundamental nature of the low level heuristics and in particular subsequence selection. Finally, it is encouraging to see that an offline approach to heuristic selection is able to outperform an online approach. Although the improvement in performance is quite narrow in the experiments presented here, it perhaps points to a future of hybrid offline and online selection hyper-heuristics.

References

Asmuni, H., Burke, E.K., Garibaldi, J.M.: Fuzzy multiple heuristic ordering for course timetabling. In: The Proceedings of the 5th United Kingdom Workshop on Computational Intelligence, pp. 302–309 (2005)

Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput. Surv. **35**(3), 268–308 (2003)

Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R.: Hyper-heuristics: a survey of the state of the art. J. Oper. Res. Soc. **64**(12), 1695–1724 (2013)

Burke, E.K., Kendall, G., Soubeiga, E.: A Tabu-search hyperheuristic for timetabling and rostering. J. Heuristics **9**(6), 451–470 (2003)

Burke, E.K., Petrovic, S., Qu, R.: Case-based heuristic selection for timetabling problems. J. Sched. **9**(2), 115–132 (2006)

Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In: Burke, E.K., Erben, W. (eds.) Practice and Theory of Automated Timetabling III. Lecture Notes in Computer Science, vol. 2079, pp. 176–190. Springer, Berlin (2001)

Crowston, W.B., Glover, F., Thompson, G.L., Trawick, J.D.: Probabilistic and Parametric Learning Combinations of Local Job Shop Scheduling Rules. ONR Research Memorandum. Carnegie Mellon University, Pittsburgh (1963)

Dempster, P., Drake, J.H.: Two frameworks for cross-domain heuristic and parameter selection using harmony search. In: Kim, H.J., Geem, W.Z. (eds.) Harmony Search Algorithm: Proceedings of the 2nd International Conference on Harmony Search Algorithm (ICHSA), pp. 83–94. Springer, Berlin (2016)

Derrac, J., García, S., Molina, D., Herrera, F.: A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evolut. Comput. **1**(1), 3–18 (2011)

Drake, J.H., Özcan, E., Burke, E.K.: An improved choice function heuristic selection for cross domain heuristic search. In: Coello, C.A.C., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) Parallel Problem Solving From Nature (PPSN). Lecture Notes in Computer Science, vol. 7492, pp. 307–316. Springer, Berlin (2012)

Drake, J.H., Özcan, E., Burke, E.K.: A comparison of crossover control mechanisms within single-point selection hyper-heuristics using HyFlex. In: IEEE Congress on Evolutionary Computation (CEC), pp. 3397–3403 (2015)

Fang, H.L., Ross, P., Corne, D.: A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In: Proceedings of the 5th International Conference on Genetic Algorithms, pp. 375–382. Morgan Kaufmann (1993)

Fisher, H., Thompson, G.: Probabilistic learning combinations of local job-shop scheduling rules. In: Muth, J., Thompson, G. (eds.) Industrial Scheduling, pp. 225–251. Prentice Hall, Upper Saddle River (1963)

Gratch, J., Chien, S.: Adaptive problem-solving for large-scale scheduling problems: a case study. J. Artif. Intell. Res. **4**(1), 365–396 (1996)

Kheiri, A., Keedwell, E.C.: A sequence-based selection hyper-heuristic utilising a hidden Markov model. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 417–424. ACM (2015)

Kheiri, A., Keedwell, E.C.: A hidden Markov model approach to the problem of heuristic selection in hyper-heuristics with a case study in high school timetabling problems. Evolut. Comput. **25**(3), 473–501 (2017)

Lehre, P.K., Özcan, E.: 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), pp. 97–104. ACM (2013)

Mısır, M., Verbeeck, K., Causmaecker, P.D., Berghe, G.V.: The effect of the set of low-level heuristics on the performance of selection hyper-heuristics. In: Coello, C.A., Cutello, V., Deb, K., Forrest, S., Nicosia, G., Pavone, M. (eds.) Parallel Problem Solving from Nature: PPSN XII, pp. 408–417. Springer, Berlin (2012)

Mısır, M., Verbeeck, K., Causmaecker, P.D., Berghe, G.V.: An intelligent hyper-heuristic framework for CHeSC 2011. In: Revised Selected Papers of the 6th International Conference on Learning and Intelligent Optimization: Volume 7219 (LION), pp. 461–466. Springer, New York (2012)

Mısır, M., Verbeeck, K., Causmaecker, P.D., Berghe, G.V.: A new hyper-heuristic as a general problem solver: an implementation in HyFlex. J. Sched. **16**(3), 291–311 (2013)

Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., Burke, E.K.: HyFlex: a benchmark framework for cross-domain heuristic search. In: Hao, J.K., Middendorf, M. (eds.) Evolutionary Computation in Combinatorial Optimization, pp. 136–147. Springer, Berlin (2012)

Ortiz-Bayliss, J.C., Terashima-Marín, H., Özcan, E., Parkes, A.J., Conant-Pablos, S.E.: Exploring heuristic interactions in constraint satisfaction problems: a closer look at the hyper-heuristic space. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 3307–3314 (2013)

Özcan, E., Mısır, M., Ochoa, G., Burke, E.K.: A reinforcement learning-great-deluge hyper-heuristic for examination timetabling. Int. J. Appl. Metaheuristic Comput. **1**(1), 39–59 (2010)

Petrovic, S., Epstein, S.L.: Random subsets support learning a mixture of heuristics. Int. J. Artif. Intell. Tools **17**(3), 501–520 (2008)

Ross, P., Schulenburg, S., Marín-Bläzquez, J.G., Hart, E.: Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 942–948. Morgan Kaufmann (2002)

Soria-Alcaraz, J.A., Ochoa, G., Sotelo-Figeroa, M.A., Burke, E.K.: A methodology for determining an effective subset of heuristics in selection hyper-heuristics. Eur. J. Oper. Res. **260**(3), 972–983 (2017)

Soria-Alcaraz, J.A., Ochoa, G., Swan, J., Carpio, M., Puga, H., Burke, E.K.: Effective learning hyper-heuristics for the course timetabling problem. Eur. J. Oper. Res. **238**(1), 77–86 (2014)

Terashima-Marín, H., Ortiz-Bayliss, J.C., Ross, P., Valenzuela-Rendón, M.: Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 571–578. ACM (2008)

Walker, J.D., Ochoa, G., Gendreau, M., Burke, E.K.: Vehicle routing and adaptive iterated local search within the HyFlex hyper-heuristic framework. In: 6th International Conference on Learning and Intelligent Optimization (LION), pp. 265–276. Springer, New York (2012)

Yates, W.B., Keedwell, E.C.: Clustering of hyper-heuristic selections using the Smith–Waterman algorithm for offline learning. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 119–120. ACM (2017)

Yates, W.B., Keedwell, E.C.: Offline learning for selection hyper-heuristics with Elman networks. In: Lutton, E., Legrand, P., Parrend, P., Monmarché, N., Schoenauer, M. (eds.) Artificial Evolution. Lecture Notes in Computer Science, vol. 10764, pp. 217–230. Springer, Berlin (2018)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.