Offline Learning for Sequence-based Selection Hyper-heuristics

Submitted by William Brian Yates to the University of Exeter as a thesis for the degree of
Doctor of Philosophy
In August 2020

I certify that all material in this thesis which is not my own work has been identified and that no material has previously been submitted and approved for the award of a degree by this or any other University.

Signature: .....

Abstract

This thesis is concerned with finding solutions to discrete NP-hard problems. Such problems occur in a wide range of real-world applications, such as bin packing, industrial flow shop problems, determining Boolean satisfiability, the traveling salesman and vehicle routing problems, course timetabling, personnel scheduling, and the optimisation of water distribution networks. They are typically represented as optimisation problems where the goal is to find a "best" solution from a given space of feasible solutions. As no known polynomial-time algorithmic solution exists for NP-hard problems, they are usually solved by applying heuristic methods.

Selection hyper-heuristics are algorithms that organise and combine a number of individual low level heuristics into a higher level framework with the objective of improving optimisation performance. Many selection hyper-heuristics employ learning algorithms in order to enhance optimisation performance by improving the selection of *single* heuristics, and this learning may be classified as either online or offline. This thesis presents a novel statistical framework for the offline learning of *subsequences* of low level heuristics in order to improve the optimisation performance of sequenced-based selection hyper-heuristics.

A selection hyper-heuristic is used to optimise the HyFlex set of discrete benchmark problems. The resulting sequences of low level heuristic selections and objective function values are used to generate an offline learning 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, that tend to lead to improvements in optimisation performance, and "disruptive" subsequences that tend to lead to worsening performance. Effective subsequences are used to improve hyper-heuristics performance directly, by embedding them in a simple hyper-heuristic design, and 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 results presented here demonstrates that the use of well chosen subsequences of heuristics can lead to small, but statistically significant, improvements in optimisation performance.

Dedication

This thesis is dedicated to Emilia Crespo Calvete (1 May 1927 - 28 February 2016).

Contents

List of Figures

List of Algorithms

Chapter 1

Introduction

1.1 Overview

This thesis is concerned with finding solutions to discrete NP-hard problems. Such problems occur in a wide range of real-world applications, such as bin packing, industrial flow shop problems, determining Boolean satisfiability, the traveling salesman and vehicle routing problems, course timetabling, personnel scheduling, and the optimisation of water distribution networks. These problems are typically expressed as optimisation problems where the goal is to find a "best" solution from a given space of feasible solutions. As no known polynomial-time algorithmic solution exists for NP-hard problems, they are usually solved by applying heuristics. Heuristics are processes or methods that learn to employ loosely defined rules of thumb or trial and error in order to resolve a given problem.

Hyper-heuristics

In their seminal work on the job scheduling problem [46] and [27] 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* [10] [8] [12] and more recently *hyper-heuristics* [22] [15] [14] [16].

Higher level heuristics are algorithms that organise and combine a number of individual low level heuristics into a higher level framework with the objective of improving optimisation performance. Metaheuristics and hyper-heuristics differ in a number of respects. For example, metaheuristics can access problem specific knowledge, while a hyper-heuristic which is subject to the limitations of the *domain barrier* cannot. As a result, metaheuristic approaches tend to be rich in domain knowledge, requiring significant expertise in both the problem and the available heuristics, and so they can be

expensive to implement and maintain. Furthermore, a metaheuristic approach developed for one particular problem may not be applicable to other problem domains, or even other instances of the same or similar problems [25]. The domain barrier requires a hyper-heuristic to perform well in the absence of problem specific knowledge and therefore, it is hoped, to be "re-useable" across different problems and problem domains with minimal changes. This places hyper-heuristics at a higher level of abstraction than many metaheuristics.

Hyper-heuristics may be classified according to a number of criteria. In general, hyper-heuristics can be used to either *generate* or *select* low level heuristics [15]. A generation hyper-heuristic generates new heuristics from primitive components or observed search trails of existing heuristics. Most of the approaches to heuristics generation in the literature employ genetic programming [72] to discover new heuristics by modifying or combining existing low level heuristics [16]. A selection hyper-heuristic selects and applies a heuristic chosen from a set of low level heuristics. The goal of both types of hyper-heuristic is to improve the search process through learning and/or optimisation. This thesis is concerned with selection hyper-heuristics.

Learning Algorithms

Most hyper-heuristics (and some metaheuristics) employ learning algorithms [8]. Typically, the learning algorithm optimises a hyper-heuristic's internal structure and parameters in order to improve the selection or generation of heuristics. Such learning algorithms can be categorised as either *online* or *offline*. Online learning is performed during the execution of a hyper-heuristic on a particular problem, while offline learning is performed on a number of benchmark training problems. The objective of online learning is to improve optimisation performance on the problem at hand. In contrast, a successful offline learning algorithm must *generalise* across the benchmark training problems leading to improvements in optimisation performance on *unseen* test problems. Offline learning can be further classified as either *intra-domain* or *cross-domain*. In intra-domain learning, the training sequences and the test optimisation problem are drawn from the same problem domain. In cross-domain learning, the training sequences and test problem can be drawn from different domains.

In both the online and offline cases, a learning algorithm will attempt to identify effective heuristics that tend to lead to improvements in optimisation performance, and disruptive heuristics that tend to lead to worsening performance. A hyper-heuristic uses this learnt knowledge to favour effective heuristics over disruptive ones during the optimisation process. Typically, such learning algorithms are trained to produce single heuristic selections (or in some cases heuristic pairs), with little regard for any synergies that may occur between a set of individual heuristics.

A Sequence-based Approach

Hyper-heuristics and metaheuristics move through a search space by applying sequences of perturbations to a set of existing solutions. Although the mechanisms 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. Furthermore, it is well known that the effectiveness of heuristics can vary during the optimisation process. For example, in [97] the authors observe that some heuristics are more effective at the beginning of the optimisation problems, and less effective at the end, and *vice versa*, while others heuristics are mainly used at the beginning, middle or end of the process.

From these two small examples it becomes clear that the correct ordering of a sequence of heuristics is imperative if search efficacy is to be achieved. The idea that the order in which a set of heuristics are applied is important leads naturally to the study of sequences (and subsequences) of heuristics.

Adopting a sequence-based approach can also reduce the number of objective function evaluations required- in the first example given above, 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.

A Framework for Offline Learning

This thesis introduces a novel statistical framework for the offline analysis of heuristic selections. An offline learning database of heuristic selections, and their associated objective function values, is generated by repeatedly executing an unbiased selection hyper-heuristic on a number of problem instances and problem domains. The problem instances and domains are drawn from the HyFlex benchmark set [84], which is a well known set of discrete optimisation problems that has been used in a number of studies.

The proposed framework is intended to be independent of the problem domain, the number or

type of heuristics, or the process used to generate the heuristic selections. The framework employs logarithmic returns which are used widely in finance where they are employed to compare two or more variables when the originating price series consist of highly unequal values [61]. Here, logarithmic returns are used to normalise subsequences of objective function values.

The framework is used to quantify the behaviour of heuristic selections such as

1. individual heuristics,
2. subsets of heuristics, and
3. subsequences of heuristics.

Statistics are calculated over the sets of heuristic selections that occur in the database, and these statistics are used to quantify heuristic behaviours such as optimisation performance, the effect of heuristic order on performance, and the changes in performance that occur during the optimisation process. A number of performance measures based on logarithmic returns are proposed and evaluated, such as the mean log return $\overline{\alpha}$ of a heuristic, the mean unit log return $\overline{\beta}$ and the γ-ratio of a subset or subsequence of heuristics, and the order sensitivity δ of a subset of heuristics. In general, when the goal of optimisation is to minimise a given objective function, heuristic selections can be categorised as either *effective* when they tend to decrease the objective function value, or *disruptive* when they tend to increase the objective function value.

By quantifying heuristic behaviour both within and between problem domains, it becomes possible to employ visualisation techniques as an aid to the analysis of heuristics selections.

Although the framework can be applied to a wide range of discrete and continuous optimisation problems, it should be noted that a candidate problem must have at least two low level heuristics, and these heuristics must have different performance characteristics; otherwise there is nothing to be learnt from the heuristic selections. Furthermore, the framework has only been applied to deterministic problems, static problems (rather than dynamic ones) which are fixed during the optimisation process, and problems with "noiseless" objective functions. An appraisal of the framework when applied to stochastic, dynamic or "noisy" optimisation problems could form the basis for future research.

Effective Subsequences

It is hoped that by investigating the effect of order on subsets and subsequences of heuristics, it will be possible to demonstrate that subsequences are useful structural components of the optimisation process. Effective subsequences of heuristics could be used to improve the optimisation performance of sequence-based hyper-heuristics either directly, by embedding them in a suitable hyper-heuristic design, or indirectly as the inputs to an appropriate offline learning algorithm. This thesis presents examples of both subsequence embedding, and offline subsequence learning. Specifically, subsequences

are chosen from an offline learning database using the γ-ratio and embedded in an unbiased, sequence-based hyper-heuristic, denoted EvalHH, in order to evaluate the subsequences, and by extension the framework. Here the objective is to demonstrate that offline subsequence learning can achieve results that are comparable to online subsequence learning. With this in mind, the optimisation performance of EvalHH is compared with results produced by the SSHH hyper-heuristic [67]. The SSHH hyper-heuristic is a published, sequenced-based hyper-heuristic with online learning capabilities that is known to perform well on the HyFlex problems. Effective subsequences can also be used to offline train the SSHH hyper-heuristic using the Baum-Welch learning algorithm [95]. In this case, the framework is used to investigate the potential for *hybrid* subsequence learning which is a mixture of online and offline learning.

1.2 Research Contributions

The framework presented here is intended to contribute to research in the fields of hyper-heuristic offline learning, and the role of heuristic subsequences in optimisation. The primary objectives are to

1. quantify the performance of heuristic selections in order to identify effective subsequences of heuristics in an offline learning database,
2. quantify the effect of order on heuristic subsets and subsequences,
3. demonstrate that subsequences of heuristics are useful structural components of the optimisation process,
4. quantify the scope and limits of offline subsequence learning,
5. determine the existence of effective cross-domain subsequences of heuristic classes in order to investigate the potential for cross-domain learning and generalisation,
6. investigate *hybrid* subsequence learning, and
7. investigate the potential for *scalable* learning where effective subsequences learned from small problems are applied to larger, computationally expensive problems.

These innovations advance the field of hyper-heuristics in terms of performance, and their potential for application to problem instances and domains that could not have been considered previously due to their size or complexity.

1.3 Published Work

A number of papers have been written based on the work described in this thesis. These are:

[123] Yates, W. B., and 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* (2017), GECCO, ACM, pp. 119–120.

In this paper, the similarities and dissimilarities that occur between *sequences* of heuristics in an offline learning database are identified and analysed algorithmically. The sequences of heuristics are generated by an individual hyper-heuristic run on a given problem instance, and constitute a trace of the optimisation process. By employing a suitable measure of similarity, the sequences of the database can be grouped or clustered according to the view of the similarity algorithm. It can be shown that by using a well-known algorithm from bioinformatics more commonly used to explore the conserved regions of DNA sequences, the Smith-Waterman algorithm [106], it is possible to characterise problems using only the sequence of heuristic choices made by the hyper-heuristic. The Smith-Waterman algorithm is able to provide a measure of the level of similarity between two strings operating over any alphabet, and is used in [123] to define a distance function between sequences of heuristics which is then used to perform a cluster analysis. The results presented show that the Smith-Waterman algorithm is able separate the offline database into distinct problem domains.

The automatic separation and identification of problem domains from sequences of heuristics is useful because it demonstrates that there are subsequences of heuristics that are common to each problem domain, and that these subsequences vary between domains. This strengthens the thesis that subsequences of heuristics play an important role in the optimisation process. In addition, the identification of a (similar) problem domain can improve the choice of learning algorithm, learning algorithm parameterisation, and training data. For example, in [113] a k-nearest neighbour classifier is used to identify problems in an offline database that are similar to a target problem based on a set of measurable problem characteristics. This *metaknowledge* is then used to retrieve further problem specific information which is used to optimise the performance of a planning algorithm. The method described here differs from conventional metalearning approaches to algorithm selection in that, as only sequences of low level heuristic classes are employed, no problem specific information is required, preserving the domain barrier.

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

In this paper, an Elman network is trained on sequences of heuristic selections chosen from an offline learning database, and its ability to learn and generalise from these sequences is evaluated. Elman networks are recurrent neural networks that learn from, process and produce sequences of data. They are typically applied to problems which express themselves naturally as temporal sequences such as natural language processing applications [40] [41].

The networks are trained using a leave-one-out cross-validation methodology and the sequences of heuristic selections they produce are tested on unseen problems drawn from the HyFlex benchmark set. The results demonstrate that the Elman network is capable of intra-domain learning and generalisation with 99% confidence. In this context, generalisation means that the network is able to produce a sequence of heuristic selections which, when evaluated on unseen examples, outperform the training sequence. When the network was trained using a cross-domain training set, the Elman network did not exhibit generalisation indicating that cross-domain generalisation is a more difficult problem and that strategies learned on one domain cannot necessarily be transferred to another.

The work in [123] and [124] has been omitted from the main body of this thesis because it was felt that these studies, which focus on sequences, were peripheral to the general study of subsets and subsequences. For completeness, the published texts have been included in Appendix A and Appendix B, respectively. It should be noted that following the publication of [123] the suitability of the parametric paired t-test that is used to validate the results came into question. Although the t-test is quite robust, the assumption of normality is not generally met in machine learning applications, and as a consequence all the statistical significance tests have been repeated, successfully, using the non-parametric Wilcoxon signed rank test employed throughout the remainder of this study (see Chapter 3, Section 3.5).

[126] Yates, W. B., and Keedwell, E. C. An analysis of heuristic subsequences for offline hyper-heuristic learning. *Journal of Heuristics 25*, 3 (2019), 399–430.

This paper contains the essential elements of the framework which is presented in Chapter 3 and the results of applying this framework to subsequences in Chapter 4.

[127] Yates, W. B., and Keedwell, E. C. Combining online and offline learning for a sequence-based selection hyper-heuristic. *In preparation* (2020)

This paper describes the results of combining offline and online subsequence learning contained in Chapter 5.

[125] Yates, W. B., and Keedwell, E. C. Analysing heuristic performance for optimising water distribution networks. In *17th International Computing and Control for the Water Industry Conference (CCWI)* (2019), Extended Abstract.

[128] Yates, W. B., and Keedwell, E. C. Offline learning with a selection hyper-heuristic: An application to water distribution network optimisation. *Evolutionary Computation*, accepted (2020).

These papers present the results of applying the framework to the optimisation of water distribution networks, and form the basis for Chapter 6.

1.4 Thesis Structure

This thesis presents a novel statistical framework for the offline learning of effective subsequences of low level heuristics in order to improve the optimisation performance of sequenced-based selection hyper-heuristics.

The structure of this thesis is as follows. Chapter 2 contains a general description of the optimisation problem together with a number of real world examples. It also presents a classification of metaheuristic and hyper-heuristic methods in general, and selection hyper-heuristics in particular. Chapter 3, presents a framework for offline learning, based on the mathematical concept of logarithmic returns. This framework is used to construct and analyse an offline learning database generated by repeatedly executing a selection hyper-heuristic on the HyFlex benchmark set. In Chapter 4 this framework is used to identify and analyse subsequences of heuristic selections. Specifically, the γ-ratio is introduced, and used to select sets of subsequences of heuristics from a number of offline learning databases. These subsequence sets are used to parameterise a sequenced-based, selection hyper-heuristic which is executed on a number of unseen HyFlex problem. In Chapter 5 the framework is applied to mixed or hybrid learning which is a combination of offline and online learning. Specifically, the SSHH hyper-heuristic, is trained offline with the Baum-Welch learning algorithm. The offline trained SSHH is then evaluated on unseen HyFlex problem instances. In Chapter 6 the methodology developed and evaluated on the HyFlex problems is applied to a novel, real-world problem domain; the optimisation of water distribution networks, and the potential for scalable learning, where knowledge learned from a small problem is usefully transferred to a second larger problem, is explored. Finally, Chapter 7 presents the conclusions of this thesis.

Appendices A and B contain the published texts of [123] and [124] respectively. Appendix C contains an overview of the relational schema used to implement an offline learning database.

Chapter 2

Background

Heuristics are processes or methods that learn to employ loosely defined rules of thumb or trial and error in order to solve a given problem. They are usually applied to computationally NP-hard problems where no known effective algorithmic solution exists. Such problems occur in a wide range of real-world applications, such as bin packing, industrial flow shop problems, determining Boolean satisfiability, the traveling salesman and vehicle routing problems, course timetabling, personnel scheduling, and the optimisation of water distribution networks. Typically, the problems are expressed as optimisation problems where the goal is to find a "best" solution from a given space of feasible solutions, and such problems arise in a wide variety of real world applications.

Higher level heuristic approaches such as metaheuristics and hyper-heuristics are algorithms that organise and combine a number of individual heuristics into a higher level framework with the objective of improving optimisation performance.

Most hyper-heuristics (and some metaheuristics) employ learning algorithms. This learning can be categorised as either online or offline. Online learning is performed during the execution of a high level heuristic on a particular problem, while offline learning is performed on a number of benchmark training problems. In each case, a learning algorithm will attempt to identify effective heuristics that tend to lead to improvements in optimisation performance, and disruptive heuristics that tend to lead to worsening performance. A high level heuristic uses this learnt knowledge to favour effective heuristics over disruptive ones during the optimisation process.

Many of the learning algorithms employed by high level heuristics are trained to produce single heuristic selections (or in some cases heuristic pairs), with little regard for any synergies that may occur between a group of individual heuristics. Furthermore, in many problem domains there exist large numbers of potential heuristics to choose from, and selecting an appropriate subset of heuristics can be non-trivial. These considerations have led to interest in the construction and performance of *subsets* of heuristics, and the interactions that occur between the elements of such subsets. However,

a subset is, by definition, an unordered collection of distinct elements, and it is well known that the order in which heuristics are applied during the optimisation process is also important if search efficacy is to be achieved. This leads naturally to a consideration of *subsequences* of heuristics, that is, ordered groups of heuristics.

This thesis examines the potential for improving hyper-heuristic optimisation performance by learning, offline from effective subsequences of heuristics.

The structure of the chapter is as follows. Section 2.1 begins with an overview of the field of optimisation and provides a number of examples of canonical optimisation problems. This is followed in Section 2.2 with an introduction to the higher level heuristic approaches that are typically employed to solve such problems. Specifically, Section 2.2.1 contains a description and classification of metaheuristic methods, while Section 2.2.2 contains a description and classification of hyper-heuristic methods. Section 2.2.3 contains a description and classification of selection hyper-heuristics which are the focus of this study. In Section 2.3, the concept of heuristic sequence or order which is central to this thesis is presented, and placed within the context of the wider literature. Specifically, Section 2.3.1 discusses the construction and performance of subsets of heuristics, while Section 2.3.2 discusses subsequences of heuristics. Section 2.4 presents a discussion of learning for hyper-heuristics, and contains examples of the methodologies and algorithms that have been successfully employed for offline learning. Section 2.5 describes the sequence-based selection hyper-heuristic, SSHH which is employed to test the offline subsequence learning methodology introduced in this thesis. Finally, in Section 2.6 the *No Free Lunch* theorems for optimisation are summarised, and their applicability to hyper-heuristics is discussed.

2.1 Optimisation Problems

Optimisation is a branch of applied mathematics that addresses the problem of finding the best solution from a given space of feasible solutions, and such problems arise in a wide variety of real world applications [59]. A typical optimisation problem is the allocation of valuable resources among possible alternative uses in order to minimise an objective function such as the total monetary cost.

In practice, optimisation is the minimisation (or maximisation) of a given real valued *objective function* $f : X \longrightarrow \mathbb{R}$ of n *decision variables* $x = (x_1, \ldots, x_n) \in X$ which may be subject to a number m of functional *constraints* $g_1, \ldots, g_m$. Optimisation problems may be classified as *discrete* when the decision variables are drawn from a discrete set, often a subset of integers, or *continuous* when the decision variables are real valued. The functional constraints are usually expressed as equalities of the form $g_i(x) = 0$ for $i \in E$, and inequalities of the form $g_i(x) \geq 0$ for $i \in I$. Constraints can be classified as either *hard* constraints that cannot be violated under any circumstances, or *soft* constraints which should be satisfied as much as is possible. Problems that have constraints are termed *constrained* optimisation problems, while problems without constraints are termed *unconstrained*. Some optimisation problems have no natural objective function and consist solely of constraints. Such optimisation

problems are termed *constraint satisfaction* or *feasibility* problems. In contrast, other optimisation problems have multiple objective functions $f_1, \ldots, f_N$. Here the task is to minimise (or maximise) all N objective functions simultaneously. Thus, a general optimisation problem can be specified as:

$$\min_{x \in X} f_1(x), \ldots, f_N(x) \text{ such that } (\forall i \in E) \;\; g_i(x) = 0 \text{ and } (\forall i \in I) \;\; g_i(x) \geq 0.$$

There are a number of canonical optimisation problems that have received a large amount of attention in the literature. The optimisation problems considered in this thesis are summarised below.

Bin Packing

The *Bin Packing* problem (BP) concerns the packing of items into a finite number of bins or containers in such a way so as to minimises the number of bins used [64]. In the one dimensional bin packing problem (1D BP), each item j has a weight w_j, and each bin has capacity c. The objective is to minimise the number of bins used, where each item is assigned to exactly one bin, and the weight of the items in each bin does not exceed c. Mathematically, let

$$y_i = \begin{cases} 1 & \text{if bin } i \text{ contains items,} \\ 0 & \text{otherwise} \end{cases}$$

and

$$x_{i,j} = \begin{cases} 1 & \text{if bin } i \text{ contains item } j, \\ 0 & \text{otherwise.} \end{cases}$$

Then, the function to be minimised is

$$N(y) = \sum_{i=1}^{n} y_i$$

with hard constraints

$$(i = 1, \ldots, n) \;\; \sum_{j=1}^{n} w_j x_{i,j} \leq c y_i$$

$$(j = 1, \ldots, n) \;\; \sum_{i=1}^{n} x_{i,j} = 1.$$

The first constraint ensures that each bin's capacity is not exceeded, while the second constraint ensures that each item is assigned to exactly one bin.

A practical application of the 1D BP problem is cutting lengths of stock material that has fixed width, such as pipes, or metal beams. When the number of bins is restricted to 1 and each item is characterised by a volume and a monetary value, the problem of maximising the value of items that can fit in the bin is known as the *Knapsack* problem.

Permutation Flow Shop

The *Permutation Flow Shop* (PFS) problem concerns the scheduling of n jobs on m machines [115]. Each job i consists of m operations where the j-th operation is to be performed, in order, by machine j for a specified time $p_{i,j}$. Intuitively, for each job, the first operations is processed by the first machine, then as the first operation is finished, the second operation is processed by the second machine, and so on, until the m-th operation, when the job is considered complete. The problem definition implies that the order in which each machine processes the jobs is identical for all machines. Mathematically, consider a permutation $\pi = \pi(1), \pi(2), \ldots, \pi(m)$ where $\pi(q)$ is the index of the operation in the q-th position. Given π, a unique *schedule* can be obtained by calculating the start and completion times of each operation on each machine. The start time $s_{\pi(q),j}$ of the q-th operation on machine j is

$$(q, j = 1, \ldots, m) \;\; s_{\pi(q),j} = \max\{\; s_{\pi(q),j-1},\, s_{\pi(q-1),j} \;\}$$

where

$$s_{\pi(q),0} = s_{\pi(0),j} = 0.$$

The completion time $C_{\pi(q),j}$ of the qth operation on machine j is

$$(q, j = 1, \ldots, m) \;\; C_{\pi(q),j} = s_{\pi(q),j} + p_{\pi(q),j}.$$

Given a schedule, let C_i be the time when job i completes its processing on the last machine m. A solution to the PFS problem is a processing order that minimises the completion time of the last job to exit the shop, that is, a processing order π that minimises the function

$$T(\pi) = \max_{i=1}^{n} C_i.$$

A related problem to the PFS is the *Job Shop Scheduling* problem (JSP) which again concerns the scheduling of n jobs on m machines. Each job i has a *duration* D_i and consists of a sequence of n_i operations that must be completed in that order. Each operation has a specific machine that it needs to be processed on and only one operation in a job can be processed at a given time. The objective is to minimise the duration or *makespan* which is the time that elapses between the start of a job and its finish. A common variant of the JSP is the *Flexible Job Shop* where each operation can be processed on any machine, that is, the machines are considered identical.

Boolean Satisfiability

The *Boolean Satisfiability* problem (SAT) concerns Boolean formulae expressed in conjunctive normal form, such as

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1$$

where $\wedge$, $\vee$, and $\neg$ denote the logical operations AND, OR, and NOT respectively. A Boolean expression is said to be *satisfiable* if it can be made TRUE by assigning appropriate logical values (TRUE or FALSE)

to its variables, which in the example above are x_1, x_2, and x_3. The objective of the SAT problem is to decide if some given Boolean expression is satisfiable. The SAT problem can be generalised to the *Maximum Satisfiability* problem (MAX-SAT) where the objective is to determine the maximum number of clauses that can be made true by a suitable assignment of truth values [63].

The Traveling Salesman Problem

The *Traveling Salesman* problem (TSP) concerns a salesman based at a city 1 who must visit cities $2, 3, \ldots, n$ and then return to city 1. The distance from city i to city j is denoted $c_{i,j}$. The objective is to visit each city exactly once, in a sequence that minimises the distance travelled. In symbols, let the elements of an $n \times n$ matrix x represent a *tour* of the n cities where

$$x_{i,j} = \begin{cases} 1 & \text{if there is a path from city } i \text{ to city } j, \\ 0 & \text{otherwise} \end{cases}$$

and define the distance travelled to be minimised as

$$D(x) = \min \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} c_{i,j} x_{i,j}$$

with hard constraints

$$(j = 1, \ldots, n) \quad \sum_{i=1, i \neq j}^{n} x_{i,j} = 1$$

$$(i = 1, \ldots, n) \quad \sum_{j=1, j \neq i}^{n} x_{i,j} = 1$$

$$(\forall Q \subseteq \{1, \ldots, n\}, |Q| \geq 2) \quad \sum_{i \in Q} \sum_{j \in Q} x_{i,j} < |Q| - 1.$$

The first constraint ensures that each city is arrived at from exactly one other city, while the second constraint ensures that from each city there is a departure to exactly one other city. The last constraint ensures that there is only a single tour covering all the cities, and not two or more disjoint subtours. The TSP can be generalised to the *Vehicle Routing Problem* (VRP). Here, the objective is to find a set of optimal routes for a fleet of m vehicles to traverse in order to make deliveries to a given set of customers.

Course Timetabling

The *Course Timetabling* problem (CTP) consists of a set of time periods, a set of events to run in those periods, a set of agents that attend the events, and a set of places where the events take place [108]. The agents are groups or classes of students and teachers, while the places represent the classrooms. Agents and places have hard and soft constraints associated with them, such as limits on the number of classes

students are required to take in one day, teacher workload or room capacity. Events can be single lessons or a set of lessons, and have a number of properties that influence the allocation of agents to them. These properties are expressed as constraints or other optimality criteria. Examples of event properties include the event's duration, restrictions that some events must take place in certain classrooms, the contribution of the event to workload, and pre-assigned time slots. The objective is to ensure that a number of additional constraints such as avoiding event clashes, limiting the "idle-time" of students or classrooms, or limiting teacher workloads, are met. Mathematically, the CTP can be represented as a constraint satisfaction problem where the decision variables are events. Specifically, consider a set of events (courses or subjects) $E = \{e_1, e_2, \ldots, e_n\}$, a set of time-periods $T = \{t_1, t_2, ..., t_s\}$, a set of places (classrooms) $P = \{p_1, p_2, ..., p_m\}$, and a set of agents (teachers and students) $A = \{a_1, a_2, ..., a_o\}$. Each element $e \in E$ is a unique event that requires the assignment of a time period $t \in T$, a place $p \in P$, and a set of agents $S \subseteq A$. An assignment is thus a quadruple (e, t, p, S), and a solution to the problem is a complete set of n assignments (one for each event) that satisfies the set of constraints.

Personnel Scheduling

The *Personnel Scheduling* problem (PS) concerns the allocation of a number of employees to a number of tasks over a specific planning period [29]. This involves deciding at which times and on which days the employees should work over the planning period. Although there is no canonical representation of the personnel scheduling problem, it constitutes an identifiable group of problems with a common structure but which differ in their constraints and objectives. For example, for some problems there may be a constraint on the maximum number of hours an employee can work in the planning period. Alternatively, in another problem this constraint may be an objective, so that the employee is allowed to exceed a certain number of hours but the excess is to be minimised. As in the CTP, the aim is to ensure that the given constraints and objectives are met.

Optimising Water Distribution Networks

Optimising the design and rehabilitation of water distribution networks (WDN) is an important real-world problem [120]. A WDN delivers water from reservoirs, tanks, and water treatment facilities to consumers via a network of pipes and makes use of pumps and valves to meet consumer demand. The optimisation of such networks aims to deliver water at an adequate pressure to all demand nodes for the minimum monetary cost. The monetary cost to be minimised is usually expressed in millions (M) and is defined by

$$C(D) = \sum_{i=1}^{np} U_c(D_i) L_i$$

where the decision variables D are the network's pipe diameters, U_c is the unit pipe cost which depends on the diameter D_i selected, and L_i is the length of pipe $i = 1, \ldots, np$. Although cost

minimisation is the primary objective, there are many other objectives that can also be considered such as minimisation of water age, adherence to velocity and pressure constraints, and increasing the robustness of the network to reduce supply outages.

2.2 High Level Heuristics

A *heuristic* is a process or method that learns to employ loosely defined rules of thumb or trial and error in order to solve a given problem. The word heuristic derives from the Greek verb *heuriskein* (or ευρισκειν) which means "to find", or "to discover". Heuristics are applied to computationally hard problems where no known tractable 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 shop scheduling problem [46] and [27] 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 [10] [8] [12] and more recently hyper-heuristics [22] [15] [14] [16].

2.2.1 Metaheuristics

Metaheuristics are algorithms that combine low level heuristic operations into a higher level framework with the objective of efficiently searching a problem space X [10]. The Greek prefix *meta* (or μετά) used in the word metaheuristic means "beyond" or "over". Although there is no generally accepted definition of what constitutes a metaheuristic there are a number of fundamental properties that characterise them. Specifically:

1. they are strategies that "guide" the search process,
2. such algorithms are approximate, and usually non-deterministic,
3. they are not problem specific,
4. though they may make use of problem specific knowledge, and
5. they may use search experience and learning to improve the search.

An important concept in metaheuristic search is the trade off between *diversification* and *intensification*. The term diversification refers to the global exploration of the search space, while intensification refers to the exploitation of the local search space. Maintaining a dynamic balance between diversification and intensification is crucial to metaheuristic performance [10]. This balance is necessary so that

regions of the search space that contain high quality solutions are quickly identified without wasting time in regions of the search space that have already been explored or do not contain high quality solutions.

Metaheuristics can be broadly classified as to whether they operate on a single point or solution, or are population based (or multi-point), and whether they have a memory or are memoryless. Metaheuristics can also be classified by the type of low level heuristics they employ.

Single point metaheuristics operate on a single solution, and such methods are also known as *trajectory methods* because the search process, starting from some single initial solution can be characterised as a trajectory in the search space. In contrast, population based metaheuristics employ a number of solutions. In general, single point metaheuristics are more exploitation-oriented whereas population based metaheuristics are more exploration-oriented [12].

The use of memory is recognised as one of the fundamental properties of a powerful metaheuristic. Memory can be characterised as either *short term* which keeps track of recently performed operations, solutions or, in general, decisions taken, such as a Tabu list, or *long term* which usually consists of a number of internal parameters that hold information accumulated over the search process, such as heuristic selection probabilities. Memoryless metaheuristics determine their next action solely on the current state of the search process [10].

The low level heuristics employed by a metaheuristics are typically problem specific, and can be broadly classified as either *constructive* or *perturbative* (also referred to as *local search*) operators [10]. Constructive heuristics are used to incrementally build a complete solution. A constructive metaheuristic, starting with an empty solution, makes selections from a pool of constructive heuristics for different stages of the construction process. For example, in Ant Colony Optimisation (ACO) [35], the ants incrementally build a solution to the problem by moving through the nodes of a construction graph. It should be noted that for constructive methods there is a natural end to the construction process, that is, when a complete solution is reached. Thus, a complete solution corresponds to a finite sequence of constructive heuristics (in the heuristic space) whose length is determined by the size of the underlying combinatorial problem. Constructive heuristics are typically the fastest approximate methods, but they often produce solutions of inferior quality when compared to perturbative heuristics [10]. In contrast, perturbative heuristics start with a complete solution, and try to iteratively improve the current solution using neighbourhood structures and/or simple local searches. Perturbative heuristics, such as mutation operators, can also allow a metaheuristic to escape from local optima by "jumping out" of the basin of attraction of the current local optimum.

There are many well known metaheuristics in the literature. For example, *Ant Colony Optimisation* algorithms (ACO) [35], *Evolutionary Computation* (EC) [39] including *Genetic Algorithms* (GA) [55], *Differential Evolution* (DE) [110], *Iterated Local Search* (ILS) [75], *Variable Neighbourhood Search* (VNS) [57], *Simulated Annealing* (SA) [71], and *Tabu Search* (TS) [54].

Algorithm 1 An ant colony optimisation algorithm (ACO) in pseudocode.

```
1. initialise ants and pheromone trails
2. while termination criteria not met do
3.     position each ant at a starting node
4.     repeat
5.         for all ants in the colony do
6.             choose ant's next node by applying state transition rule
7.             apply step-by-step pheromone trail update rule
8.         end
9.     until every ant has constructed a solution path
10.    update best path
11.    apply global pheromone evaporation update rule
12. end
```

Ant Colony Optimisation

Ant colony optimisation algorithms [35] are a family of methods that are typically employed to solve problems which can be reduced to finding optimal paths in some given graph, such as the TSP and the VRP. Ant colony optimisation employs a population of artificial ants, and often a number of local search algorithms. An artificial ant's behaviour is modelled on the behaviour of biological ants seeking a path between their colony and a source of food. The artificial ants explore the graph, thus constructing a solution, by following and laying down *pheromone trails* which act as a memory, and use a pheromone based communication system to control their search. Over time, the pheromone trails evaporate and thus the pheromone density becomes higher on the more traversed, shorter paths and lower on the less traversed, longer ones. The evaporation mechanism also helps the method escape local optima, by allowing early solutions to be "forgotten" and replaced by newer, better solutions. Pheromone based communication is an effective method of communication which is widely observed in nature, and is used by social insects such as bees, ants and termites for inter-agent and agent-swarm coordination [11]. The pseudocode for an ant colony optimisation metaheuristic is shown in listing 1.

Evolutionary Algorithms

Evolutionary and genetic algorithms [39] [55] are optimisation methods that are based on the Darwinian principles of *natural selection* and *survival of the fittest* [30], and the concept of *genetic recombination* originated in [26]. Typically, such algorithms consists of a population of *chromosomes* that represent the solutions of a problem, a *fitness* function, a *selection* strategy, a *replacement* strategy and genetic operators such as *crossover* and *mutate*. Given some initial, randomly generated population, the fitness function is used to assign a fitness value to each chromosome. Parent chromosomes are selected from the population so that chromosomes with higher fitnesses are more likely to be chosen than chromosomes with lower fitnesses. The genetic operators are applied to these parent chromosomes in order to produce child chromosomes. The child chromosomes are assigned fitness values and used to

replace weaker chromosomes in the population. By applying this process iteratively, the population evolves over time to become fitter, that is, the chromosomes in the population become better solutions to the problem encoded in the fitness function. The pseudocode for a general evolutionary algorithm is shown in listing 2.

Algorithm 2 An evolutionary algorithm (EA) in pseudocode.

```
1. randomly initialise chromosomes in population
2. evaluate chromosomes in population
3. while termination criteria not met do
4.     select strong parent chromosomes
5.     recombine and mutate pairs of parents to produce children
6.     evaluate child chromosomes
7.     replace weakest in population with child chromosomes
8. end
```

Differential Evolution

Differential evolution (DE) is a population based metaheuristic approach to minimising (possibly) nonlinear and non-differentiable continuous space functions [110]. The population consists of NP randomly generated D-dimensional solution or *parameter vectors*. Differential evolution generates a new parameter vector from a some target vector by selecting three other vectors at random from the population, and applying a mutation operation and a crossover operation. The mutation operation adds the weighted difference between two of the selected parameter vectors to the third vector. The crossover operation mixes this mutated vector's parameters with the parameters of the target vector to yield a trial vector to be evaluated. If the trial vector has a lower objective function value than the target vector, then the trial vector replaces the target vector, and this process is repeated, in parallel, for each of the NP parameter vectors in the population until a termination criteria is met. The pseudocode for a differential evolution algorithm is shown in listing 3.

Algorithm 3 A differential evolution algorithm (DE) in pseudocode.

```
 1. randomly initialise parameter vectors in population
 2. while termination criteria not met do
 3.     for each target vector in population do
 4.         evaluate target vector
 5.         select 3 vectors at random
 6.         mutate and crossover with target vector to produce trial vector
 7.         evaluate trial vector
 8.         if trial is superior to target vector then
 9.             replace target vector with trial vector
10.         end
11.     end
12. end
```

Iterated Local Search

Iterated local search (ILS) is a memoryless, single point metaheuristic that constructs a "sequence of local optima" by iteratively perturbing some current solution and then applying local search (such as steepest descent) to the modified solution [75]. If the new solution passes an acceptance test, then it becomes the current solution; otherwise the new solution is discarded, and the process is repeated with the current solution. The acceptance test balances intensification and diversification. For example, accepting only improving solutions favours intensification, while accepting any solution favours diversification. Between these two logical extremes it is possible to design intermediate strategies such as accepting improving solutions *or* non-improving solutions with some small probability p. The pseudocode for the ILS metaheuristic is shown in listing 4.

Algorithm 4 The ILS metaheuristic in pseudocode.

```
1. cur-sol ← initialiseSolution()
2. while termination criteria not met do
3.     new-sol ← perturb(cur-sol)
4.     new-sol ← localSearch(new-sol)
5.     if accept(new-sol, best-sol) is TRUE then
6.         cur-sol ← new-sol
7.     end
8.     if f(new-sol) < f(best-sol) then
9.         best-sol ← new-sol
10.    end
11. end
```

Variable Neighbourhood Search

Variable neighbourhood search (VNS) is a memoryless, single point metaheuristic that uses local search to explore a set of *neighbourhood structures* [57]. Specifically, given some value $k \in [k_{\min}, k_{\max}]$ and a solution $x \in X$, let $N_k(x)$ be a set of neighbourhoods of x of increasing size so that $|N_k(x)| < |N_{k'}(x)|$ when $k < k'$. A solution is randomly selected from a neighbourhood $N_k(x)$ of the current solution x. A local search procedure is then applied to the selected solution, and the new solution is accepted if it is superior to the current solution. Thus the VNS metaheuristic, like ILS, generates a sequence of local optima. However, the VNS metaheuristic differs from ILS in that VNS accepts only improving solutions and employs an adaptive perturbation procedure, where the size of the neighbourhood under consideration is governed by the parameter k. Switching between different neighbourhood structures allows VNS to escape from local optima. The pseudocode for the VNS metaheuristic is shown in listing 5.

Algorithm 5 The VNS metaheuristic in pseudocode.

```
1. cur-sol ← initialiseSolution()
2. k ← k_min
3. while termination criteria not met do
4.     new-sol ← extractNeighbour(cur-sol, k)
5.     new-sol ← localSearch(new-sol)
6.     if f(new-sol) < f(cur-sol) then
7.         cur-sol ← new-sol
8.         k ← k_min
9.     else
10.        k ← k + δk
11.    end
12.    if k > k_max then
13.        k ← k_min
14.    end
15. end
```

Simulated Annealing

Simulated annealing (SA) is a memoryless, single point metaheuristic. It is one of the first algorithms to have an explicit strategy for avoiding local minima [71]. The idea is to allow "uphill moves", that is solutions that are inferior to the current solution in order to escape from local minima. Simulated annealing is an abstraction of the physical annealing process of metals and glass, which assume low energy configurations when cooled with an appropriate cooling schedule. The algorithm starts by generating an initial solution and by initialising a temperature parameter T and a cooling parameter $c \in (0, 1]$. At each iteration, a new solution is randomly generated from the current solution and the difference in fitness $\Delta f = f(\text{new-sol}) - f(\text{cur-sol})$ is computed. New solutions are accepted if they are superior, that is $\Delta f < 0$ or with a probability $\exp(\frac{-\Delta f}{T})$ which follows a Boltzmann distribution. During the search process, the temperature T is decreased, so that at the beginning of the search, the probability of accepting uphill moves is high. As the temperature gradually decreases the algorithm converges to a simple iterative algorithm where only improving moves are accepted. The pseudocode for the SA metaheuristic is shown in listing 6.

Tabu Search

Tabu search (TS) is a single point metaheuristic that employs a local search algorithm, and a short term memory [54]. The short term memory is implemented as a *tabu list* that records the most recently visited solutions and forbids moves toward them. The tabu list ensures that a neighbourhood of the current solution will not contain any of the solutions on the list. At each iteration, the best solution from a neighbourhood around the current solution is chosen. The chosen solution is then added to the tabu list, while another solution already in the tabu list is discarded. The use of a tabu list prevents the algorithm returning to recently visited solutions which prevents cycling, and can, in some circumstances, force the search to accept "uphill moves", allowing the algorithm to escape from local

Algorithm 6 Simulated annealing (SA) in pseudocode.

```
1. initialise temperature T and cooling factor c
2. cur-sol ← initialiseSolution()
3. while termination criteria not met do
4.     for number of new solutions do
5.         new-sol ← perturb(cur-sol)
6.         Δf ← f(new-sol) - f(cur-sol)
7.         if Δf < 0 then
8.             cur-sol ← new-sol
9.         else if random() < exp(−Δf / T) then
10.            cur-sol ← new-sol
11.        end
12.    end
13.    T ← cT
14. end
```

minima. The pseudocode for the Tabu Search metaheuristic is shown in listing 7.

Algorithm 7 The Tabu search (TS) metaheuristic in pseudocode.

```
1. cur-sol ← initialiseSolution()
2. tabu-list ← ∅
3. while termination criteria not met do
4.     cur-sol ← chooseBestOf(neighbourhood(cur-sol) \ tabu-list)
5.     updateTabuList(tabu-list, cur-sol)
6. end
```

2.2.2 Hyper-heuristics

Hyper-heuristics, like metaheuristics, also combine low level heuristic operations into a higher level framework with the objective of improving search efficiency. However, metaheuristics and hyper-heuristics differ in that most metaheuristics search the space X of problem solutions, whereas hyper-heuristics search the space S of heuristic selections [14]. Another difference is that metaheuristics can access problem specific knowledge, while a hyper-heuristic which is subject to the limitations of the *domain barrier* cannot. As a result, metaheuristic approaches tend to be rich in domain knowledge, requiring significant expertise in both the problem and the available heuristics, and so they can be expensive to implement and maintain. Furthermore, a metaheuristic approach developed for one particular problem may not be applicable to other problem domains, or even other instances of the same or similar problems [25]. The domain barrier requires a hyper-heuristic to perform well in the absence of problem specific knowledge and therefore, it is hoped, to be "re-useable" across different problems and problem domains with minimal changes. This places hyper-heuristics at a higher level of abstraction than many metaheuristics.

Hyper-heuristics have been successfully applied to a number of real world optimisation problems including: course timetabling [2] [20] [19] [90] [108] [68], production scheduling [46] [44], 1D bin-

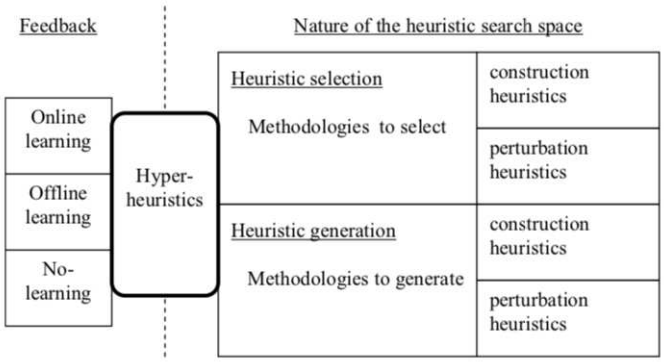


Figure 2.1: A classification of hyper-heuristic approaches, according to two dimensions (i) the nature of the heuristic search space, and (ii) the source of feedback during learning (reproduced from [15]).

packing [99] [79], 2D cutting and packing [112], workforce scheduling [98], constraint satisfaction [111] [85], travelling salesman [66], vehicle routing [50] [51], and optimising water distribution networks [69] [128].

Hyper-heuristics may be classified according to a number of criteria [15] [16], and can be used 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. A selection hyper-heuristic selects and applies a heuristic chosen from a set of low level heuristics. The goal of both types of hyper-heuristic is to improve the search process through learning and/or optimisation. Hyper-heuristics (like metaheuristics) can also be classified by the types of low level heuristic (constructive or perturbative) they employ [15] [16]. Many hyper-heuristics employ learning algorithms to improve optimisation performance, and this learning can be categorised as either *online* or *offline*. Online learning is performed during the execution of a hyper-heuristic and is intended to improve optimisation performance on the problem at hand. In contrast, offline learning is performed on a number of benchmark training problems in order to to improve optimisation performance on new "unseen" test problems.

This classification is illustrated in figure 2.1.

2.2.3 Selection Hyper-heuristics

A selection hyper-heuristic selects and applies heuristics chosen from a given set of low level heuristics [15] [36]. A selection hyper-heuristic consists of

1. a set of low level heuristics,

2. a space of solutions to operate on,

3. a *heuristic selection* strategy, and

4. a *move acceptance* strategy.

A single point hyper-heuristic uses a selection strategy to select a particular low level heuristic s and its parameter set p. Given a solution x, the low level heuristic is used to compute a new solution $x' = s(x, p)$. The acceptance strategy, 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 initial solution x_0 over n iterations or time steps. Multi-point hyper-heuristics have a similar structure, but also require a *solution selection* strategy. The solution selection strategy is used to choose solutions from the population of solutions to operate on, and later, if a new solution is accepted, to choose a solution from the population to be replaced.

Selection hyper-heuristics, like metaheuristics, and hyper-heuristics in general, can be classified according to a number of criteria [14] [36] (see figure 2.2).

Selection hyper-heuristics can be differentiated by their use (or not) of learning algorithms. A learning algorithm for a selection hyper-heuristic optimises the hyper-heuristic's internal structure and parameters in order to refine the selection of heuristics (and their parameters), and in some cases, the acceptance of solutions. Such learning, as noted previously, may be online, offline or a mixture of the two [36].

Selection hyper-heuristics can also be classed as either perturbative or constructive depending on the type of low level heuristics employed. Another distinctive characteristic is how the set of low level heuristics is used or "managed". Specifically, a selection hyper-heuristic may be allowed to manage the whole set of heuristics, a reduced set where poor performing heuristics or heuristics of a certain type are excluded, or an increased set of heuristics, produced by varying the heuristics parametrisation, or combining existing heuristics from the whole set [36].

Many selection hyper-heuristic select and apply single heuristics, one at a time, "without distinction". In contrast, some hyper-heuristics group their low level heuristics into *heuristic classes*, such as mutational, or local search heuristics, based on a heuristic's mode of operation. These hyper-heuristics select low level heuristics from a predefined sequence of heuristic classes, such as mutation followed by

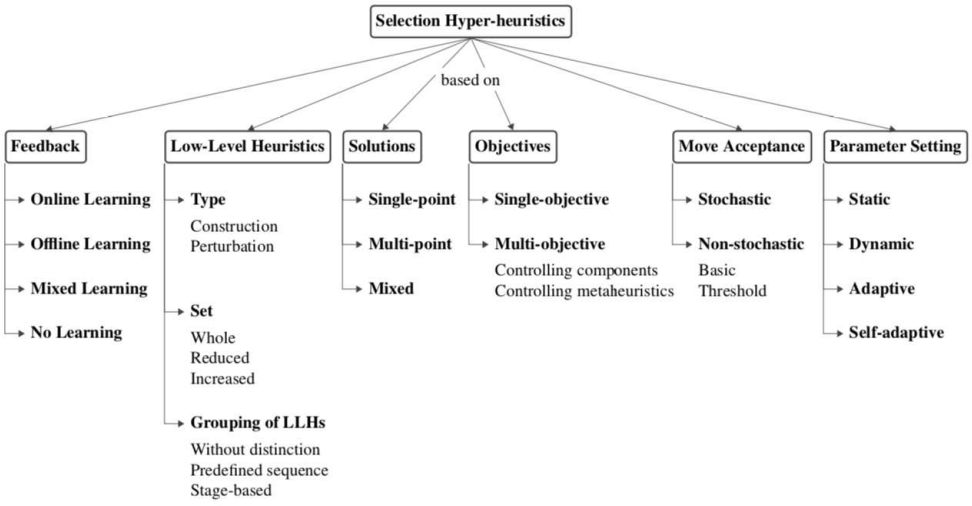


Figure 2.2: An extended classification of selection hyper-heuristics (reproduced from [36]).

local search, during the search process. Some other hyper-heuristic methods operate in a *stage-based* manner, where the subset of low level heuristics to use is decided at distinct stages of the search process. Such approaches can use a fixed number of stages, or switch between stages in an adaptive manner [36].

As has been noted, population based (or multi-point) hyper-heuristics use multiple current solutions as they perform a search, while single point based hyper-heuristics employ a single active current solution.

Acceptance strategies can be classified as stochastic or deterministic. Deterministic acceptance strategies can be further classified into basic methods such as Accept All Moves (AAM), Accept Improving or Equal moves (AIE) and Accept Only Improving moves (AOI), and strategies based on acceptance thresholds [36].

Typically, the low level heuristics, and the selection and acceptance strategies depend on parameters, and these parameters can be classed as static, dynamic, adaptive or self-adaptive. Static parameters are set to some fixed value which is determined prior to the search process, while dynamic parameters are allowed to vary during the search process in some predefined manner. Adaptive parameters can change in a reactive manner due to feedback received during the search process, while self-adaptive parameters settings are actively searched for by the hyper-heuristic [36].

There are numerous examples of selection hyper-heuristics in the literature [14] [16]. For example, RL-GD [86], VNS-TW [60], and AdapHH [82] are three well known selection hyper-heuristics that perform

well across a number of problem domains[1], and are explained in more detail below.

The RL-GD Hyper-heuristic

The reinforcement learning, great deluge hyper-heuristic RL-GD [86] is a single point, perturbative hyper-heuristic combining online reinforcement learning for heuristic selection and the thresholded great deluge move acceptance strategy.

The selection strategy assigns a *utility* value u_i to each low level heuristic i and uses these values to choose heuristics. Specifically, heuristics with a large utility are more likely to be selected, while heuristics with a small utility are less likely to be selected. Reinforcement learning is used to update the utility values at each iteration, depending on the success or failure of the chosen heuristic. An improving move is rewarded, while a worsening move is punished, using a predetermined additive adaptation rate.

The great deluge acceptance strategy uses a dynamic threshold τ that decreases linearly in time to determine an acceptance range for solution fitness or quality, and is defined by

$$\tau_i = f_{opt} + \Delta R\Big(1 - \frac{t}{maxIter}\Big)$$

where *maxIter* is the maximum number of iterations (or total time), t is the current iteration, ΔR is an expected range for the maximum fitness change between the initial fitness f_0, and some "optimal" lower bound on the fitness value f_{opt}. Improving moves are always accepted, while a worsening move is accepted only if the fitness value of the new solution is less than the computed threshold. The pseudocode for the RL-GD hyper-heuristic is shown in listing 8.

The VNS-TW Hyper-heuristic

The VNS-TW hyper-heuristic [60] operates on a population of solutions and is based on the Variable Neighbourhood Search metaheuristic described in Section 2.2.1. It consists of four main stages: perturbation or shaking, local search, environmental selection, and an online learning stage called periodical adjustment (see figure 2.3).

The low level heuristics are separated into two classes: *shaking* `S` and *local search* `L`. The shaking class `S` consists of heuristics such as mutation, ruin and recreate, or other exploratory heuristics that could be used to escape from local optima, while the local search class `L` consists of heuristics that search the local neighbourhood for a better, or non-worsening solution.

Given a random initial solution, a low level heuristic from the shaking class is selected and used to perturb the current solution. A tabu mechanism is employed to prevent the frequent application of poor shaking heuristics. This is followed by the local search stage where several heuristics in the

[1]The hyper-heuristics AdapHH and VNS-TW came first and second respectively in the first international Cross-domain Heuristic Search Challenge (CHeSC 2011) (`http://www.asap.cs.nott.ac.uk/external/chesc2011/index.html`).

Algorithm 8 The reinforcement learning, great-deluge hyper-heuristic (RL-GD) in pseudocode. Reproduced from [86].

```
1. Generate a random solution S_current
2. Initialise utility values u
3. f_current ← f_0 ← quality(S_current)
4. t ← 0
5. while t < maxIter do
6.    i ← selectHeuristic(u)
7.    S_temp ← applyHeuristic(i)
8.    f_temp ← quality(S_temp)
9.    if f_temp < f_current then
10.      u_i ← reward(u_i)
11.      S_current ← S_temp
12.   else
13.      u_i ← punish(u_i)
14.      if f_temp < f_opt + (f_0 − f_opt)(1 − t/maxIter) then
15.         S_current ← S_temp
16.      end
17.   end
18.   t ← t + 1
19. end
```

L class are applied to the solution iteratively, until one of the heuristic's stopping criteria is met. Next, environmental selection determines which old solution in the population should be replaced and which new solution in the population should be selected for the next iteration. The replacement strategy replaces a worse solution x' in the population with a new solution x if $f(x) < f(x')$. If no such solution exists, an alternative worse solution x'' to be replaced is found by 2-tournament selection. The 2-tournament selection mechanism is also used to select the next solution. Finally, a periodical adjustment mechanism is employed to adjust two parameters c and N, each time the algorithm consumes a portion T_b of its overall run-time budget T. The parameter c represents the number of consecutive non-improving moves allowed in the local search stage, while the parameter N dictates the population size. During a periodical adjustment, the parameter c is reset to some maximum number of consecutive non-improving moves, while the population size N is reduced by one when either of the following conditions is met:

1. half of the overall time budget T is consumed, or
2. if a new best solution is found during a time interval T_b.

The periodical adjustment stage allows the hyper-heuristic to adapt to different problem domains.

The AdaptHH Hyper-heuristic

The AdaptHH hyper-heuristic [82] consists of an *adaptive dynamic heuristic set* (ADHS) selection strategy, and *adaptive iteration limited list-based threshold accepting* (AILLA) as an acceptance strategy. In addition, AdaptHH uses *relay hybridisation* to discover effective pairs of heuristics.

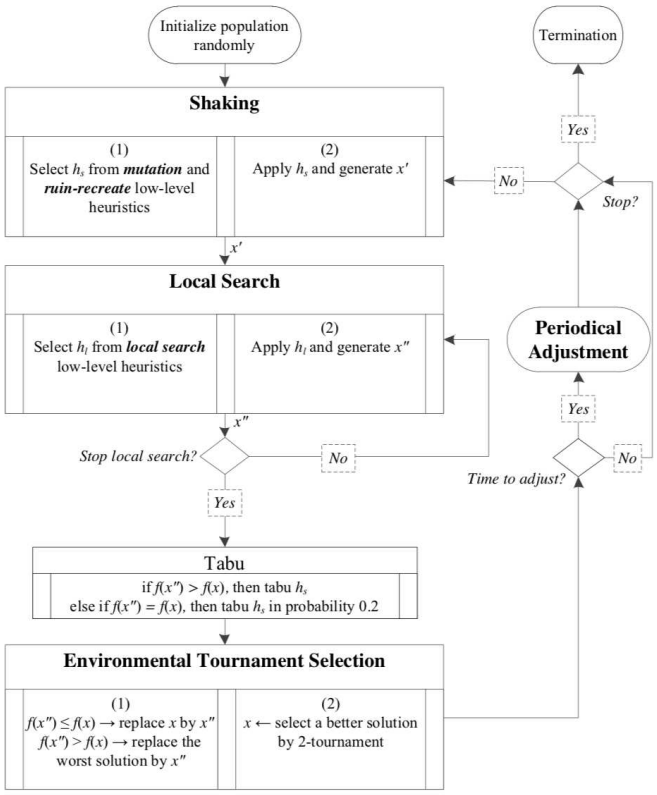


Figure 2.3: A flowchart for the VNS-TW hyper-heuristic (reproduced from [60]).

The adaptive dynamic heuristic set (ADHS) method assesses the performance of each heuristic over a period of time or *phase* with the intention of keeping the best performing heuristics in the working set of heuristics while excluding the others. The phase length is typically set to some predetermined constant. For each low level heuristic i, the performance metric p_i is a choice function based on the weighted sum of a number of simple quality indicators such as solution improvement capability and heuristic execution speed, and is used to decide which heuristics are excluded or *tabued* from the ADHS.

In symbols

$$\begin{aligned} p_i &= w_1\left[(C_{p,best}(i)+1)^2\left(\frac{t_{remain}}{t_{p,spent}(i)}\right)\right]\times b \\ &\quad + w_2\left(\frac{f_{p,imp}(i)}{t_{p,spent}(i)}\right) - w_3\left(\frac{f_{p,wrs}(i)}{t_{p,spent}(i)}\right) \\ &\quad + w_4\left(\frac{f_{imp}(i)}{t_{spent}(i)}\right) - w_5\left(\frac{f_{wrs}(i)}{t_{spent}(i)}\right) \end{aligned}$$

where

$$b = \begin{cases} 1 & \text{if } \sum_i^n C_{p,best}(i) > 0, \\ 0 & \text{otherwise.} \end{cases}$$

The term $C_{p,best}(i)$ denotes the number of new best solutions found, $f_{imp}(i)$ and $f_{wrs}(i)$ show the total objective function improvement and worsening, $f_{\mathrm{p},\ imp}(i)$ and $f_{p,wrs}(i)$ refer to the same measurement for phase p, t_{remain} denotes the remaining search time, and $t_{spent}(i)$ and $t_{p,spent}(i)$ are the execution time of each heuristic i overall, and for each phase p respectively. At each iteration, the non-tabu heuristic with the highest ranking p_i score is selected and applied to the current solution.

The AdaptHH hyper-heuristic also employs a *relay hybridisation* mechanism for approximately 50% of the iterations. Relay hybridisation is intended to discover effective *pairs of heuristics* which can be applied consecutively. For each low level heuristic, AdaptHH maintains a list of past heuristic selections (see figure 2.4).

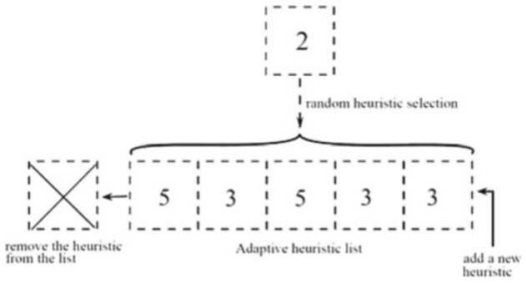


Figure 2.4: AdaptHH's relay hybridisation mechanism (reproduced from [82]).

After selecting and applying a heuristic, if relay hybridisation is enabled, a second heuristic is randomly chosen from that heuristic's list and applied to the current solution. If the resulting solution is a new best solution, then the second heuristic is added to the end of the list, while the first heuristic in the list is discarded so that the list size remains constant.

Some low level heuristics have parameters such as "intensity of mutation" denoting the impact of a perturbation, or "depth of search" referring to the number of consecutive iterations to search for an improvement. A reward-penalty strategy is employed to adapt these parameters during the search process.

The AdaptHH hyper-heuristic uses *adaptive iteration limited list-based threshold accepting* (AILLA) as an acceptance strategy. The mechanism uses the objective function values of the previously found best solutions to dynamically calculate a threshold value which is used to determine solution acceptance.

2.3 A Sequenced-based Approach

Metaheuristics and hyper-heuristics move through a search space by applying sequences of low level heuristics to a set of existing solutions. Although the mechanisms for selecting a solution, and the heuristic operations themselves can vary greatly between algorithms, there exists an identifiable sequence of heuristics for a single solution or population of solutions. For some metaheuristics, the heuristics and therefore the sequence is fixed. For example, a genetic 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 repeated sequence of heuristics that 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 heuristics, in any order it chooses, the issue of context and the notion of sequence becomes important. For example, an exploratory heuristic such as a mutation operator that partially randomises a solution when paired with an exploitative heuristic such as a local search in that order might yield improved solutions. The randomisation moves the algorithm to a new, unexplored region of the solution space and the local search finds the best solution in that neighbourhood. However, the reverse (local search followed by mutation) is likely to be a poor strategy, with the majority, if not all, of the work carried out by the local search being discarded. Moreover, the work of [97] on workforce scheduling demonstrates that a low level heuristic's effectiveness is highly variable during the optimisation process. Some low level heuristics which are ineffective at the start of the search process prove to be highly effective at the end, and *vice versa*, while others heuristics are mainly used at the beginning, middle or end of the process. Clearly, when applying sequences of heuristic selections to optimise a problem, knowing the point at which a particular low level heuristic is most effective could lead to improved optimisation performance. From these two examples it becomes clear that the choice *and* correct ordering of a sequence of heuristics is imperative if search efficacy is to be achieved, and it should be noted that such sequences can be discovered automatically as shown in [67] and [68].

Adopting a sequence-based approach can also reduce the number of objective function evaluations required; in the first example given above, when mutation is followed by local search, there is no need to evaluate the result of the mutation. This is desirable because, for many real world problem domains

and problem sizes, evaluating the objective function is computationally expensive.

It was noted in Section 2.2.3 that some selection hyper-heuristics employ subsets of the available low level heuristics during the optimisation process. These subsets may be heuristic classes which contain heuristics of a particular type, or they could be arbitrary subsets. The construction and performance of such subsets of low level heuristics has received some attention in the literature such as [21], [89], [90], [80], [107], and [74]. However, for subsets, the concept of order or sequence is absent. Adopting a sequence-based approach naturally gives rise to the study of *subsequences* of heuristics, and there are also several examples in the literature of studies concerning (sub-)sequences such as [92], [50], [51], [101], and [85].

2.3.1 Subsets of Heuristics

There are several examples in the literature of empirical research concerning the optimisation performance of subsets of low level heuristics such as [21], [89], [90], [80], and [107], and from a purely theoretical perspective [74].

For example, in [21] the authors compare nine hyper-heuristics on six realistic personnel scheduling problems The hyper-heuristics manage a large collection of 95 low level heuristics using two simple online strategies. The strategies, "warming up approach" and "step-by-step reduction" are used to reduce the set of low level heuristics, leaving only those most likely to produce regular optimisation improvements for a particular problem instance. The warming up approach identifies a specified number of the fittest low level heuristics during some initial "warm up" period specified by a given number of iterations, while the step-by-step reduction approach gradually reduces the set of low level heuristics during a run until some number of the fittest low level heuristics remain. The empirical results demonstrate that the hyper-heuristics using step-by-step reduction consistently out perform the "warming up approach", and those approaches using the whole set of heuristics, producing high quality solutions for the personnel scheduling problems considered.

In [89] subsets of heuristics are randomly selected from a large pool of available heuristics in order to solve constraint satisfaction problems. The choice of an appropriate set of heuristics from the many reported in the constraint satisfaction literature is non-trivial, and the performance of individual heuristics can vary significantly on different problems. Randomly selected subsets of heuristics are evaluated on a number of benchmark problems, and an offline learning algorithm is used to determine weights for the elements of a subset. These subsets of heuristics are then used to parameterise a hyper-heuristic where the weights guide the selection process. Each low level heuristic computes a *strength* that indicates the "degree of support" for a choice of variable or value, and heuristics are selected by voting; the heuristic with the greatest weighted strength is chosen. The experimental results demonstrate that randomly chosen subsets of heuristics can help in the identification of an effective mixture of low level heuristics, and that offline learning with the small, random subsets of

heuristics performs significantly better than learning with a large set.

In [90] the authors search a space of "heuristic combinations" to solve examination timetabling problems. A heuristic combination consists of $p \geq 2$ primary heuristics, one of which is designated the priority heuristic, and in some cases, a secondary heuristic. Each heuristic computes a specific value for an exam such as the number of students or the number of conflicts. The heuristics are applied simultaneously to pairs of exams, and the p heuristic values are used to compute a Pareto comparison. The priority heuristic is used resolve conflict situations, while the secondary heuristic, when present, is used as a tie breaker. The authors evaluate four combinations consisting of two or three low level heuristics to order a set of exams for scheduling. The quality of the timetables generated by this approach are close to the best quality timetables cited in the literature.

In [80] two heuristic selection strategies and seven acceptance strategies are used to construct 14 distinct hyper-heuristics. These hyper-heuristics are parameterised with nine different heuristic sets, and are evaluated on seven patient admission scheduling problems. The heuristic subsets are generated by varying the choice of heuristics and their parameterisation resulting in nine sets of different sizes, with different speed and improvement capabilities. The experimental results demonstrate that different subsets of low level heuristics produce very different optimisation performances across a number of hyper-heuristic designs and that the "best" hyper-heuristic can change based on the heuristic set used and the execution time limits employed.

In [107] the authors present an empirical methodology for determining the most effective subset of low level heuristics from a given larger set. Specifically, 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. A subset of low level heuristics, selected using the methodology, is used to parameterise a hyper-heuristic which is tested on 24 course timetabling problems and 10 vehicle routing problems. The parameterised hyper-heuristic outperforms state-of-the-art hyper-heuristics and competes with problem-specific methods in course timetabling, producing five new best-known solutions.

In [74] the authors present a theoretical analysis of a selection hyper-heuristic's expected run-time behaviour with various selection and acceptance strategies. By using *fitness based partitions* which is a well known method for the runtime analysis of randomised search heuristics, they establish upper bounds on a hyper-heuristic's expected run-time. The results demonstrate that "mixing" two heuristics can lead to exponentially faster search than an individual (deterministically chosen) heuristic on some problems. The expected runtime (exponential vs polynomial) depends critically on a *mixing parameter* p, which is the probability of choosing one heuristic or the other, and that the right choice of p is problem-dependant.

In each case, the authors demonstrate (unsurprisingly) that different choices of heuristic subset can lead to large differences in optimisation performance. Furthermore, it is possible to identify small subsets of heuristics that can work well together, and which lead to improvements in optimisation

performance for the problems of a particular domain.

2.3.2 Subsequences of Heuristics

There are also several examples in the literature of empirical research concerning sequences of heuristics where the concept of order is explicit such as [92], [50], [51], [101], and [85].

In [92] the authors present a hybrid variable neighbourhood search hyper-heuristic (HVNS) for solving exam timetabling problems. The HVNS hyper-heuristic searches for lists of low level heuristics that are used to construct solutions, by applying each heuristic in a list, one after another. Two neighbourhood structures are studied: "single flipped" where a small number of heuristics in the list are randomly changed, and "block flipped" where a small number of consecutive heuristics in the list are randomly changed. The overall results demonstrate that while Iterated Local Search outperforms tabu search, and steepest descent search, the proposed hybrid method HVNS produces four of the best results reported out of a set of 11 test exam timetabling problems.

In [50] and [51] variable length sequences of heuristics are used to solve the constrained vehicle routing problem. The sequences are constructed from heuristic pairs which are applied sequentially to some initial solution in order to derive a full solution to the problem. The heuristic pairs consist of a constructive heuristic followed by an exploitative perturbative heuristic. The hyper-heuristic can add, delete and replace the heuristic pairs in a sequence, and these sequence modification operators are applied with equal probability at each step of the optimisation process. The sequence modification operations allow the hyper-heuristic to discover new, improved sequences. By using this "collaborative" approach, the authors of [50] are able to find effective sequences of low level heuristic pairs that produce results that are comparable with the best reported results in the vehicle routing optimisation literature.

In [85] the authors construct and evaluate sequences of heuristics in order to solve constraint satisfaction problems. A hyper-heuristic is constructed from two low level heuristics, and represented as a binary strings where a '0' denotes the first heuristic and a '1' denotes the second. Each possible permutation of the heuristics in a sequence is evaluated on a set of benchmark problems, and the best performing permutation is selected. This permutation is then applied repeatedly to optimise unseen test problems. The experimental results demonstrate that by combining some heuristics it is possible to achieve a better optimisation performance than applying the heuristics in isolation. However, the results also show that not all heuristic permutations work well, and that it is not sufficient to simply combine the most effective heuristics. The chosen heuristics must exhibit some extra joint property in order to achieve "heuristic synergy" [119]. Although the methodology presented can be applied to larger sets of heuristics, in practice evaluating every permutation of a sequence is only feasible for relatively short sequences employing a small numbers of heuristics.

In [101] the authors present a Monte Carlo tree search hyper-heuristic (MCTS-HH). Here the search space of low level heuristics is modelled as a tree, and a Monte Carlo tree search is employed to search

this tree in order to identify the best sequence of low level heuristics to be applied to the current solution. The MCTS-HH hyper-heuristic uses a population of solutions, and a number of different population updating rules. The generality of the proposed approach is evaluated on the six domains of the HyFlex benchmark problems [84]. The results demonstrate that the proposed hyper-heuristic performs well over all six domains and obtains competitive results when compared to the best known results that have been presented in the literature, with MCTS-HH producing 17 new best solutions out of 30 instances.

These examples demonstrate that the choice of the low level heuristics and the order in which they are applied is important when improving optimisation performance. In this thesis, subsequences of heuristic selections and their objective function values are analysed statistically in order to distinguish between *effective* subsequences which tend to reduce the objective function value, and *disruptive* subsequences which tend to increase the objective function value, in a manner that does not depend on the number or type of heuristics or the length of the subsequences.

2.4 Hyper-Heuristic Learning

A learning algorithm for a selection hyper-heuristic optimises the hyper-heuristic's internal structure and parameters in order to refine the selection of heuristics (and their parameters), and in some cases, the acceptance of solutions. Such learning, may be online, offline or a mixture of the two [36].

Online learning is based on the low level heuristic selections and resulting objective function values computed during the execution of a hyper-heuristic, and is intended to improve optimisation performance on the problem at hand. Many algorithms have been proposed and successfully employed for online learning such as genetic algorithms [44], local search algorithms [56] [50], choice functions [25], fuzzy systems [4], meta-heuristics [2] and [19], reinforcement learning [86], tabu search [18] [60] [82], and adaptive selection probabilities [108]. In contrast, offline learning is performed on a database of low level heuristic sequences and objective function values computed by a selection 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 [99], case based reasoning [20], messy genetic algorithms [111], and the *ILSParam* parameter tuning algorithm, first introduced in [62] and on based on the ILS metaheuristic, which is employed in [108].

In [99] 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 [20] 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 [111] 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 [108] offline and online learning are combined to optimise an Iterated Local Search (ILS) hyper-heuristic for the course timetabling problem. ILS uses 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[2], benchmark test set, producing a new best-known solution.

These examples demonstrate that offline learning is able to improve hyper-heuristic performance by learning from, and generalising over the problems of a domain. However, in each case, the methodologies employed differ markedly and are dictated by the choice of learning algorithm and/or the choice of problem state space representation. Furthermore, these learning algorithms, and those employed by most other selection hyper-heuristics are trained to produce *single heuristic selections*, or heuristic pairs like [25] or AdaptHH in [82]. The framework for offline learning presented here is based on arbitrary subsequences of heuristics and does not depend on the choice of learning algorithm or the problem space representation.

Offline learning can be further categorised as either *intra-domain* or *cross-domain*. In intra-domain learning, the benchmark training problems and the target optimisation problem are drawn from the same problem domain. This simplifies the learning task considerably as the low level heuristics are identical for each problem and so the heuristics, and the effective subsequences constructed from them, will have similar statistical characteristics across the problems of the domain. In cross-domain learning the benchmark training problems and the target problem can be drawn from different problem domains. In this case, the different domains can have different low level heuristics, and so the effective subsequences obtained by optimising the benchmark problems consist of heuristic classes, and these heuristic classes may have very different statistical characteristics in each domain. As a result, cross-

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

domain learning is generally a much more difficult problem than intra-domain learning. Although there are many studies concerning cross-domain online learning such as [37] [84] [31], there are, to the best of the author's knowledge, no examples of offline cross-domain learning in the current literature.

Another type of learning of particular interest is *scalable learning*, where a model trained for one task is re-used as the starting point of a model for a second task. In this study, the first task is a simple problem that is smaller, and therefore more computationally tractable, than the second task. The idea is to offline train a hyper-heuristic on small benchmark problems, and then transfer this learning to a larger, more computationally expensive target problem in order to improve optimisation performance.

2.5 A Sequence-based Selection Hyper-heuristic

The statistical framework presented in this thesis is evaluated on the HyFlex benchmark problems, and the results are compared with those produced by the SSHH hyper-heuristic [67] [68]. This section contains a detailed presentation of SSHH and the associated algorithms and concepts used for learning, and later, for the visualisation of learning.

The SSHH hyper-heuristic is a sequenced-based selection hyper-heuristic with online learning. It has been tested on the HyFlex[3] problems [84] 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 [82] on these problem instances.

The SSHH hyper-heuristic uses a hidden Markov model (HMM) to generate sequences of heuristic selections, their parameters, and acceptance check decisions. The HMM consists of a set of hidden states, and four probability matrices:

1. a state transition matrix to determine the probability of moving from one hidden state to another,
2. a low level heuristic emission matrix to determine which low level heuristic to select,
3. a parameter emission matrix to determine the parameter for a low level heuristic, and
4. an acceptance strategy emission matrix to determine when a solution should be checked for acceptance.

In the absence of *a priori* knowledge regarding a particular problem the number of hidden states is set to be the number of low level heuristics in the domain, and the state transition, the parameter and acceptance strategy matrices are set to be equiprobable. The low level heuristic emission matrix is set to the identity matrix. This ensures that, initially, each equiprobable hidden state emits a single low level heuristic together with an equiprobable choice of heuristic parameter and acceptance check decision.

[3]The HyFlex Cross-domain Heuristic Search Challenge (CHeSC 2011) library chesc.jar is used throughout this study (see http://www.asap.cs.nott.ac.uk/chesc2011/).

At each iteration of the optimisation process, the SSHH hyper-heuristic moves from its current state to a new state according to the probabilities of the transition matrix. Once a new state has been chosen, the emission probability matrices are used to determine a low level heuristic, its parameters, and whether to check for acceptance or not. The selected parametrised low level heuristic is applied to the current solution in order to derive a new solution. If the acceptance check decision is true, the objective function is evaluated on the new solution, and a decision is made whether to accept it or not. Specifically, a new solution is accepted if it improves on the current solution or is within 5% of the current best solution; otherwise it is rejected.

Hidden Markov Models

Hidden Markov models (HMM) are stochastic processes that can produce and analyse sequences of symbols. Such models were originally applied to speech recognition problems [95].

A hidden Markov model consists of five components:

1. a set of n *hidden states* $S = \{S_1, S_2, \ldots S_n\}$,
2. an alphabet of m output *symbols* $V = \{v_1, v_2, \ldots, v_m\}$,
3. an initial state vector $\pi = (\pi_1, \pi_2, \ldots \pi_n)$ where π_i is the probability of being in state S_i at time $t = 0$,
4. an $n \times n$ *state transition* matrix $A = \{a_{i,j}\}$ where $a_{i,j}$ is the the probability of moving from state S_i to state S_j, and
5. an $n \times m$ *emission* probability matrix $B = \{b_{j,k}\}$ where $b_{j,k}$ is the probability of producing symbol v_k when in state S_j.

At time $t = 0$, a HMM is assigned some initial hidden state S_i according to the probabilities $\pi = (\pi_1, \ldots \pi_n)$. For each subsequent time step, a new state S_j is determined using the state transition probabilities $a_i = \{a_{i,1}, \ldots a_{i,n}\}$ associated with the current state S_i. In addition, the HMM emits or outputs a symbol v_k according to the emission probabilities $b_i = \{b_{i,1}, \ldots b_{i,m}\}$ which are also associated with state S_i. Thus, over time, the model transitions from one hidden state to another while producing a sequence of symbols drawn from V.

It should be noted that the probability $a_{i,j}$ of transitioning from state S_i to state S_j and the probability $b_{i,k}$ of emitting symbol v_k at some time t depend only on the current state S_i and that previous states and emissions are irrelevant. This property is called the *Markov property*.

Figure 2.5 shows the general architecture of a HMM with three hidden states.

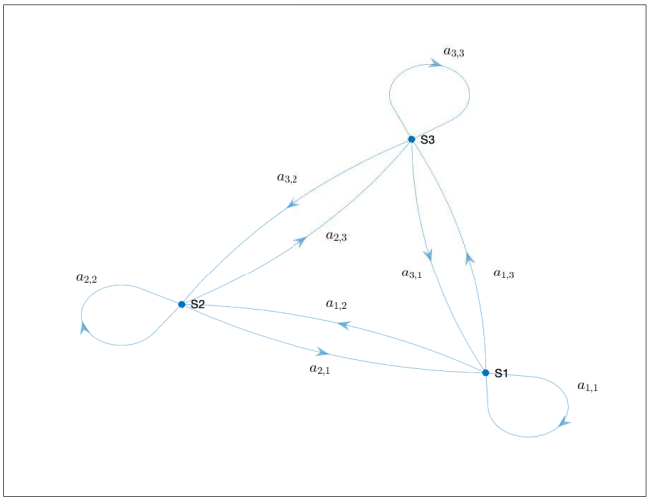


Figure 2.5: A hidden Markov model with three states. The nodes **S1**, **S2**, and **S3** represent the hidden states while the arcs denote state transitions. An arc weight $a_{i,j}$ is the probability of transitioning from state **Si** to state **Sj**. The model's emissions have been omitted for clarity.

Online Learning

The SSHH hyper-heuristic employs online learning. During optimisation, SSHH keeps a history of the heuristic selections, parameters, and acceptance checks produced by the HMM. If, following an acceptance check, a new, best solution is found, the online learning algorithm steps through the history, increasing the probabilities of the accepted state transitions and emissions that led to the new minima. Thus the probability that the HMM produces the sequence of heuristic selections and emissions contained in the history is now higher. After the acceptance check, the history is erased and the optimisation process is resumed.

Offline Learning

A HMM can be trained offline using the Baum-Welch learning algorithm [95] [9]. The Baum-Welch algorithm calculates the maximum likelihood estimates of a HMM's parameters $\lambda = (\pi, A, B)$ for a given sequence of observations of symbols drawn from V. In this thesis, the observations are sequences of low level heuristics chosen from an offline learning database.

Adopting the notation of [95] (Section III C), consider a HMM consisting of n states $S_1, \ldots, S_n$,

and a sequence of training observation $O = O_1, O_2, \ldots, O_T$. Let the model's parameterisation λ be specified by an initial state distribution π, a state transition matrix A, and an emission probability matrix B. The Baum-Welch learning algorithm estimates the model parameters $\lambda = (\pi, A, B)$ that maximise (locally) the probability $P(O|\lambda)$ of the model producing the observation sequence O.

In what follows, the value $a_{i,j} \in A$ is the probability that the model passes from state S_i to state S_j, the value $b_j(O_t) \in B$ is the probability that the model emits the observation O_t while in state S_j at time t, and the value π_i is the probability that the model begins its computation in state S_i.

The probability $P(O|\lambda)$ can be calculated efficiently using the *forward-backward* procedure. The *forward variable*[4] $\alpha_t(i)$ is defined recursively for each state by

$$\begin{aligned} \alpha_1(i) &= \pi_i \, b_i(O_1) \\ \alpha_{t+1}(j) &= \Big[\sum_i^n \alpha_t(i) \, a_{i,j} \Big] \, b_j(O_{t+1}) \end{aligned}$$

for $t = 1, \ldots, T-1$. Similarly, the *backward variable* $\beta_t(i)$ is also defined recursively for each state by

$$\begin{aligned} \beta_T(i) &= 1 \\ \beta_t(i) &= \sum_j^n a_{i,j} \, b_j(O_{t+1}) \, \beta_{t+1}(j) \end{aligned}$$

for $t = T-1, T-2, \ldots, 1$.

The function $\xi_t(i,j)$ is the probability of being in state S_i at time t, and state S_j at time $t+1$, given the model parameters λ and the observation sequence O, and is defined by

$$\xi_t(i,j) = \frac{\alpha_t(i) \, a_{i,j} \, b_j(O_{t+1}) \, \beta_{t+1}(j)}{\sum_i^n \sum_j^n \alpha_t(i) \, a_{i,j} \, b_j(O_{t+1}) \, \beta_{t+1}(j)}$$

for $t = 1, \ldots, T-1$. The function $\gamma_t(i)$ is the probability of being in state S_i at time t given the model parameters λ and the observation sequence O, and is defined by

$$\gamma_t(i) = \frac{\alpha_t(i) \, \beta_t(i)}{\sum_j^n \alpha_t(j) \, \beta_t(j)}$$

for $t = 1, \ldots, T$.

Thus, the Baum-Welch algorithm is specified by the three update equations:

$$\begin{aligned} \overline{\pi}_i &= \gamma_1(i) \\ \overline{a}_{i,j} &= \frac{\sum_t^{T-1} \xi_t(i,j)}{\sum_t^{T-1} \gamma_t(i)} \\ \overline{b}_{j,k} &= \frac{\sum_t^{T} \delta(O_t, v_k) \gamma_t(j)}{\sum_t^{T} \gamma_t(j)} \end{aligned}$$

[4] It should be noted that $P(O|\lambda) = \sum_i^n \alpha_T(i)$.

where $\delta(O_t, v_k)$ is the Kronecker delta function

$$\delta(O_t, v_k) = \begin{cases} 1 & \text{if } O_t = v_k, \\ 0 & \text{otherwise} \end{cases}$$

and the term $\delta(O_t, v_k)\gamma_t(j)$ is the probability of being in state S_j while observing the symbol $O_t = v_k$.

Providing the sequence of observations is long enough to learn from, applying these update equations iteratively to an initial model λ leads to an improvement in the probability of O being produced by the model. In practice, when implementing the Baum-Welch algorithm the update equations are modified slightly to prevent underflow issues when multiplying probabilities and to deal with multiple sequences of observations ([95], Section V-A and Section V-B).

Probability Vectors

A *probability vector* is any vector with non-negative components that sum to 1. The initial state vector π, and the state transition and emission vectors a_i and b_i are probability vectors. The mean of a probability vector with n components is $1/n$. The probabilistic length of such a vector is

$$\sqrt{n\sigma^2 + \frac{1}{n}}$$

where σ^2 is the variance of the elements of the probability vector.

The length of a probability vector measures uncertainty; the shortest vector corresponds to maximum uncertainty, while the longest vector corresponds to minimum uncertainty (or equivalently maximum certainty). The shortest probability vector has the value $1/n$ for each component, and has a length of $1/\sqrt{n}$. The longest probability vector has the value 1 for a single component, and 0 for all the others, and has a length of 1.

The Kullback-Leibler Distance

Mathematically, a HMM encodes a stationary distribution defined on the symbols of V. The non-symmetric Kullback-Leibler divergence D is a measure of how one probability distribution differs from another, and it can be used to define a "distance" between HMMs [65].

The symmetric Kullback-Leibler distance $D_{\mathrm{KL}}(\lambda_0, \lambda)$ between two HMMs λ_0 and λ is defined by

$$D_{\mathrm{KL}}(\lambda_0, \lambda) = \frac{1}{2}\big[D(\lambda_0, \lambda) + D(\lambda, \lambda_0)\big]$$

where

$$D(\lambda_0, \lambda) = \lim_{T\to\infty} \frac{1}{T}\big[\log P(O_T|\lambda_0) - \log P(O_T|\lambda)\big].$$

It should be noted that the Kullback-Leibler distance D_{KL} is not a metric in the technical sense as it does not satisfy the triangle inequality [77].

In this study, the distance D_{KL} is estimated over a number $i = 1, \ldots, S$ of Monte Carlo trials. Specifically, in order to estimate $D(\lambda_0, \lambda)$, the first HMM λ_0 is used to generate $S = 1000$ sequences of length $T = 1000$. For each such sequence O_T^i, the probabilities that the HMMs λ_0 and λ generate this sequence, denoted $P(O_T^i|\lambda_0)$ and $P(O_T^i|\lambda)$ respectively, are calculated using the forward-backward algorithm [95]. As the sequences were generated by λ_0, the difference $P(O_T^i|\lambda_0) - P(O_T^i|\lambda)$ is non-negative. When S and T are large

$$D(\lambda_0, \lambda) \approx \frac{1}{TS} \sum_{i=1}^{S} \left[\log P(O_T^i|\lambda_0) - \log P(O_T^i|\lambda) \right].$$

This process is then repeated for the second HMM λ in order to estimate $D(\lambda, \lambda_0)$ and thus calculate D_{KL}.

The Kullback-Leibler distance D_{KL} can be used to quantify the "information gain" before and after Baum-Welch learning.

2.6 The No Free Lunch Theorems

Despite the interest in general purpose optimisation algorithms there is no known algorithm that offers the best overall performance across different problem instances and problem domains. In fact, it has been demonstrated that under some fairly general mathematical conditions no such algorithm can exist. Specifically, the *No Free Lunch* (NFL) theorems presented in [121] for optimisation algorithms states that

> "if an algorithm does particularly well on average for one class of problems then it must do worse on average over the remaining problems."

The NFL theorems imply that different optimisation algorithms will exhibit identical *average* performance over a given set of problems and objective functions. Although the NFL theorems concern algorithms that search some given problem space X, similar arguments can be also be applied to algorithms that search spaces of heuristics S [91]. Mathematically, the NFL theorems apply to finite problem spaces where the set of objective functions is *closed under permutation* [103]. Given a set of objective functions $\mathcal{F}$, each objective function assigns objective values to the solutions in the problem space X. A permutation of an objective function is simply a rearrangement of the objective values assigned to the solutions $x \in X$. If a set of objective functions for a finite problem space is closed under permutation, then the expected performance of any search algorithm over a set of problems defined on X is some constant c and is independent of the chosen algorithm and performance measure [103]. In symbols, if $\mathcal{F}$ is a set of objective functions that is closed under permutation, then

$$\sum_{f \in \mathcal{F}} P(f, a_1) = \sum_{f \in \mathcal{F}} P(f, a_2) = c$$

for any pair of (non-resampling) search algorithms a_1 and a_2 and for any performance measure P. Furthermore, it is also the case that if two arbitrary algorithms have identical average performance over a set of objective functions then that set of functions is closed under permutation [91].

In these cases, a hyper-heuristic search based on a low level heuristic's performance "makes no sense" as each low level heuristic is, on average, as effective as every other [91]. However, in practice, a low level heuristics's performance may vary considerably during the optimisation process of particular problem instances, or over a number of identifiable problem domains, and such variations can be exploited by a hyper-heuristic search algorithm. Furthermore, the sets of objective functions that are closed under permutation, and hence, over which the NFL theorems apply, represent a small fraction of the set of all possible sets of functions on a particular domain. As a result, the problem sets where the NFL theorems apply to hyper-heuristics are quite rare, although it should be noted that there is no guarantee that NFL will not be applicable for a particular set of problems [91].

Chapter 3

A Framework for Offline Learning

This chapter presents a novel methodological framework for the construction and analysis of offline learning databases.

A simple selection hyper-heuristic is repeatedly executed on a well known set of benchmark optimisation problems, in order to generate a database of sequences of heuristic selections, heuristic parameters, and objective function values. The benchmark problems can be viewed as a set of training instances, that could be expected to generalise to the process of solving unseen test instances.

The sequences of heuristic selections and objective function values are analysed using a statistical framework which employs the mathematical concept of logarithmic returns. The framework is used to identify and quantify the performance and behaviour of individual low level heuristics, subsets of heuristics, and subsequences of heuristics. The framework is also used to analyse the interactions that occur between the heuristics of a subset or subsequence, and the measurement of hyper-heuristic performance in general.

Although the computational cost of constructing an offline learning database is significant, the objective is not to compare optimisation performance using equivalent computational resources, but rather it is to investigate what can be learned from a such a database.

The structure of this chapter is as follows. In Section 3.1, the set of benchmark problems is described. Section 3.2 contains the methodology used to generate an offline learning database of heuristic selections. In Section 3.3, a general framework for analysing an offline learning database is introduced, and, in Section 3.3.2, the framework is applied to the quantification and analysis of individual low level heuristic behaviour. This framework is extended to subsets, in Section 3.3.3, and then subsequences of heuristic selections, in Section 3.3.4. In Section 3.4, the framework is also applied to the measurement of hyper-heuristic performance. Finally, in Section 3.5, the statistical test used to validate results throughout this thesis is presented.

3.1 The HyFlex Problems

Offline learning requires a set of benchmark problems to train and test on. This thesis uses the Hyper-heuristics Flexible framework (or HyFlex[1]) problems [84]. HyFlex is a well known set of discrete optimisation problems that has been used in a number of studies (see for example [118], [37], [23], [82], [38], [5], [101], [67], [31], [123], [124], [126]), and is an implementation of the six computationally hard problem domains shown in table 3.1. The use of a preexisting benchmark set significantly reduces the time required to develop and implement computational experiments.

Table 3.1: The HyFlex problem domains (with available references).

Dom.	Description	Ref.
BP	1D Bin Packing	[64]
PFS	Permutation Flow Shop	[115]
SAT	Max Boolean Satisfiability	[63]
VRP	Vehicle Routing	
TSP	Travelling Salesman	
PS	Personnel Scheduling	[29]

HyFlex is designed to enable the development, testing, and comparison of iterative, general-purpose heuristic search algorithms (such as hyper-heuristics). The implementation decomposes a heuristic search problem into two parts: a general-purpose part consisting of the algorithm or hyper-heuristic itself, and a problem-specific part provided by the HyFlex framework. In the hyper-heuristics literature, this idea is also referred to as the *domain barrier* between the problem specific heuristics and the hyper-heuristic [25]. HyFlex maintains a population of solutions in the problem domain layer, and provides a rich variety of problem specific heuristics and search operators [84].

Each HyFlex problem domain contains a number of distinct problem instances of varying complexity. HyFlex hides all problem specific information such as the solution representations, solution construction, and the low level heuristic implementations. The low level heuristics can be divided into four general *classes of heuristic* $\{\,\mathtt{M}, \mathtt{C}, \mathtt{R}, \mathtt{L}\,\}$

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

[1]The HyFlex Cross-domain Heuristic Search Challenge (CHeSC 2011) library is used in this thesis (`http://www.asap.cs.nott.ac.uk/chesc2011/`).

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 3.2). The parameters for the heuristics are drawn from the set $\{0, 1, 2, \ldots, 10\}$.

Table 3.2: The low level heuristics for each domain.

Dom.	Heuristics
BP	$\mathtt{M}_0, \mathtt{R}_1, \mathtt{R}_2, \mathtt{M}_3, \mathtt{L}_4, \mathtt{M}_5, \mathtt{L}_6, \mathtt{C}_7$
PFS	$\mathtt{M}_0, \mathtt{M}_1, \mathtt{M}_2, \mathtt{M}_3, \mathtt{M}_4, \mathtt{R}_5, \mathtt{R}_6, \mathtt{L}_7, \mathtt{L}_8, \mathtt{L}_9, \mathtt{L}_{10}, \mathtt{C}_{11}, \mathtt{C}_{12}, \mathtt{C}_{13}, \mathtt{C}_{14}$
SAT	$\mathtt{M}_0, \mathtt{M}_1, \mathtt{M}_2, \mathtt{M}_3, \mathtt{M}_4, \mathtt{M}_5, \mathtt{R}_6, \mathtt{L}_7, \mathtt{L}_8, \mathtt{C}_9, \mathtt{C}_{10}$
VRP	$\mathtt{M}_0, \mathtt{M}_1, \mathtt{R}_2, \mathtt{R}_3, \mathtt{L}_4, \mathtt{C}_5, \mathtt{C}_6, \mathtt{M}_7, \mathtt{L}_8, \mathtt{L}_9$
TSP	$\mathtt{M}_0, \mathtt{M}_1, \mathtt{M}_2, \mathtt{M}_3, \mathtt{M}_4, \mathtt{R}_5, \mathtt{L}_6, \mathtt{L}_7, \mathtt{L}_8, \mathtt{C}_9, \mathtt{C}_{10}, \mathtt{C}_{11}, \mathtt{C}_{12}$
PS	$\mathtt{L}_0, \mathtt{L}_1, \mathtt{L}_2, \mathtt{L}_3, \mathtt{L}_4, \mathtt{R}_5, \mathtt{R}_6, \mathtt{R}_7, \mathtt{C}_8, \mathtt{C}_9, \mathtt{C}_{10}, \mathtt{M}_{11}$

It should be emphasised that the heuristic $\mathtt{M}_0$ in the BP domain is an entirely different heuristic to $\mathtt{M}_0$ 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 greedily search the surrounding space.

The local search heuristics differ from the other classes in two respects. Firstly, they are required to produce a non-worsening solution, and secondly, they may perform one or more objective function evaluations. The six HyFlex domains 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 a "depth of search" parameter.

For each problem domain there is also a real valued, non-negative, *objective function* denoted f that is to be minimised. 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.

3.2 Offline Learning Databases

A simple, unbiased, random, single selection hyper-heuristic, denoted DBGen, is repeatedly executed on HyFlex problem instances, for a number of iterations n, in order to generate sequences of

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

These sequences are used to construct three separate offline learning databases, denoted DB_1, DB_2, and DB_3. Specifically, DBGen is executed

1. 40 times for 150 iterations on 10 problems instance in the BP, PFS, SAT and PS domains (DB_1),
2. 400 times for 150 iterations on 10 problem instances in all six domains (DB_2), and
3. 10 times for 10 minutes on 10 problem instances in all six domains (DB_3).

The database DB_1 is the smallest of the three, and is used for preliminary experiments. For DB_1, each DBGen run is seeded by a single number defined by

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

where $p = 0, \ldots, 39$ is the problem index, and $r = 0, \ldots, 39$ is the run index. The seed is used to initialise each HyFlex problem instance, which uniquely determines a run's initial solution x_0, and is also used to initialise DBGen's random number generator. The DBGen hyper-heuristic is then executed 40 times on 10 problem instances in the BP, PFS, SAT and PS domains, for 150 iterations; a total of 1600 runs and 240,000 heuristic selections, heuristic parameters, and objective function values. The number of 40 runs was chosen so as to ensure that robust statistics could be calculated for each problem instance.

The database DB_2 is essentially a larger version of DB_1. For DB_2, each DBGen run is parameterised by two numbers

$$\mathit{hseed} \in \{1, \ldots, 10\} \text{ and } \mathit{pseed} \in \{1, \ldots, 40\}.$$

The *hseed* is used to seed the pseudorandom number generator used by the DBGen hyper-heuristic, while the *pseed* is used to randomly initialise a HyFlex problem instance. For each of the 10 *hseeds*, the DBGen hyper-heuristic is run 40 times, once for each *pseed*, on 10 problem instances in each HyFlex domain, for 150 iterations; a total of 24,000 runs and 3,600,000 heuristic selections, heuristic parameters, and objective function values.

To put the size of these databases into perspective, as the set of low level heuristic classes $\{\mathtt{M}, \mathtt{C}, \mathtt{R}, \mathtt{L}\}$ contains four symbols there are $4^{150} \approx 2.0370 \times 10^{90}$ unique sequences of length 150.

The number of iterations or run length of 150 was chosen for computational feasibility. It was used for each domain, regardless of low level heuristic execution times, so that the sequences generated for each domain contained the same "amount of information", ensuring that no domain has an "informational advantage" over the others. However, for some domains, the statistical information extracted from such relatively short runs may not capture behaviour that occurs over longer optimisation periods. With this in mind, for database DB_3, each DBGen run lasts for ten minutes, and is parameterised by a single number

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

where $p = 0, \ldots, 59$ is the problem index, and $r = 0, \ldots, 9$ is the run index. Such runs can produce 10,000's of heuristic selections (see table 3.3).

Table 3.3: The average number of heuristics selections in DB_3 produced by the DBGen hyper-heuristic for each domain.

Dom.	BP	PFS	SAT	VRP	TSP	PS
Length	611,208	95,898	228,254	1,440,252	246,833	659

The number of 10 runs was chosen because beyond this point the computational time and space costs become prohibitive. The DB_3 database is used to cross check results from DB_1 and DB_2 in order to ensure that any observed behaviour also occurs over longer runs.

The structure of the resulting databases is illustrated in figure 3.1.

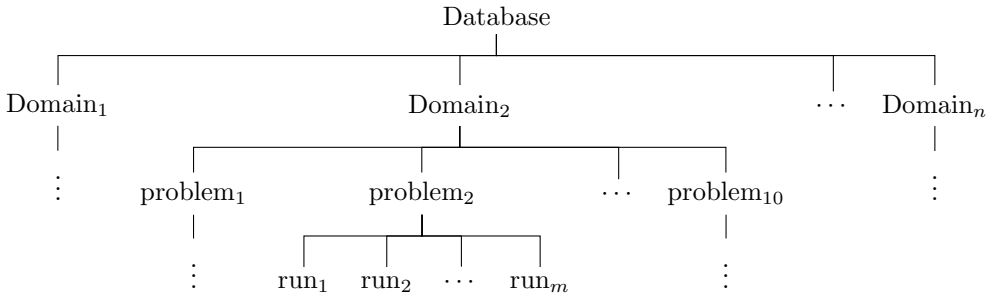


Figure 3.1: The conceptual structure of the three offline learning database's DB_1, DB_2, and DB_3.

The DBGen algorithm [123] [124] [126] is shown in listing 9. The function *ranInt()* (lines 12 and 19) returns a uniformly distributed pseudorandom number in the set $\{1, \ldots, \texttt{POOLSIZE}\}$, while the function *ranFloat()* (line 15) returns a uniformly distributed pseudorandom number in the interval $(0, 1)$. The function *selectHeuristic()* (line 11) selects a single heuristic class at random from the set $\{\,\texttt{M}, \texttt{L}, \texttt{R}, \texttt{C}\,\}$. The function *apply()* (line 13) 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 a crossover solution which is randomly selected from the crossover pool (lines 5–6 and lines 12–13). An objective function evaluation (line 14) and an acceptance check (line 15) are then performed. If a new solution's objective value is less than the current solution's objective value or *ranFloat()* < 0.5 then it is accepted, and also stored, at random, in the crossover pool (lines 19–21). Otherwise the new solution is rejected. The random term allows new solutions to be accepted regardless of their objective function for 50% of the cases. 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 [38]. The DBGen crossover mechanism (lines 12-15) is taken from the crossover management scheme employed by the AdapHH hyper-heuristic [38]. This crossover mechanism is also used by SSHH.

For the construction of DB_1 the crossover mechanism uses a population of one solution, that is `POOLSIZE` $= 1$, and for the construction of DB_2 a population of five potential crossover solutions including the current best solution.

Algorithm 9 The DBGen hyper-heuristic in pseudocode.

```
1.  ITERATIONS ← MAX_ITER
2.  new-sol ← initialiseSolution()
3.  new-obj ← f(new-sol)
4.  for i = 1 to POOLSIZE do
5.     cross-sol[i] ← initialiseSolution()
6.     cross-obj[i] ← f(cross-sol[i])
7.  end
8.  while (ITERATIONS > 0) do
9.     cur-sol ← new-sol
10.    cur-obj ← new-obj
11.    Heuristic h ← selectHeuristic()
12.    i ← ranInt()
13.    new-sol ← apply(h, new-sol, cross-sol[i])
14.    new-obj ← f(new-sol)
15.    if (new-obj ≥ cur-obj and ranFloat() ≥ 0.5) then /* reject new solution */
16.       new-sol ← cur-sol
17.       new-obj ← cur-obj
18.    else
19.       i ← ranInt()
20.       cross-sol[i] ← new-sol
21.       cross-obj[i] ← new-obj
22.    end
23.    ITERATIONS ← ITERATIONS − 1
24. end
```

Each DBGen run is seeded by a unique number (pair), and thus starts from a unique initial solution x_0. After execution, the resulting sequences of heuristic selections, heuristic parameters, and objective function values are broken down into consecutive subsequences of length $n = 2, 3, \ldots$, and so on. For example, given a sequence of heuristics $\{ \mathtt{M_0C_7L_4L_6R_1} \}$ of length five, the sets of subsequences of lengths two, three, and four are

$$\{\mathtt{M_0C_7},\ \mathtt{C_7L_4},\ \mathtt{L_4L_6},\ \mathtt{L_6R_1}\}, \{\mathtt{M_0C_7L_4},\ \mathtt{C_7L_4L_6},\ \mathtt{L_4L_6R_1}\},\ \text{and}\ \{\mathtt{M_0C_7L_4L_6},\ \mathtt{C_7L_4L_6R_1}\}$$

respectively.

The set of all such subsequences is denoted S. A subsequence $s \in S$ may occur a number of times in the database and each occurrence can have a different subsequence of objective function values depending on the run, and the position in a run where s arises. The objective values also vary depending on the problem, and the initial solution x_0. Define s^i to be the ith occurrence of subsequence s so that

$$s^i = (s^i_1, s^i_2, \ldots, s^i_n)$$

with objective function values

$$(o^i_0, o^i_1, \ldots, o^i_n)$$

for $i = 1, \ldots, N_s$, where N_s is the number of occurrences of each s in S.

It should be noted that some heuristics are parameterised by a variable $p \in \{0, 1, 2, \ldots, 10\}$, and that these heuristic parameters are not directly employed in the following framework. The framework uses sets of heuristic occurrences s^i_j to calculate a number of performance estimates. When parametrised occurrences $s^i_{j,p}$ are considered, these sets become very small, and the performance estimates become unreliable. Grouping all parametrised occurrences together results in estimates that are averaged over all choices of p.

3.3 Estimating Heuristic Performance

In this section, a statistical framework for analysing the offline learning databases described in Section 3.2 is introduced. The framework is used to quantify the effectiveness and behaviour of

1. individual heuristics,
2. subsets of heuristics, and
3. subsequences of heuristics.

The framework is also used to analyse the interactions that occur between the low level heuristics of a subset or subsequence.

The framework employs the mathematical concept of logarithmic returns. Logarithmic returns are used widely in finance where they are employed to compare two or more variables when the originating price series consist of highly unequal values [61]. In this thesis, logarithmic returns are used to normalise subsequences of objective function values.

.

3.3.1 Logarithmic Returns

Each problem domain has its own objective function f, and the range of f may differ between problem instances and problem domains. Without *a priori* knowledge of the objective functions, the objective function values from different instances or domains cannot be compared directly. Instead, subsequences of normalised 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[2] are

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

The *log returns* [61] of this series are simply the sequential differences of the log objective values

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

or equivalently

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

Such subsequences are invariant to scaling. For example, the $\log_{10}$ returns for the objective value subsequences

$$(6,\ 5,\ 4,\ 3,\ 2,\ 1) \text{ and } (120,\ 100,\ 80,\ 60,\ 40,\ 20)$$

are

$$(-0.0792, -0.0969, -0.1249, -0.1761, -0.3010).$$

Figure 3.2 illustrates the $\log_{10}$ returns for a given percentage change in the objective function value.

Each low level heuristic selection s_i in $s = s_1, \ldots, s_n$ is associated with a log return

$$\log\left(\frac{o_i}{o_{i-1}}\right).$$

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

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

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

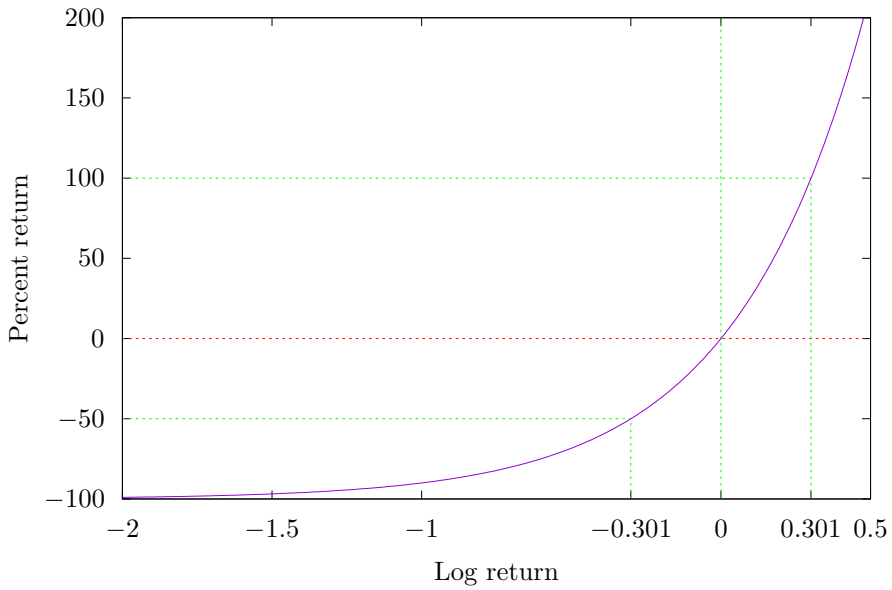


Figure 3.2: The $\log_{10}$ return scale and the equivalent percentage change.

This is not the case for subsequences of *decimal returns* where each return has the form

$$\left(\frac{o_i - o_{i-1}}{o_{i-1}}\right).$$

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 r of x, followed by a change $-r$ of x does not return the original value x. For example, if $o_0 = 100$, a decimal increase of 0.1 is 10 and so $o_1 = 110$. A decimal decrease of 0.1 yields 11 and so $o_2 = 99$. The formulae to convert between $\log_{10}$ returns r and decimal returns d, are

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

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

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

Notice that the objective function is only evaluated at the start, and at the end of a subsequence; intermediate objective function values are not required. 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 two subsequences s_1 and s_2,

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

implies that s_1 is a better subsequence than s_2.

3.3.2 Low Level Heuristics

The effectiveness of a low level heuristic can be measured by the mean log return $\overline{\alpha}$. The *mean log return* of a set of N occurrences of a given low level heuristic s is

$$\overline{\alpha}(\{s^1, \ldots, s^N\}) = \frac{1}{N} \sum_{i=1}^{N} \alpha(s^i).$$

Negative values of $\overline{\alpha}$ indicate an effective heuristic, that tends to reduce the objective function value, while positive values of $\overline{\alpha}$ indicate a disruptive heuristic, that tends to increase the objective function value (see figure 3.2). In this classification, heuristics that have a $\overline{\alpha}$ that is zero, or close to zero, can be regarded as neutral.

The function $\overline{\alpha}$ is the mean of log values. In general, the mean of the logs is not equal to the log of the mean[3]. That is

$$\frac{1}{N} \sum_{i=1}^{N} \log(x_i) \neq \log\Big(\frac{1}{N} \sum_{i=1}^{N} x_i\Big).$$

If the values x_i are all positive (or have the same sign), then the anti-log of the mean of the logs is equivalent to the *geometric mean*. In symbols

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

assuming the values x_i are positive, or

$$(-1)^m \log^{-1}\left(\frac{1}{N} \sum_{i=1}^{N} \log(\,|x_i|\,)\right)$$

otherwise, where m is the number of negative values. The geometric mean normalises the ranges, so that no range dominates the average, and is always less than, or equal to the arithmetic mean. 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 problem instances are more difficult to optimise than others. For this reason, $\overline{\alpha}$ is used in preference to the arithmetic mean of the decimal returns.

The mean log return $\overline{\alpha}$ of the low level heuristics in each HyFlex domain is shown in figure 3.3. The results are calculated over DB_2, and averaged over 4000 runs of 150 heuristic selections.

[3]The median of logs is equal to the log of the median.

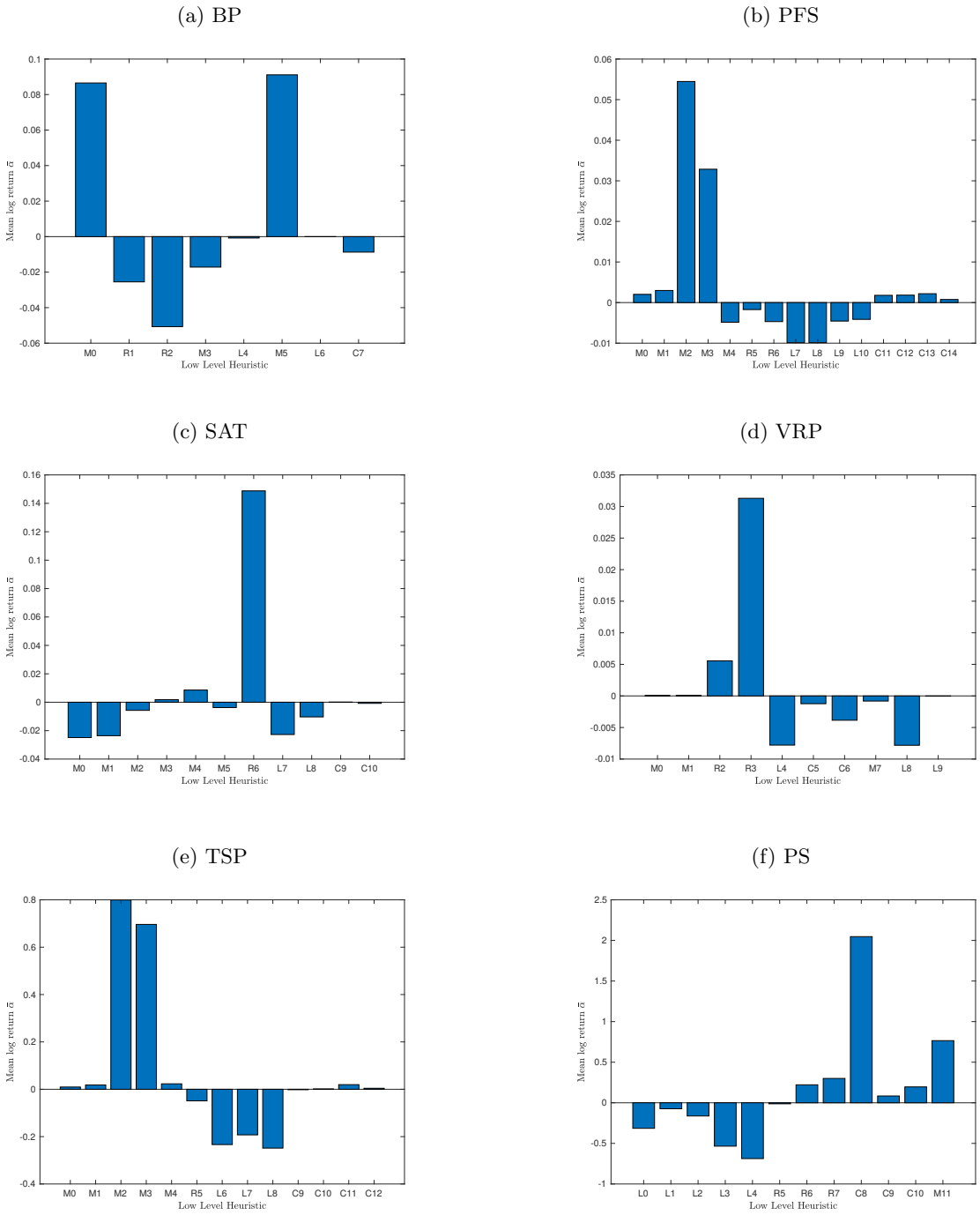


Figure 3.3: The mean log returns $\overline{\alpha}$ for each low level heuristic in DB_2 for each domain.

The figures demonstrate that, unsurprisingly, some low level heuristics are more effective than others. Furthermore, the effectiveness of the heuristic classes $\{\mathtt{M}, \mathtt{C}, \mathtt{R}, \mathtt{L}\}$ varies by domain. For example, the mutation heuristic class `M` is highly disruptive in the TSP and PFS domains, while it is amongst the most effective classes in the SAT domain. Similarly, the ruin and recreate class `R` is disruptive in SAT and VRP, and effective in the BP, PFS and TSP domains.

Figure 3.3 illustrates the low level heuristic performance over the entire optimisation process. However, it is well known that some heuristics are better suited to certain stages or phases of the optimisation process. For example, the work of [97] on workforce scheduling, and that of [108] on course timetabling, demonstrates that some low level heuristics which are ineffective at the start of the search process prove to be highly effective at the end, and *vice versa*, while others heuristics are mainly used at the beginning, middle or end of the process. The mean log return $\overline{\alpha}$ can also be used to characterise the changes in heuristics performance that occur during the optimisation process.

Let the *current objective function* value for a low level heuristic s be o_t, where o_t is the objective function value at iteration t, *before* applying the low level heuristic s at iteration $t+1$. The set HIGH consists of all those instances of a heuristic s which have current objective function values

$$o_t > P^r_{90}$$

where P^r_{90} is the 90th percentile, so that the set HIGH contains the 10% of heuristic instances with the highest current objective function values. The set LOW consists of those instances of a heuristic s which have current objective function values

$$o_t < P^r_{10}$$

where P^r_{10} is the 10th percentile, and so the set LOW contains the 10% of heuristics with the lowest current objective function values. These sets contain heuristic occurrences that occur at the "beginning" of the optimisation process, when objective function values are relatively high, and at the "end" of the optimisation process, when objective function values are relatively low.

Calculating the percentile values over the runs or sequences in each domain can lead to heuristic occurrences from a few problem instances dominating the sets; those that produce very high or very low objective function values. Here the percentiles P^r are calculated locally over the objective function values of *each* sequence $r = 1, \ldots, 4000$ in DB_2. This ensures that heuristic instances from all the problems in a domain are included in the sets.

The low level heuristic selections in the HIGH and LOW sets can be ranked by their $\overline{\alpha}$ values. The differences in these rankings demonstrate the changes in heuristic effectiveness that occur during the optimisation process. The heuristic selections in the HIGH and LOW sets, ranked by their $\overline{\alpha}$ value in increasing order from left to right, are shown in table 3.4. The change in $\overline{\alpha}$-order can be quantified by calculating the Spearman's Footrule distance [33]. The Footrule distance is calculated by taking the sum of the absolute values of the difference between two ranks. In symbols, if σ and π denote two

Table 3.4: The low level heuristics in the HIGH and LOW subsets in DB_2 for each domain ordered by ascending mean log return $\bar{\alpha}$ from left to right, the Spearman's Footrule distance d, and the normalised Footrule distance $d' = \frac{d}{m}$.

Dom.	Set	Order	d	d'
BP	HIGH	R_2, R_1, M_3, C_7, L_4, L_6, M_0, M_5	8	0.2500
	LOW	R_2, M_3, L_4, C_7, L_6, R_1, M_0, M_5		
PFS	HIGH	L_7, L_8, M_4, R_6, L_9, R_5, L_{10}, C_{13}, C_{11}, C_{14}, C_{12}, M_1, M_0, M_3, M_2	46	0.4107
	LOW	L_7, L_8, L_9, L_{10}, C_{14}, C_{12}, M_0, R_6, C_{13}, M_1, C_{11}, R_5, M_4, M_3, M_2		
SAT	HIGH	L_7, M_0, M_1, L_8, C_{10}, C_9, M_2, M_5, M_3, M_4, R_6	10	0.1667
	LOW	M_0, M_1, L_7, L_8, M_2, C_{10}, M_5, C_9, M_3, M_4, R_6		
VRP	HIGH	R_3, R_2, C_6, L_8, L_4, M_7, C_5, L_9, M_0, M_1	42	0.8400
	LOW	L_4, L_8, L_9, M_1, M_0, C_6, C_5, M_7, R_2, R_3		
TSP	HIGH	L_6, L_7, L_8, R_5, C_9, C_{10}, C_{12}, C_{11}, M_2, M_3, M_1, M_4, M_0	38	0.4524
	LOW	L_8, L_6, L_7, M_0, C_{10}, C_{12}, C_9, M_1, M_4, C_{11}, R_5, M_3, M_2		
PS	HIGH	L_4, L_3, L_0, C_9, R_7, R_6, R_5, L_2, L_1, M_{11}, C_{10}, C_8	30	0.4167
	LOW	L_3, L_4, L_2, L_1, L_0, R_5, C_9, C_{10}, R_6, R_7, M_{11}, C_8		

permutations of n elements such that $\sigma(i)$ and $\pi(i)$ denote the rank of an element $i = 1, \ldots, n$ in the permutation, then Spearman's Footrule is defined by

$$d(\sigma, \pi) = \sum_{i=1}^{n} |\sigma(i) - \pi(i)|$$

and has a maximum integer value of $m = \lfloor \frac{1}{2} n^2 \rfloor$ where $\lfloor \cdot \rfloor$ is the floor function. The greater the Footrule distance, the greater the difference between the two orders.

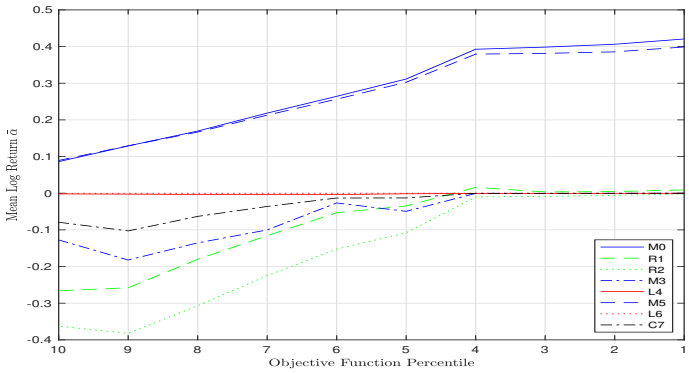
The rankings show that some heuristics are indeed better suited to some phases of the optimisation process than others, and that these changes in order vary by domain. For example, the VRP domain has a high d' value indicating that the effective heuristics at the start of the optimisation perform poorly later on and *vice versa*. This can be seen in the heuristic orderings, with ruin and recreate performing well when objective function values are high, but poorly when they are low. The SAT domain has the lowest d' value, and shows a much more consistent order with the same mutation and local search operators appearing in the top four positions, regardless of the optimisation state.

The heuristic instances can be further separated by their current objective function values into 10 sets

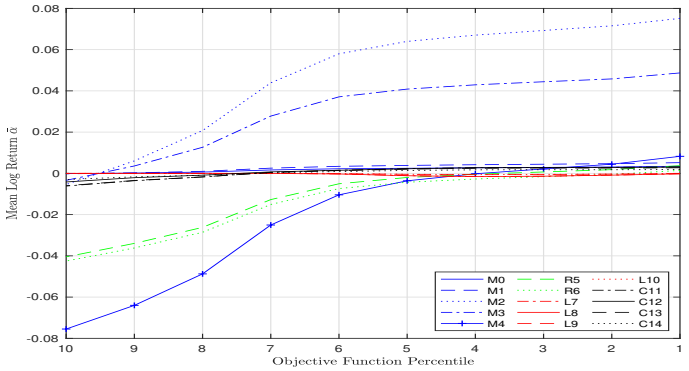
$$P_1 = [P_0^r, P_{10}^r],\ P_2 = [P_{10}^r, P_{20}^r], \ldots, P_{10} = [P_{90}^r, P_{100}^r]$$

where $P_1 = [P_0^r, P_{10}^r]$ and $P_{10} = [P_{90}^r, P_{100}^r]$ correspond to the LOW and HIGH sets defined previously. The $\overline{\alpha}$ values are calculated over these 10 sets, and the results of plotting $\overline{\alpha}$, calculated over DB_3, against each interval are shown in figure 3.4 and figure 3.5.

(a) The BP domain.



(b) The PFS domain.



(c) The SAT domain.

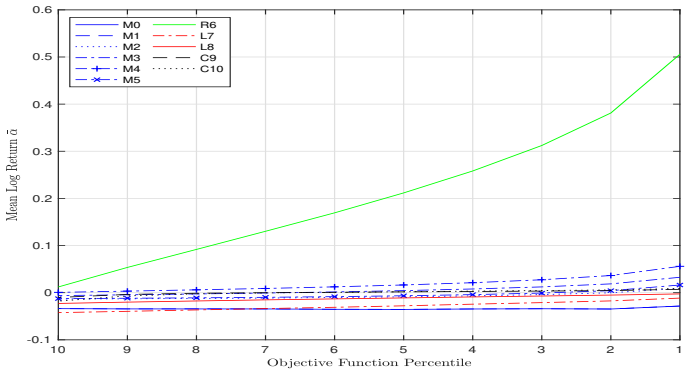
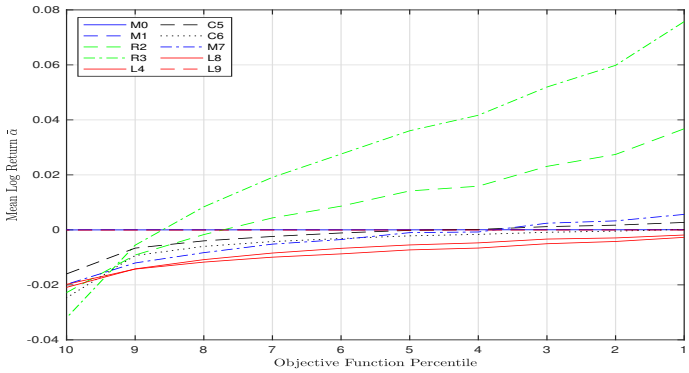
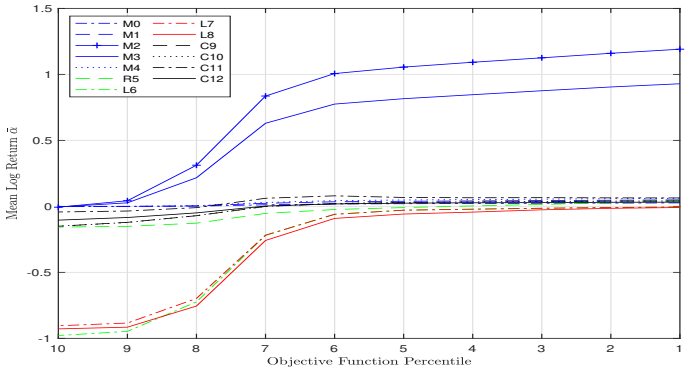


Figure 3.4: The mean log return $\overline{\alpha}$ of the low level heuristics in DB_3 for the BP, PFS, and SAT domains, averaged over the 10 local percentiles.

(a) The VRP domain.



(b) The TSP domain.



(c) The PS domain.

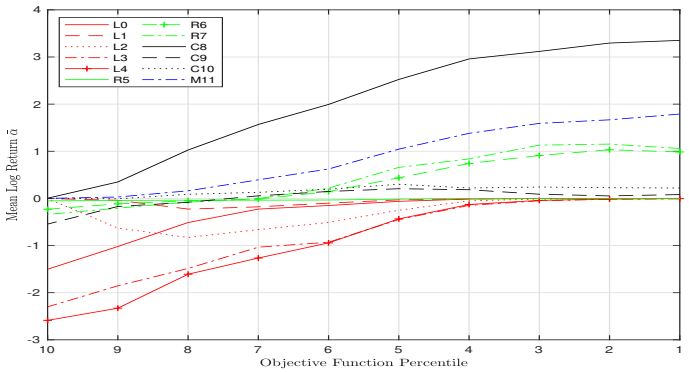


Figure 3.5: The mean log return $\overline{\alpha}$ of the low level heuristics in DB_3 for the VRP, TSP and PS domains domain, averaged over the 10 local percentiles.

The figures illustrate that the low level heuristics tend to be more effective early on in the optimisation process when the objective function is relatively high as one would expect. However, the heuristics not only vary in effectiveness during the optimisation process, but the relationship between them also changes.

3.3.3 Subsets

As noted in Chapter 2, a number of studies have examined the selection and performance of subsets of low level heuristics. In this section, the methodology developed for individual low level heuristics in Section 3.3.2 is applied to subsets of heuristics.

Consider a subset of k low level heuristics chosen from the heuristics of a given domain (see table 3.2). The (unordered) subset can be also be viewed as $k!$ ordered subsequences; the subsequences constructed from the permutations without repetition of the k low level heuristics. For example, the permutations of the BP heuristic subset $\{\mathtt{M}_0, \mathtt{R}_1, \mathtt{R}_2\}$ are

$$\mathtt{M}_0\mathtt{R}_1\mathtt{R}_2,\ \mathtt{R}_1\mathtt{M}_0\mathtt{R}_2,\ \mathtt{M}_0\mathtt{R}_2\mathtt{R}_1,\ \mathtt{R}_2\mathtt{M}_0\mathtt{R}_1,\ \mathtt{R}_1\mathtt{R}_2\mathtt{M}_0,\ \text{ and }\ \mathtt{R}_2\mathtt{R}_1\mathtt{M}_0.$$

In general, the *mean unit log return* of a set of N subsequences s_i is

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

The performance of a subset of k heuristics is estimated from the online learning database by calculating the mean unit log return $\overline{\beta}$ over the $k!$ permutations of that subset. For example, the performance of the subset $\{\mathtt{M}_0, \mathtt{R}_1, \mathtt{R}_2\}$ in DB_2 is -0.0031 (see table 3.5). Using $k!$ permutations *without* repetition reduces the number of possible subsequences that are required to calculate a subset's performance while, it is hoped, still producing a reliable estimate[4].

Consider the set C of all combinations of k elements constructed from the heuristics in each domain. The elements of C are subsets of k heuristics. The number of k-element subsets that can be constructed from a given set of $n > k$ heuristics is

$$C_k^n = \frac{n!}{k!(n-k)!}.$$

Following Section 3.3.2, the C_k^n subsets $c \in C$ can be ordered by their $\overline{\beta}$ and divided into effective subsets, where $\overline{\beta} < 0$, and disruptive subsets, where $\overline{\beta} > 0$.

The sets LOW and HIGH can also be defined for subsets of heuristics. In this case, the sets LOW and HIGH consist of all those instances of a heuristic subset c which have a current objective function value

$$o_t < P_{10}^r \ \text{ and } \ o_t > P_{90}^r$$

[4]There are n^k subsequences *with* repetition.

Table 3.5: The mean unit log return $\overline{\beta}(s)$, the standard deviation SD, and the number N_s of occurrences of the permutations of $\{\mathtt{M_0}, \mathtt{R_1}, \mathtt{R_2}\}$ in DB_2 for the BP domain.

s	$\overline{\beta}(s)$	SD	N_s
$\mathtt{M_0R_1R_2}$	-0.0186	0.0410	812
$\mathtt{M_0R_2R_1}$	-0.0142	0.0337	920
$\mathtt{R_1M_0R_2}$	-0.0128	0.0461	893
$\mathtt{R_2M_0R_1}$	-0.0090	0.0390	1539
$\mathtt{R_2R_1M_0}$	0.0156	0.0449	1229
$\mathtt{R_1R_2M_0}$	0.0214	0.0603	719
$\{\mathtt{M_0}, \mathtt{R_1}, \mathtt{R_2}\}$	-0.0031	0.0462	6112

respectively, where P^r_{10} is the 10th and P^r_{90} is the 90th percentile. The set LOW contains the 10% of heuristic subsets with the lowest current objective function values while the set HIGH contains the 10% of heuristic subsets with the highest current objective function values. Ranking the subsets by their $\overline{\beta}$ value, calculated over DB_2, in increasing order from left to right demonstrates the change in effectiveness between the HIGH and LOW sets (see table 3.6).

The results demonstrate that some heuristic subsets, like individual heuristics, are better suited to some phases of the optimisation process than others, and that these changes in order vary by domain.

Many of the best subsets in the HIGH and LOW subsets are made up of the best performing heuristics. For example, in the BP domain, which has the smallest change in subset order, the effective subsets consist of the same four best heuristics (see table 3.4).

However, this is not always the case. In the PFS domain the $\mathtt{M_0}$ heuristic occurs in both the best HIGH and LOW subsets despite being a relatively poor performer. Similarly, in the TSP domain the heuristics $\mathtt{M_0}$ occurs in the best HIGH subsets despite being the worst performing heuristic in that set. In the VRP domain, which exhibits the largest change in subset order, the $\mathtt{C_7}$ heuristic occurs in the best HIGH subset while, again, being a poor performer individually. These results, strengthen the argument that the optimisation performance of a subset is not solely due to individual low level heuristic performance, but also a function of the interaction between individual heuristics [119].

As the performance of a k element heuristic subset is estimated from $k!$ subsequences of those heuristics, the question naturally arises as to the effect of (subsequence) order on optimisation performance. The effect of order on a subset c is estimated by the *order sensitivity* $\delta(c)$ which is calculated by subtracting the mean unit log return of the best performing subsequence $s_{\min}$ from the mean unit log return of the worst performing subsequence $s_{\max}$. For the example given in table 3.5 this is

$$\delta(c) = \overline{\beta}(s_{\max}) - \overline{\beta}(s_{\min}) = 0.0214 - -0.0142 = 0.0356.$$

Table 3.6: The best and worst performing subsets in the HIGH and LOW sets in DB_2 for each domain ordered by ascending mean log return $\bar{\beta}$ from left to right, the Spearman's Footrule distance d, and the normalised Footrule distance $d' = \frac{d}{m}$.

Dom.	Set	Best	Worst	d	d'
BP	HIGH	$\{R_1, R_2\}$ $\{R_2, M_3\}$ $\{R_2, C_7\}$	$\{M_0, L_4, M_5\}$ $\{L_4, M_5\}$ $\{M_0, M_5\}$	1056	0.2993
	LOW	$\{R_2, C_7\}$ $\{R_2, M_3, C_7\}$ $\{R_2, M_3\}$	$\{M_0, L_4\}$ $\{M_0, L_4, M_5\}$ $\{M_0, M_5\}$		
PFS	HIGH	$\{L_7, L_{10}\}$ $\{L_8, C_{12}\}$ $\{M_0, L_7\}$	$\{M_3, C_{12}, C_{14}\}$ $\{M_2, L_7, C_{14}\}$ $\{M_2, R_6, C_{12}\}$	86176	0.5496
	LOW	$\{M_0, L_8, C_{13}\}$ $\{L_7, L_9, L_{10}\}$ $\{L_7, L_{10}, C_{11}\}$	$\{M_2, L_{10}\}$ $\{M_2, M_3\}$ $\{M_2, C_{11}\}$		
SAT	HIGH	$\{L_7, C_{10}\}$ $\{M_1, L_7\}$ $\{M_0, L_7\}$	$\{M_4, R_6, C_{10}\}$ $\{R_6, C_9, C_{10}\}$ $\{R_6, C_{10}\}$	8118	0.3355
	LOW	$\{M_0, M_1, L_7\}$ $\{M_0, M_1\}$ $\{M_0, L_7\}$	$\{M_3, R_6\}$ $\{M_4, R_6\}$ $\{R_6, L_7\}$		
VRP	HIGH	$\{R_3, C_5\}$ $\{R_2, R_3\}$ $\{R_3, L_8\}$	$\{M_0, L_9\}$ $\{M_1, L_9\}$ $\{M_0, M_1\}$	12752	0.9368
	LOW	$\{L_4, L_8\}$ $\{M_1, L_4, L_8\}$ $\{L_4, L_8, L_9\}$	$\{R_3, M_7\}$ $\{R_2, R_3\}$ $\{M_1, R_3\}$		
TSP	HIGH	$\{L_6, C_{10}\}$ $\{M_0, L_6\}$ $\{L_8, C_{12}\}$	$\{M_2, M_4, C_{11}\}$ $\{M_2, M_3, C_9\}$ $\{M_0, M_2, M_4\}$	28840	0.4353
	LOW	$\{R_5, L_6, C_{12}\}$ $\{L_7, L_8, C_{11}\}$ $\{M_1, R_5, C_{10}\}$	$\{M_2, C_9\}$ $\{M_2, M_4\}$ $\{M_2, M_3\}$		
PS	HIGH	$\{L_3, L_4\}$ $\{L_4, C_9\}$ $\{L_2, L_4\}$	$\{L_0, L_1, C_8\}$ $\{L_1, C_8, C_{10}\}$ $\{L_0, C_8, C_{10}\}$	15786	0.3860
	LOW	$\{L_3, L_4, R_5\}$ $\{L_3, L_4\}$ $\{L_2, L_3\}$	$\{L_1, C_8\}$ $\{C_8, M_{11}\}$ $\{C_8, C_{10}\}$		

The mean of the order sensitivities over all C_k^n subsets is termed the *mean order sensitivity* and is denoted $\bar{\delta}_k$. The mean order sensitivity of the subsets of size $k = 2$ and $k = 3$, denoted $\bar{\delta}$, and the heuristics with the minimum and maximum order sensitivity, calculated over DB_2, for each domain are shown in table 3.7.

Table 3.7: The mean order sensitivity $\bar{\delta}$, the standard deviation SD, and the subsets of heuristics with the minimum and maximum order sensitivity $\delta(s)$ in DB_2 for each domain.

Dom.	$\bar{\delta}$	SD	Min. s	Min.	Max. s	Max.	$C_2^n + C_3^n$
BP	0.0410	0.0399	$\{L_6, L_4\}$	0.0000	$\{M_5, R_2\}$	0.1432	84
PFS	0.0037	0.0054	$\{M_0, M_1\}$	0.0000	$\{M_2, M_4\}$	0.0344	560
SAT	0.0059	0.0079	$\{M_5, L_8\}$	0.0000	$\{R_6, C_{10}, M_0\}$	0.0365	220
VRP	0.0017	0.0015	$\{M_0, M_1\}$	0.0000	$\{R_3, C_6, L_4\}$	0.0065	165
TSP	0.0780	0.1123	$\{M_4, M_0\}$	0.0000	$\{M_2, L_6\}$	0.4878	364
PS	0.4271	0.3052	$\{R_5, L_0\}$	0.0022	$\{C_8, M_{11}, L_4\}$	1.5150	286

The results demonstrate that some subsets of heuristics are more order sensitive than others, and that the causes for this sensitivity, or lack thereof, are varied. For example, in the BP domain the subset $\{L_6, L_4\}$ is the least sensitive. This is because the heuristics L_6 and L_4 are the most ineffectual heuristics (see figure 3.3a) and so changing their ordering has little effect on the performance. The

subset $\{\texttt{M}_0, \texttt{M}_5\}$ also has a low order sensitivity of $\delta = 0.0007$. In this case, $\texttt{M}_0$ and $\texttt{M}_5$ are the most disruptive heuristics in the BP domain, and so again, changing their ordering has little effect. In contrast, the $\texttt{M}_5$ and $\texttt{R}_2$ heuristics are the most disruptive and effective heuristics respectively in the BP domain, and changing their ordering produces the largest change in performance. The magnitude of order sensitivity also varies significantly across domains with a maximum of 0.0065 in VRP up to a maximum of 1.5150 in PS; a difference of three orders of magnitude.

Figure 3.6 shows the effective and disruptive subsets of C in DB_3, for each domain, ordered by their $\overline{\beta}$, and plotted against a subset's order sensitivity δ. The red crosses indicate subsets that contain subsequences that can be reordered so that a disruptive subsequence can be transformed into an effective one, and *vice versa*. These subsets are termed *order sensitive* subsets. The figure illustrates that the number of order sensitive subsets also varies significantly by domain. Although these subsets tend to have the largest sensitivities, in general they exhibit poor performance as their mean log return $\overline{\beta}$ is already close to zero. For example, the SAT domain contains two order sensitive subsets; the least number of any domain. The two subsets are

$$\{\texttt{M}_0, \texttt{M}_3, \texttt{C}_9\} \text{ with } \delta = 0.0012 - -0.0002 = 0.0014$$

and

$$\{\texttt{M}_3, \texttt{M}_5, \texttt{R}_6\} \text{ with } \delta = 0.0018 - -0.0003 = 0.0021$$

and these subsets are ranked 144^{th} and 145^{th} out of 220 when the subsets are ordered by $\overline{\beta}$.

The effective subset (ranked 65^{th}) with the largest sensitivity

$$\delta = -0.0116 - -0.0178 = 0.0062$$

in SAT is $\{\texttt{L}_8, \texttt{C}_9, \texttt{C}_{10}\}$. The existence of such subsets demonstrates that order sensitivity can also be exploited in some of the better performing subsets.

In contrast, the PS domain has the largest proportion of order sensitive and effective order sensitive subsets, and the changes in mean log return can be quite large. For example, the subset $\{\texttt{L}_2, \texttt{R}_5, \texttt{M}_{11}\}$ is the effective subset (ranked 112^{th} out of 286) with the largest

$$\delta = 0.3578 - -0.6468 = 1.0046.$$

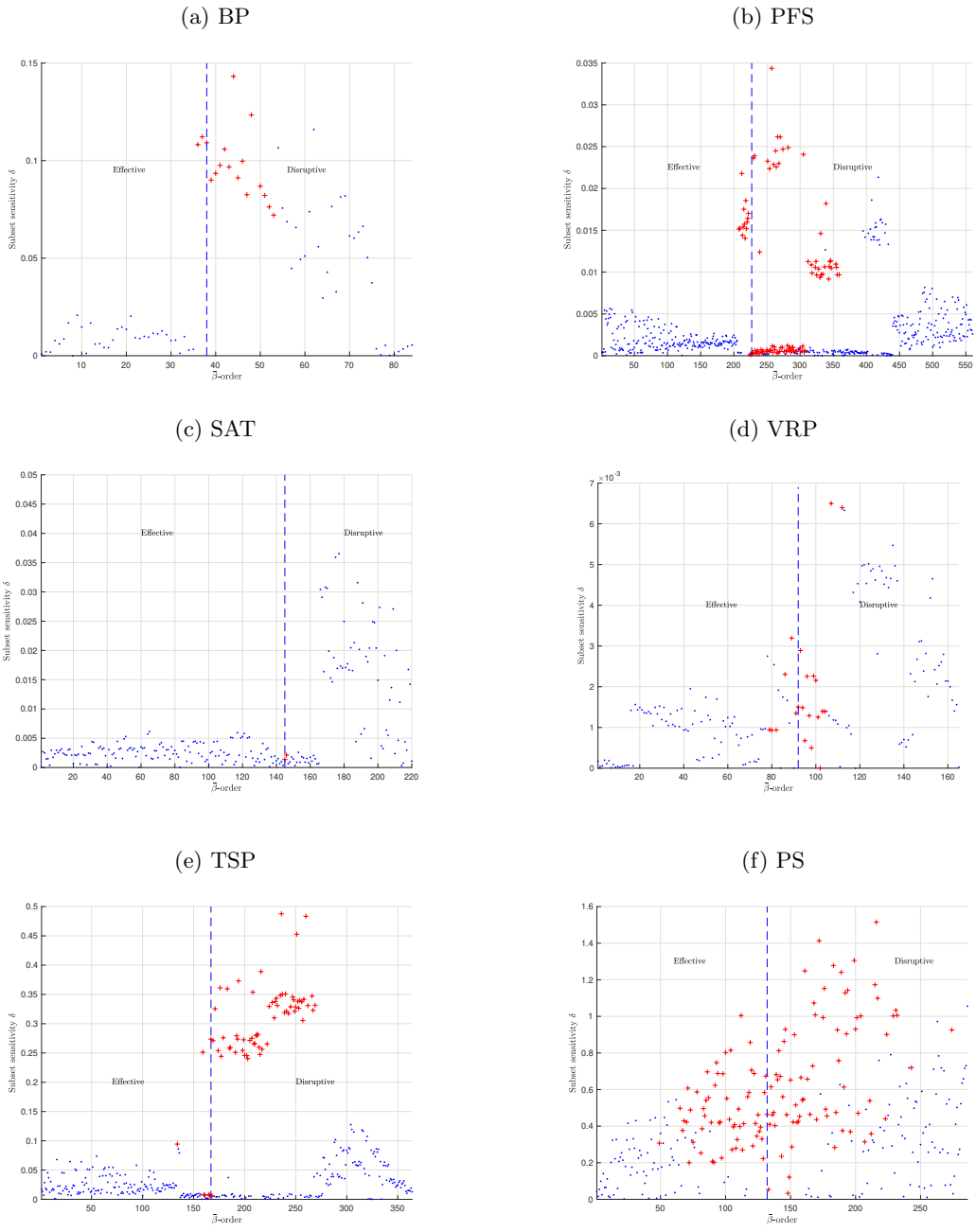


Figure 3.6: Subset sensitivity δ versus the $\overline{\beta}$-order in DB_3 for each domain. The red crosses denote subsets that contain subsequences that can be transformed from a disruptive subsequence to an effective one (and *vice versa*) by reordering.

3.3.4 Subsequences

It is clear from Section 3.3.3 that the choice and order of the low level heuristics in a subsequence of heuristic selections is crucial to optimisation efficacy.

The unit log return of a set of N subsequences is

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

The function β is a *signed measure*. The Hahn decomposition theorem states that if β is a signed measure there exist disjoint subsets S^+ and S^- of S such that S^+ is the set of positive values of β, S^- is the set of negative values of β, and $S = S^+ \cup S^-$. Application of Jordan's decomposition theorem separates out the positive and negative parts of β. In symbols $\beta = \beta^+ - \beta^-$ where

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

for every subset U of S. The functions β^+ and β^- are *measures* [24].

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 thesis, the γ-*ratio*, defined by

$$\gamma(U_s) = \frac{\beta^-(U_s)}{\beta^+(U_s) + 1} \tag{3.1}$$

is used to select subsequences from the offline database. Large values of $\gamma(U_s) > 1$ indicate an effective subsequence. Small values of $\gamma(U_s) < 1$ indicate a disruptive subsequence.

The γ-ratio, and the mean unit log return $\overline{\beta}$ produce different orderings over the subsets and subsequences of the HyFlex domains. Table 3.8 shows the normalised Footrule distance d' between the γ-ordering and the $\overline{\beta}$-ordering of the subsets C, and the subsequences S of length two and three. The results show that the orderings are similar with a maximum distance of 0.2946 for the subsequences in PFS. Although the γ-ratio and the mean unit log return $\overline{\beta}$ produce similar orderings over the subsets and subsequences of the HyFlex domains, the γ-ratio is used in preference to $\overline{\beta}$ for selecting subsequences because, in practice, it produces consistently better results.

Table 3.8: The normalised Footrule distance d' between the γ-ordering and $\overline{\beta}$-ordering of the subsets and subsequence in DB_2 for each domain.

Dom.	Subsets	Subseqs.
BP	0.1395	0.1472
PFS	0.1826	0.2946
SAT	0.1050	0.1925
VRP	0.1956	0.2747
TSP	0.1610	0.2267
PS	0.1284	0.1687

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.

3.4 Hyper-heuristic Performance

The performance of a hyper-heuristic is measured against the five criteria shown in table 3.9.

Table 3.9: Hyper-heuristic performance criteria.

Name	Description
$\overline{\alpha}_f$	Overall mean change in objective function value.
Min. Sel.	Heuristic selections required to find $x_{\min}$.
Obj. Eval.	Objective function evaluations required to find $x_{\min}$.
Min. T	Time required to find $x_{\min}$ (ms).
Total T	Overall run time (ms).

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 final best 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 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 necessarily 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.

3.5 Statistical Validation

In the evolutionary computation literature statistical tests are widely used to compare and rank the performance of algorithms that have been evaluated on a number of benchmark problems (see for example [48], [49], [32], and [116]). Such test are usually non-parametric. Although non-parametric tests tend to be less powerful than parametric tests they are more robust as they require less statistical assumptions.

The non-parametric, one tailed Wilcoxon signed-rank test is used to validate the proposed methodology by establishing a *stochastic ordering* 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. The null hypothesis is rejected if the p-value is less than 0.01, and 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 $\overline{\alpha}_f(\mathrm{A})$ is "smaller" than the random variable $\overline{\alpha}_f(\mathrm{B})$, and thus hyper-heuristic A is more effective than hyper-heuristic B. In symbols

$$\overline{\alpha}_f(\mathrm{A}) < \overline{\alpha}_f(\mathrm{B})$$

with 99% confidence.

The $\alpha_f(\text{A})$ and $\alpha_f(\text{B})$ values can be paired because the initial seed (pair) and therefore the initial solution x_0 for each problem instance p in a domain, and each run r is the same for both hyper-heuristics.

The Wilcoxon test assumes that the paired values of $\alpha_f(\text{A})$ and $\alpha_f(\text{B})$ are independently drawn. The use of cross-validation to determine generalisation violates this assumption as the training sets overlap. This overlap may prevent the statistical test from obtaining a good estimate of the amount of variation that would be observed if the training sets were completely independent. As a consequence, when cross-validation is used, the results of the statistical tests must be viewed as approximate rather than rigorously correct [34].

The null hypotheses of the Wilcoxon test is that the median difference between pairs of observations is zero. As the differences between the $\alpha_f(\text{A})$ and $\alpha_f(\text{B})$ values are not symmetrically distributed around the median, the Hodges-Lehmann estimate of the median is used instead [58].

All statistical tests were performed using the R language for statistical computing [93].

3.6 Conclusions

This chapter has presented a methodological framework for the construction and analysis of a number of offline learning databases. Specifically, a simple, random, selection hyper-heuristic DBGen is repeatedly executed on the HyFlex set of benchmark problems, in order to generate sequences of heuristic selections, heuristic parameters, and objective function values.

A novel statistical framework is used to analyse the selections of low level heuristics, based on the concept of logarithmic returns. Logarithmic returns are used for the categorisation of

1. individual heuristics,
2. subsets of heuristics, and
3. subsequences of heuristics,

based on their associated objective function values. Heuristic selections can be classed as either effective when they tend to reduce the objective function value, or disruptive when they tend to increase the objective function value. Logarithmic returns can also be applied to the measurement and analysis of hyper-heuristic performance in general.

The proposed framework is able to identify, and quantify, well known heuristic behaviours, such as the effect of the choice and order of heuristics on optimisation performance, and the differences in heuristic performance that occur during the optimisation process. Furthermore, it should be noted that this framework does not depend on the problem domain, the number or type of heuristics, or the process used to generate the heuristic selections.

Chapter 4

An Analysis of Heuristic Subsequences

In this chapter, the framework for offline learning developed in Chapter 3 is used to identify and analyse subsequences of heuristic selections. The mean unit log return $\overline{\beta}$ and the γ-ratio, introduced in Section 3.3.4, measure the propensity of a subsequence s to increase or decrease the objective function value. The larger the measure, the larger the propensity for change. Here, the γ-ratio is used to select sets of effective and disruptive subsequences of heuristics from the offline database DB_1. Although $\overline{\beta}$ and γ produce similar orderings over the subsets and subsequences of the HyFlex domains, γ is used in preference to $\overline{\beta}$ for selecting subsequences because, in practice, it produces consistently better results.

The selected subsequence sets are used to parameterise a selection hyper-heuristic, denoted EvalHH, which is executed on a number of unseen HyFlex problem instances in order to evaluate the subsequences, and by extension the framework. The EvalHH results are then compared with results produced by the SSHH hyper-heuristic described in Chapter 2.

The six experiments presented here (five of which have been published in [126]) are designed to explore the effect of calculating the γ-ratio for subsequences of low level heuristics, heuristic classes within a domain, and heuristic classes across a number of domains.

The identification of effective subsequences of low level heuristics is useful because they can be used to improve optimisation performance either directly by using them to construct a sequence-based selection hyper-heuristic, or indirectly as training patterns for some offline learning algorithm. Effective subsequences of heuristic classes can, in some cases, also be useful in constructing optimisers for problems from novel or unseen domains.

The experiments in this chapter only asses the performance of the selected subsequences. Their suitability for use as the training patterns for a learning algorithm is investigated in Chapter 5.

Of the six HyFlex domains, only the BP, PFS, SAT and PS domains are used. The TSP and VRP

domains are not used, and are reserved for future experiments.

The structure of this chapter is as follows. Section 4.1, presents the selection hyper-heuristic EvalHH which is used to evaluate a given set of heuristic subsequences. In Section 4.2, an effective subsequence set of low level heuristics is selected using a γ-ratio that is calculated over each domain, and evaluated with EvalHH on "unseen" HyFlex problem instances. In Section 4.3 this experiment is repeated for subsequences of heuristic classes. The objective of these two experiments is to assess the scope and limitations of the proposed framework, and demonstrate empirically its predictive capability. Another objective of this chapter is to determine the existence of effective cross-domain subsequences of heuristic classes. With this in mind, in Section 4.4, an effective and a disruptive subsequence set of heuristic classes are selected using a γ-ratio that is calculated over the whole database. Section 4.5 presents a detailed analysis of these results for the Bin Packing domain, which leads in Section 4.6 to the development of a method for improving the performance of cross-domain subsequences. 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 and compared with a number of other well known hyper-heuristics [67]. The published results demonstrate that SSHH is able to outperform the then best-in-class hyper-heuristic AdapHH [82] on these problems.

The preceding experiments all employ short subsequences of length two and three. In Section 4.7 the issue of subsequence length is examined by selecting and evaluating an effective cross-domain subsequence set, where the subsequences have unrestricted length.

Section 4.8 presents an analysis of the performance of the subsequence sets employed so far, using the concept of *Pareto rank* which allows comparisons to be made across the four criteria of mean number of selections to a minimum, mean number of objective function evaluations to a minimum, mean time to a minimum, and mean overall time.

Finally, 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 experiment in Section 4.9 addresses the issue of run length by evaluating two subsequence sets for 10 minutes of wall clock time which produces a more realistic number of selections.

For clarity, the experiments are labeled according to the following convention. The results of selecting subsequences using a γ-ratio 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 "globally" 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.

The abbreviations used throughout the following sections together with their descriptions are summarised in table 4.1.

Table 4.1: Experimental abbreviations, descriptions, and sections.

Abbr.	Description	Sec.
DBGen	Hyper-heuristic used to generate the learning database.	C2–2.5
SSHH	Sequenced-based hyper-heuristic used for comparisons.	C3–3.2
EvalHH	Hyper-heuristic used to evaluate subsequence sets.	4.1
TOP-DH	Subsequences of low level heuristics with the largest γ (by domain).	4.2
TOP-DC	Subsequences of heuristic classes with the largest γ (by domain).	4.3
TOP-GC	Subsequences of heuristic classes with the largest γ (all domains).	4.4
BOT-GC	Subsequences of heuristic classes with the smallest γ (all domains).	4.4
SINGLE	The individual heuristic classes $\{\,\texttt{L}, \texttt{C}, \texttt{R}, \texttt{M}\,\}$.	4.4
RAND	Randomly generated subsequences of heuristic classes.	4.4
WXYZ	Subsequences of reordered heuristic classes with the largest γ (unseen domains).	4.6
CLMR	Subsequences of heuristic classes with the largest γ (unseen domains).	4.6
LONG-GC	Arbitrary length subsequences of heuristic classes.	4.7
{L}	Singleton heuristic class.	4.7
{LL}	Singleton heuristic class subsequence.	4.7

4.1 The EvalHH Hyper-heuristic

The EvalHH selection hyper-heuristic, shown in listing (10), is used to evaluate a set of subsequences. It is a sequence-based hyper-heuristic which employs a fixed set of k heuristic subsequences of arbitrary lengths. 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.

In this chapter, the crossover mechanism used in the EvalHH 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

Algorithm 10 The EvalHH hyper-heuristic in pseudocode.

```
1. ITERATIONS ← MAX_ITER
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
```

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.

4.2 Low Level Heuristics

In this experiment, the γ-ratio is used to extract subsequences of low level heuristics from the DB_1 offline learning database (see Chapter 3, Sections 3.3.4 and 3.2 respectively). The subsequences are chosen to be effective, that is they are assumed to reduce the objective function value, and this assumption is verified experimentally using EvalHH. An examination of the selected subsequences will also allow the principle that "exploration followed by exploitation" is an effective strategy to be tested [67].

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

select s *from subsequence where length* ≤ 3 *order by* $\gamma(s)$ *descend limit* 10

where

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

The γ-ratio is calculated using a *leave-one-out* cross-validation methodology [9]. Recall that there are 10 problem instances of interest in each of the four HyFlex domains. For each target problem in a domain, γ is calculated from the sequences of the remaining nine problems. The chosen subsequences are then evaluated on the target problem. This ensures that the subsequences are always evaluated on a problem that is "unseen". This methodology is illustrated in figure 4.1.

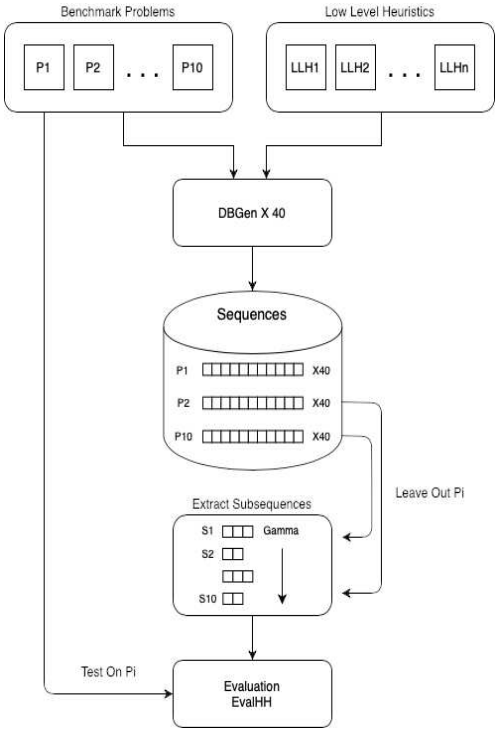


Figure 4.1: A flow chart of the leave-one-out cross-validation methodology used to evaluate the performance of subsequences of low level heuristics in a given domain.

This methodology gives rise to 40 subsequence sets, one for each problem instance in each HyFlex 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

Table 4.2: The TOP-DH subsequences of low level heuristics for each domain.

BP	$\gamma(s)$	PFS	$\gamma(s)$	SAT	$\gamma(s)$	PS	$\gamma(s)$
R_2R_2	43.8493	M_4L_7	4.0900	M_1M_0	16.2481	L_4L_4	295.7140
M_3R_2	28.9555	M_4L_8	3.6249	M_1M_1	16.1602	L_4L_3	271.2990
R_2L_4	28.6750	M_4L_{10}	3.2020	$\underline{L_7M_1}$	16.0079	L_4L_2	269.6110
$\underline{L_4R_2}$	26.1469	R_6R_6	3.0929	M_0M_1	15.7271	L_3L_4	262.8780
R_1R_2	24.3380	M_4L_9	3.0856	$\underline{L_7M_0}$	15.4519	L_2L_4	225.9580
R_2L_6	19.7258	R_5M_4	3.0016	M_0M_0	15.1376	L_3L_3	224.4890
C_7R_2	19.6726	M_4R_6	2.9105	M_0L_7	14.9501	L_1L_4	214.2970
R_2C_7	19.5043	R_6M_4	2.8195	L_7L_7	14.9351	L_0L_2	205.0780
R_2M_3	19.3604	R_5R_5	2.6015	M_1L_7	14.3604	L_2L_3	201.9660
$\underline{L_6R_2}$	18.4136	R_5R_6	2.5843	$\underline{L_8M_0}$	11.3984	L_0L_3	200.1810

TOP-DH. The subsequence sets and their γ-ratios (calculated over *all* the problem instances) for each domain are shown in table 4.2.

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 4.2 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 R_2. The BP ruin and recreate heuristics are *destroy x highest bins* and *destroy x lowest bins* [64]. 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 improvement in objective function value than the local search heuristics (see figure 4.2).

The overall γ-ratio of each subsequence set shown in table 4.2 is shown in table 4.3 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 4.4 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 over 40 runs of each HyFlex problem; a total of 1600 runs. Each run of `MAX_ITER` = 150 selections is seeded by a single unique number

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

Table 4.3: 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

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 DB_1. For EvalHH, the low level heuristic parameters are chosen at random.

Table 4.4: The mean final log return $\overline{\alpha}_f$ for EvalHH (using the subsequence sets TOP-DH), and SSHH. The domain statistics are calculated over 400 runs. Winning scores are shown in bold.

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 results demonstrate that the hyper-heuristic EvalHH, using the subsequence set TOP-DH, outperforms SSHH overall, and on the BP, PFS and SAT domains (see table 4.4). The results of the Wilcoxon signed-rank test (see Chapter 3, Section 3.5) are shown in table 4.5.

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

$$\overline{\alpha}_f(\text{TOP-DH}) < \overline{\alpha}_f(\text{SSHH})$$

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 the subsequence-based approach and the statistical framework proposed in this thesis. Of course, the online method must learn which heuristics and parameters to

Table 4.5: The Wilcoxon test results for $\overline{\alpha}_f$(TOP-DH) and $\overline{\alpha}_f$(SSHH). 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.

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**	$[-\infty, -0.0555]$

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.

4.3 Heuristic Classes

As noted in Chapter 3, Section 3.1, the number and implementation of the 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 the heuristic classes $\{\,\mathtt{M}, \mathtt{C}, \mathtt{R}, \mathtt{L}\,\}$. 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 two and three 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 illustrated in figure 4.1.

This methodology gives rise to 40 subsequence sets, one for each problem instance in each HyFlex domain. The subsequence sets TOP-DC, calculated over *all* the problem instances for each domain, are shown in table 4.6.

Table 4.6: 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}

Notice that some of the subsequences of heuristic classes shown in table 4.6 can also be observed in the subsequences of low level heuristics shown in table 4.2. For example, the most effective subsequence of low level heuristics in the BP domain is $\mathtt{R}_2\mathtt{R}_2$ while the most effective subsequence of heuristic classes in the BP domain is RR. The γ-ratio of each subsequence set shown in table 4.6 is shown in table 4.7. As the γ-ratios are greater than 1 each set has a negative mean unit log return $\overline{\beta} \leq 0$.

Table 4.7: 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	18495
PFS	-0.0057	72.7367	9.4039	0.7724	6.9913	11076
SAT	-0.0140	556.9779	12.6391	0.9580	40.8369	38922
PS	-0.2766	9897.0400	927.9030	0.8820	10.6545	32429

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 4.8 shows the results of the EvalHH hyper-heuristics using the subsequence set TOP-DC, and the SSHH hyper-heuristic, over 40 runs of MAX_ITER = 150 selections for each HyFlex problem.

Table 4.8: The mean final log return $\overline{\alpha}_f$ for the hyper-heuristic EvalHH using the subsequence set TOP-DC, and the hyper-heuristic SSHH. The domain statistics are calculated over 400 runs. Winning scores are shown in bold.

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 subsequence set TOP-DH outperforms the subsequence set TOP-DC overall, and on the BP, PFS and SAT domains (see tables 4.4 and 4.8). 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 [46] and [27].

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 4.8). The Wilcoxon test is used to establish whether the median differences are statistically significant. The results are shown in table 4.9.

Table 4.9: The Wilcoxon test results for $\overline{\alpha}_f$(TOP-DC) and $\overline{\alpha}_f$(SSHH). 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.

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**	$[-\infty, -0.0291]$

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 $\overline{\alpha}_f$(TOP-DC) $< \overline{\alpha}_f$(SSHH) is accepted with 99% confidence. For the BP domain the $\overline{\alpha}_f$(TOP-DC) and $\overline{\alpha}_f$(SSHH) hyper-heuristics are not statistically comparable. Overall the median difference is significant and the hyper-heuristics can be ordered stochastically so that

$$\overline{\alpha}_f(\text{TOP-DC}) < \overline{\alpha}_f(\text{SSHH})$$

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.

4.4 Cross-domain Heuristic Classes

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

1. the top 10 subsequences of length two and three with the largest γ-ratios denoted TOP-GC,
2. the bottom 10 subsequences of length two and three with the smallest γ-ratios denoted BOT-GC,
3. the set RAND consisting of subsequences of length two and three that are randomly generated at each iteration of the optimisation process, and

4. the set SINGLE containing the individual heuristic classes $\{\,\mathtt{L}, \mathtt{C}, \mathtt{R}, \mathtt{M}\,\}$.

The TOP-GC, and BOT-GC sets are selected using a γ that is calculated "globally" across all four domains in the database. 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 shown in table 4.10. As the γ-ratio is greater than 1 the subsequence set TOP-GC has a negative mean unit log return $\overline{\beta} \leq 0$.

Table 4.10: 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 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	17904
LLL	-0.2241	1580.0700	0.0000	1.0000	1580.0699	7052
RLL	-0.1710	752.5980	43.9964	0.8477	16.7257	4144
LRL	-0.1350	642.1100	97.3733	0.8014	6.5273	4034
RRL	-0.0915	290.9380	56.0398	0.8083	5.1006	2566
RL	-0.1142	1554.8600	311.9620	0.7968	4.9682	10881
CLL	-0.1059	451.3630	144.7040	0.8488	3.0978	2897
MLL	-0.0265	155.1060	59.1046	0.7131	2.5806	3628
LLR	-0.0744	462.8090	183.5720	0.7569	2.5075	3751
MMM	-0.0057	122.1290	52.1236	0.7982	2.2990	12262
All	-0.1322	10087.5530	948.8757	0.8698	10.6199	69119

Notice that all but the last two of the subsequences ends in an L. This is to be expected for two reasons. Firstly, this result supports discussions earlier in this thesis and in the literature that exploration followed by the exploitation of a local search operation is the preferable ordering of these heuristics [67]. 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 with the

Table 4.11: 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 BOT-GC subsequences.

s	$\overline{\beta}(s)$	$\beta^-(s)$	$\beta^+(s)$	$P(s \leq 0)$	γ-ratio	N_s
RCM	0.0380	7.0709	56.0581	0.3980	0.1239	1289
CR	0.1734	114.3490	1009.3100	0.6072	0.1132	5160
CCM	0.0325	5.1919	46.1179	0.4861	0.1102	1259
RC	0.1555	83.3491	861.3660	0.6283	0.0967	5002
CRM	0.0556	6.8256	75.1029	0.4083	0.0897	1227
RCC	0.1364	8.4616	125.7910	0.5977	0.0667	860
CC	0.1642	50.1554	779.8570	0.7531	0.0642	4443
CCR	0.1611	9.5609	150.3600	0.5400	0.0632	874
CRC	0.1709	9.7718	161.3600	0.5411	0.0602	887
CCC	0.1323	7.0897	120.7200	0.6845	0.0582	859
All	0.1411	301.8259	3386.0429	0.6085	0.0891	21860

smallest γ-ratios in the database, and are shown in table 4.11. As the γ-ratio is less than 1 the subsequence set has a positive mean unit log return $\overline{\beta} > 0$.

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 4.12 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 4.13.

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

$$\overline{\alpha}_f(\text{TOP-GC}) < \overline{\alpha}_f(\text{SINGLE}) < \overline{\alpha}_f(\text{RAND}) < \overline{\alpha}_f(\text{BOT-GC})$$

with 99% confidence (for each comparison).

Table 4.12: The mean final log return $\overline{\alpha}_f$, 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. 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	24873	43998
SSHH	-0.6934	-55.4024	111.8219	55.7438	36824	51643
TOP-GC	-0.6868	-53.0720	110.5556	39.1931	40241	58633
SINGLE	-0.6643	-52.9842	107.0994	107.0994	18608	30507
RAND	-0.6328	-51.5645	108.1688	40.3231	20781	30931
BOT-GC	-0.1781	-26.4057	86.6506	31.8488	5862	12404

Table 4.13: 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**	$[\infty, -0.0040]$
SINGLE - RAND	-0.0173	-0.0385	0.1439	0.0036	**0.0000**	$[\infty, -0.0111]$
RAND - BOT-GC	-0.2714	-0.4547	0.7050	0.0176	**0.0000**	$[\infty, -0.2387]$
TOP-GC - RAND	-0.0393	-0.0540	0.1252	0.0031	**0.0000**	$[\infty, -0.0307]$
TOP-GC- BOT-GC	-0.3300	-0.5087	0.7433	0.0186	**0.0000**	$[\infty, -0.3030]$

Table 4.12 and this statistical comparison illustrates the differences in the various selection mechanisms. The SSHH hyper-heuristic provides the best performance here, 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 4.12). 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 4.14). The SINGLE and RAND subsequences 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.

Table 4.14: 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.

HC	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

4.5 Analysing 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 three domains out of four. Table 4.15 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 six 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 4.10 and table 4.11. These values are calculated from subsequences of heuristic classes 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 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 4.16. 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 4.15: A domain by domain comparison of the mean final log return $\overline{\alpha}_f$ of TOP-GC and BOT-GC. The domain statistics are calculated over 400 runs. Winning scores are shown in bold. The results for SSHH are included for comparison.

Dom.	TOP-GC	BOT-GC	SSHH
BP	-0.2419	**-0.3079**	-0.3009
PFS	**-0.0051**	-0.0040	-0.0049
SAT	**-0.6567**	-0.1246	-0.6908
PS	**-1.8437**	-0.2760	-1.7770
All	**-0.6868**	-0.1781	-0.6934

Table 4.16: 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	14477
BOT-GC	-0.0162	81.3211	15.8588	0.8085	4.8236	4037

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 figure (4.2) which shows the scaled mean log returns $\overline{\alpha}$ 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 heuristic classes 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

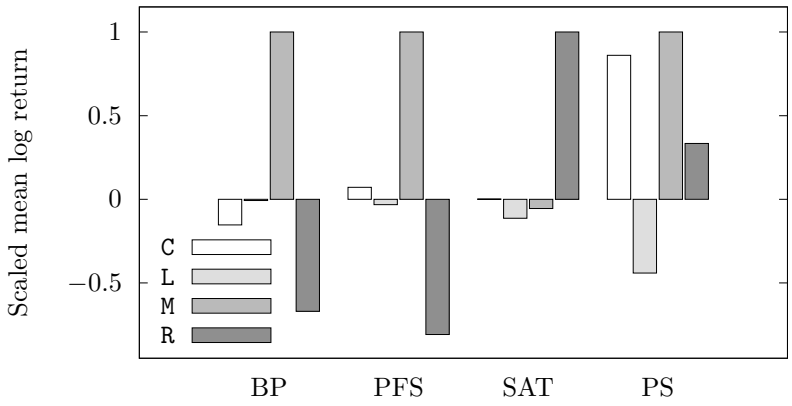


Figure 4.2: The scaled mean log returns $\overline{\alpha}$ of the heuristic classes `C, L, M`, and `R` for each domain. In each domain the $\overline{\alpha}$ values have been scaled by the largest absolute $\overline{\alpha}$ value into the interval $[-1, 1]$.

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.

4.6 Improving Cross-domain Performance

The objective of this experiment is to demonstrate that the reliability of cross-domain γ-ratio statistics and thus the performance of cross-domain subsequences can be significantly improved by a *reordering* of heuristic classes.

Figure 4.2 shows the normalised mean log returns $\overline{\alpha}$ for the HyFlex heuristic classes `C`, `L`, `M`, and `R`. The figure illustrates how the effectiveness of the heuristic classes, and the relationship between them varies by domain. As noted in Section 4.5, these changes in heuristic class behaviour between domains reduces the reliability of statistics calculated across domains.

Consider the four abstract heuristic classes `W`, `X`, `Y`, and `Z`. For each domain, the heuristic classes `C`, `L`, `M`, and `R` are mapped to the abstract classes by assigning the heuristic class with the lowest $\overline{\alpha}$ to `W`, the heuristic class with the next lowest $\overline{\alpha}$ value to `X`, and so on. Thus the $\overline{\alpha}$ histograms in figure 4.2 give rise to the four domain mappings or reorderings shown in table 4.17.

The subsequences of low level heuristic selections in each domain are mapped, to and from, subsequences of abstract heuristic classes. For example, the subsequence { `MCLLR` } from the BP domain maps to { `ZXYYW` } whereas the same subsequence in the PS domain maps to { `ZYWWX` }.

The γ-ratio is calculated over the abstract classes using a leave-one-out cross-validation methodology. Specifically, for each target domain in DB_1, the subsequences of the three remaining domains

Table 4.17: The mapping for the HyFlex heuristic classes C, L, M, and R, to the abstract heuristic classes W, X, Y, and Z, for each domain.

BP			PFS			SAT			PS		
R	$\longleftrightarrow$	W	R	$\longleftrightarrow$	W	L	$\longleftrightarrow$	W	L	$\longleftrightarrow$	W
C	$\longleftrightarrow$	X	L	$\longleftrightarrow$	X	M	$\longleftrightarrow$	X	R	$\longleftrightarrow$	X
L	$\longleftrightarrow$	Y	C	$\longleftrightarrow$	Y	C	$\longleftrightarrow$	Y	C	$\longleftrightarrow$	Y
M	$\longleftrightarrow$	Z	M	$\longleftrightarrow$	Z	R	$\longleftrightarrow$	Z	M	$\longleftrightarrow$	Z

are mapped to the abstract heuristic classes. The γ-ratio is calculated for these subsequences and the 10 subsequences with the largest γ are selected. The 10 subsequences are then mapped back into the heuristic classes of the target domain. The only domain specific information used here is encoded in the mapping from the abstract classes to the heuristic classes of the target domain. This methodology is illustrated for the BP domain in figure 4.3.

This methodology gives rise to four subsequence sets, one for each domain. The resulting subsequences, denoted WXYZ, are shown in table 4.18. In order to assess the effect of reordering, a second set of subsequences is selected using the same leave-one-out methodology, but omitting the mapping to and from the abstract classes. These subsequences are denoted CLMR, and are also shown in table 4.18.

Table 4.18: The WXYZ and CLMR subsequences for each domain.

Set	Dom.	Subsequence
WXYZ	BP	{RR, RRR, CRR, RCR, CCR, CR, LRR, RRC, MRR, CCC}
	PFS	{RR, RRR, LRR, RLR, LLR, LR, CRR, MRR, RRL, LLL}
	SAT	{LL, LLL, MLL, LML, ML, MML, CLL, RLL, LLM, MLM}
	PS	{LL, RL, LLL, LR, RRL, RLR, LRL, RLL, LLR, LRR}
CLMR	BP	{LL, LLL, RLL, LRL, MMM, RRL, RL, MML, MLL, CLL}
	PFS	{LL, LLL, RLL, LRL, RRL, RL, CLL, MMM, MLL, LLR}
	SAT	{LL, LLL, RLL, LRL, RL, RRL, CLL, LLR, MLL, RLR}
	PS	{LL, RRR, LLL, RLR, LRR, RR, RRL, CL, LC, CRR}

The subsequences sets WXYZ and CLMR are used to parametrise EvalHH and evaluated on each target domain. Table 4.19 shows the results of EvalHH, over 40 runs of MAX_ITER = 150 selections for each HyFlex problem. The results of the subsequence sets TOP-DC, and TOP-GC are included for comparison. The low level heuristics and heuristic parameters are chosen at random.

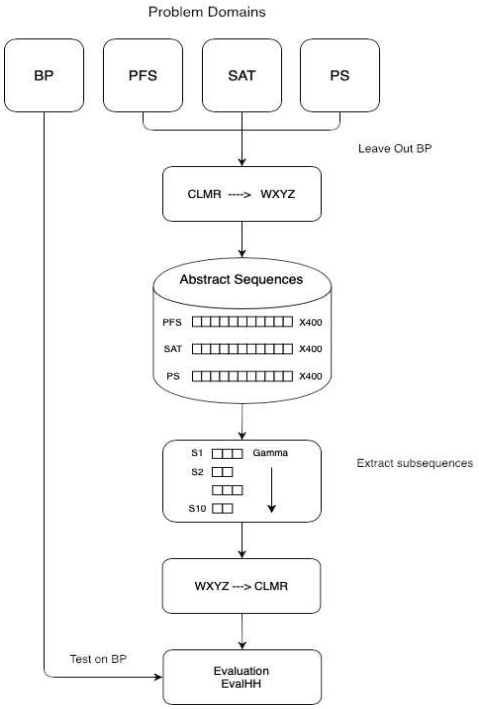


Figure 4.3: A flow chart of the leave-one-out cross-validation methodology used to evaluate the performance of subsequences of abstract heuristic classes in the BP domain.

The Wilcoxon test is used to establish whether the median differences observed are statistically significant. The results are shown in table 4.20. For each pair of hyper-heuristics the null hypothesis is rejected as the p-value is less than 0.01 and the alternative hypothesis $\overline{\alpha}_f(\text{A}) < \overline{\alpha}_f(\text{B})$ is accepted with 99% confidence. Thus the median differences are significant and the hyper-heuristics can be stochastically ordered so that

$$\overline{\alpha}_f(\text{TOP-DC}) < \overline{\alpha}_f(\text{WXYZ}) < \overline{\alpha}_f(\text{TOP-GC}) < \overline{\alpha}_f(\text{CLMR})$$

with 99% confidence (for each comparison).

The results show that, despite a higher overall $\overline{\alpha}_f$, the cross-domain subsequences WXYZ are statistically inferior to the TOP-DC subsequences. This is to be expected, as the TOP-DC subsequences are selected using statistics that are calculated for each target domain, while the WXYZ subsequences are selected using statistics calculated from the other three domains. However, the difference in performance is relatively small, with WXYZ winning the BP and PFS domains. The WXYZ subsequences outperform the TOP-GC subsequences overall, and on the BP, PFS, and SAT domains. This is notable because the TOP-GC subsequences are selected using statistics calculated across all four domains. The TOP-GC subsequences outperform the CLMR subsequences which like WZYZ are selected using statistics

Table 4.19: A domain by domain comparison of the mean final log return $\overline{\alpha}_f$ of TOP-DC, WXYZ, TOP-GC, and CLMR. The domain statistics are calculated over 400 runs. Winning scores are shown in bold.

Dom.	TOP-DC	WXYZ	TOP-GC	CLMR
BP	-0.2896	**-0.3093**	-0.2419	-0.2270
PFS	-0.0053	**-0.0054**	-0.0051	-0.0051
SAT	**-0.8275**	-0.8128	-0.6567	-0.6149
PS	-1.8378	-1.7989	**-1.8437**	-1.7559
All	-0.7109	**-0.7316**	-0.6868	-0.6507

Table 4.20: The hyper-heuristic pair, the sample median difference $\widehat{d}$, the sample mean difference $\overline{d}$, 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	$\widehat{d}$	$\overline{d}$	SD	SEM	p-value	Conf. Int.
TOP-DC - WXYZ	-0.0017	-0.0085	0.0875	0.0022	**0.0010**	$[\infty, -0.0003]$
WXYZ - TOP-GC	-0.0336	-0.04476	0.1244	0.0031	**0.0000**	$[\infty, -0.0253]$
TOP-GC - CLMR	-0.0212	-0.0361	0.1015	0.0025	**0.0000**	$[\infty, -0.0165]$

calculated across three domains. The difference in performance between TOP-GC and CLMR demonstrates the importance of domain specific information, while the difference in performance between WXYZ and CLMR demonstrates the benefits of reordering the heuristic classes.

These results show that it is possible to select cross-domain subsequences of heuristic classes that are almost as effective on an unseen target domain, as subsequences selected using domain specific information.

4.7 Long Subsequences of Heuristic Classes

The previous experiments concentrated on short subsequences of length two and three. 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 [81]), it was decided to include subsequences of length three. In this section the performance of a set of subsequences of arbitrary length chosen using the γ-ratio is examined.

Table 4.21: 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	17904
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	33974

The 10 subsequences of heuristic classes with the largest γ-ratio are selected using a γ that is calculated over all of the subsequences in the database. This subsequence set is denoted LONG-GC and is shown in table 4.21. It should be noted that the database contains subsequences 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 4.10).

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 class would perform. With this in mind, EvalHH is also run with the singleton subsequence sets {L} and {LL}. Table 4.22 contains the results for the hyper-heuristic SSHH and 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 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 4.2). 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 4.14).

Table 4.22 shows that a well chosen set of subsequences such as LONG-GC or TOP-GC outperform the single heuristic selections SINGLE, demonstrating the benefits of a sequence-based approach. Furthermore, combinations of several individual heuristics such as SINGLE outperform the single heuristics {L} and {LL} confirming again the result observed in [46] and [27]. Finally the single heuristics {L} and {LL} are superior to the random and badly chosen subsequences sets RAND and BOT-GC.

Table 4.22: 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	24873	43998
TOP-DH	-0.7714	-59.2435	97.5250	49.1031	62632	141847
WXYZ	-0.7316	-57.1712	102.9000	37.1613	50691	79530
TOP-DC	-0.7109	-47.2025	96.0275	37.4350	65249	99218
SSHH	-0.6934	-55.4024	111.8219	55.7438	36824	51643
LONG-GC	-0.6932	-52.3682	107.3506	22.5325	52455	80678
TOP-GC	-0.6868	-53.0720	110.5556	39.1931	40241	58633
SINGLE	-0.6643	-52.9842	107.0994	107.0994	18608	30507
{L}	-0.6536	-46.4370	88.3656	88.3656	37807	81847
{LL}	-0.6535	-46.4447	89.3356	44.3863	38767	83240
CLMR	-0.6507	-51.9381	109.4056	39.8425	89465	142424
RAND	-0.6328	-51.5645	108.1688	40.3231	20781	30931
BOT-GC	-0.1781	-26.4057	86.6506	31.8488	5862	12404

4.8 Pareto Ranking

The concept of *Pareto efficiency* [87] was introduced by the Italian engineer and economist Vilfredo Pareto (1848–1923) and is commonly used in multi-objective optimisation. Pareto efficiency can be defined as a state of allocation of resources amongst a number of objectives in which it is impossible to make any one objective better off without making at least one other objective worse off. In an engineering context, given a set of parameter choices and a method of evaluating them, the *Pareto front* is the subset of parameters that are Pareto efficient.

Figure 4.4 provides an alternative view of the results presented in table 4.22 which illustrates the performance trade-offs between SSHH and the other parameterisations. The dashed lines denote the Pareto front that each hyper-heuristic parameterisation belongs to, where Pareto efficiency decreases from left to right. If the Pareto fronts in each plot are numbered from 0 to n, where front-0 is the most efficient, and front-n is the least, then each parameterisation can be evaluated by calculating its *Pareto rank* which is the sum of its Pareto front indices. For example, SSHH belongs to Pareto front 2 for minimum selections, Pareto front 1 for objective evaluations, and Pareto front 0 for time to a minimum and total run time, which gives an overall rank of $2 + 1 + 0 + 0 = 3$. This ranking, or *nondominated sorting* procedure was introduced in [109].

The Pareto rank of each parameterisation in figure 4.4 is shown in table 4.23. The ranking demonstrates that TOP-DH is the best performing parameterisation when all four criteria are considered.

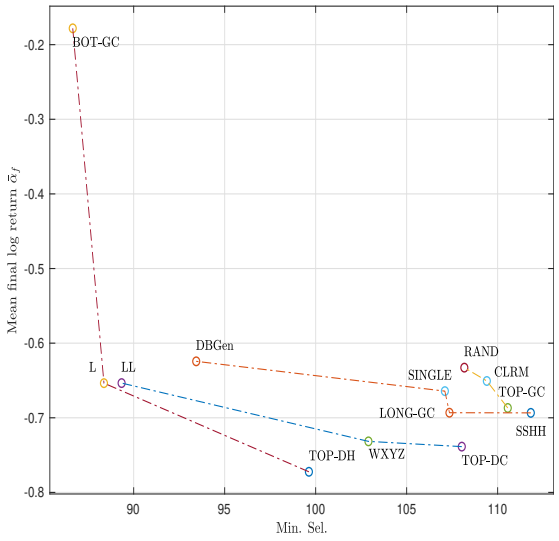
Notice that the BOT-GC parameterisation which exhibits the worst optimisation performance (see table 4.22) is ranked (joint) second. This can be attributed to the low run-times of the BOT-GC subsequences which are chosen to be ineffective. The subsequence's poor performance also gives rise to lower mean numbers of selections to a minimum, and objective function evaluations, as such subsequences are less likely to improve on a solution as the optimisation process progresses. Overall, the rankings strengthen the thesis that well chosen subsequence sets such TOP-DH, TOP-DC, and WXYZ can be used to improve optimisation performance across a number of criteria when compared to other sets, such as SINGLE, RAND, and CLRM.

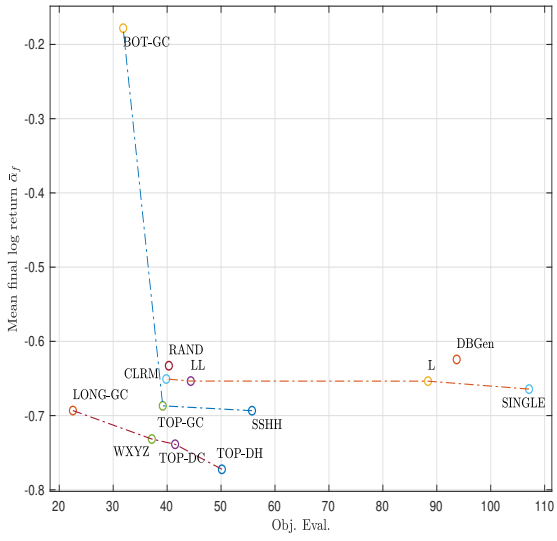
Table 4.23: The Pareto rank of each hyper-heuristic parameterisation.

Hyper-heuristic	Min. Sel.	Obj. Eval.	Min. T	Total T	Rank
TOP-DH	0	0	0	0	0
TOP-DC	1	0	0	0	1
WXYZ	1	0	0	0	1
BOT-GC	0	1	0	0	1
SSHH	2	1	0	0	3
SINGLE	2	2	0	0	4
LONG-GC	2	0	1	1	4
{L}	0	2	1	2	5
TOP-GC	3	1	1	1	6
{LL}	1	2	2	3	8
RAND	3	3	1	1	8
DBGen	2	4	2	2	10
CLRM	3	2	3	4	12

(a) $\overline{\alpha}_f$ versus Min. Sel.

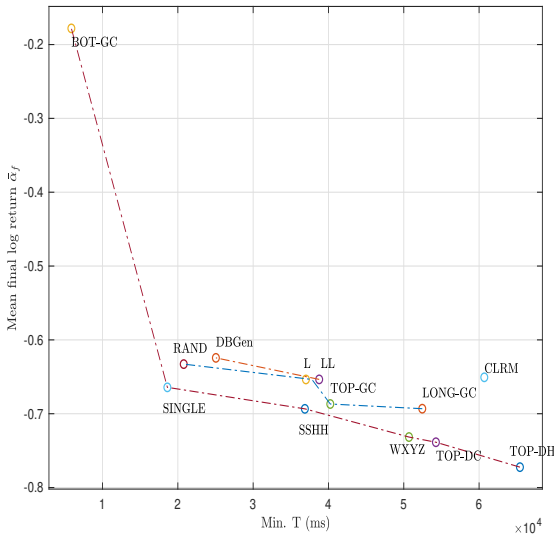
(b) $\overline{\alpha}_f$ versus Obj. Eval.





(c) $\overline{\alpha}_f$ versus Min. T.

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



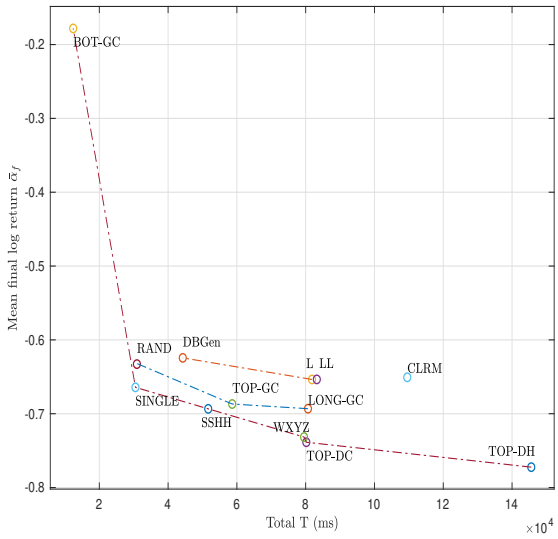


Figure 4.4: The mean final log return $\overline{\alpha}_f$ plotted against (a) the mean number of selections Min. Sel., (b) the mean number of objective function evaluations Obj. Eval. required to find a minimum, (c) the mean time Min. T required to find a minimum, and (d) the mean overall run time Total T, for the HyFlex domains.

4.9 Increasing the 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 minutes of wall clock time, using a leave-one-out methodology; a total of 400 runs. The time length of 10 minutes was chosen for computational feasibility. The results are shown in table 4.24. 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.

Table 4.24: A domain by domain comparison of the mean final log return $\overline{\alpha}_f$ of TOP-DH and BOT-DH. Winning scores are shown in bold.

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

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 thesis is also applicable to longer run lengths.

4.10 Conclusions

This chapter has applied the statistical framework developed in Chapter 3 to the analysis of subsequences of low level heuristics and heuristic classes.

The γ-ratio is used to select sets of subsequences of heuristic selections from the offline database DB_1. 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 HyFlex problem instances. The SSHH hyper-heuristic is known to perform well on the HyFlex problems, and this result demonstrates that offline selected subsequences can outperform online subsequence learning. Furthermore, the TOP-DH subsequences show that the expected exploration-exploitation

behaviour in sequences can be seen in some, but not all domains.

The offline selected subsequences of low level heuristics are able to outperform SSHH because, in this case, the chosen subsequences are effective over the whole optimisation process. Thus, there is no need to adapt the heuristic choices that constitute a subsequence during optimisation. In contrast, SSHH has no *a priori* knowledge of a particular problem, and must spend the initial stages of the optimisation process learning which subsequences are effective. However, when the performance of the offline selected subsequences changes significantly during the optimisation process, online learning must be used to adapt them so that the subsequences continue to be effective.

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 three out of the four problem domains, and that this improvement in performance is also statistically signifiant.

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. This results demonstrates that in order to choose effective cross-domain subsequences of heuristic classes, the classes must exhibit similar performance characteristics in each domain. For example, if the $\overline{\alpha}$-order of the heuristic classes varies across the problem domains, then what has been learned about a class on one domain cannot necessarily be transferred to another.

The performance of cross-domain subsequences of heuristic classes can be significantly improved by reordering the classes. By mapping subsequences of heuristic selections to and from subsequences of abstract heuristic classes it is possible to choose cross-domain subsequences that are almost as effective on an unseen target domain, as subsequences selected using domain specific information.

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 seven 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 BOT-DH subsequences for 10 minutes 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.

These experiments have 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 are novel, and 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 small in the experiments presented here, it could open the possibility of hybrid offline and online selection hyper-heuristics trained for specific domains.

Chapter 5

Hybrid Learning

As noted in Chapter 2, hyper-heuristics can be classified according to the nature of the learning mechanisms used (or not) to improve optimisation performance [36]. Specifically, a hyper-heuristic may employ no learning, online learning which uses feedback during the optimisation process, or offline learning which is performed on a distinct set of benchmark problems prior to optimisation. There are also examples in the literature of selection hyper-heuristics incorporating mixed or hybrid learning which combines offline and online learning such as [23], [114], and [108].

The PHunter hyper-heuristic described in [23] mimics a traditional diving method for perl hunting. The hyper-heuristic employs two groups of low level heuristics called "moves" and "dives". Moves correspond to crossover, mutate, and ruin and recreate heuristics, while dives correspond to local search heuristics. The heuristics in a group are determined using rules that are obtained by offline learning. During optimisation PHunter employs two online mechanisms; an unwanted (inferior to tabu) list and, a restart mechanism which resets the search procedure when the pool of current solutions is over-converged, or no better solutions are found for a certain amount of time. The PHunter hyper-heuristic has been tested on the HyFlex problem domains, and the results demonstrate that PHunter is competitive with other well known hyper-heuristics coming fourth overall in the CHeSC 2011[1] competition.

In [114] the authors present a framework that hybridises an online reinforcement learning hyper-heuristic with offline population based incremental learning (PBIL) [122]. The proposed methodology consists of two phases. In the first phase, SPBIL (a PBIL variant) is used offline to construct a posterior probability distribution of promising solutions from statistical information derived from a population of solutions. In the second phase, the best probability distributions are used as low-level heuristics for the reinforcement learning hyper-heuristic. Their approach successfully combines offline and online learning mechanisms to provide "good average performance" over a number of dynamic environment

[1] The HyFlex Cross-domain Heuristic Search Challenge (CHeSC 2011) (`http://www.asap.cs.nott.ac.uk/chesc2011/`).

problems.

In [108] the authors investigate the performance of a selection hyper-heuristic with online and offline learning on a number of course timetabling problems. The hyper-heuristic is an adaptive version of the iterated local search algorithm [76], and uses a set of probabilities to select operators from a set of low level heuristics. These selection probabilities are learned using online and offline methods. In the online case, a preliminary (offline) set of experiments are used to determine the initial probabilities used by the heuristic selection mechanism. The probabilities are then adapted online during the optimisation process. In the offline case, the selection probabilities are static parameters which are learned by the ParamILS algorithm [62] from a separate set of training instances. The paper also proposes a methodology for the offline selection of the set of heuristics employed by the hyper-heuristic from a larger pool of low level heuristics. Specifically, each heuristic in the pool is evaluated on a set of training problems, and those with the best scores are chosen. The results demonstrate that the online learning methodology statistically outperforms the offline learning methodology, and produces competitive results when compared to the state-of-the-art.

In each of these examples, the offline and online learning concerns individual heuristic selections. In this chapter the potential of combining *sequence-based* online and offline learning is investigated. The objective is to demonstrate that hybrid learning of effective subsequences can also lead to significant improvements in optimisation performance across a number of problem domains.

The SSHH hyper-heuristic, which has online learning capabilities, is trained offline with the Baum-Welch learning algorithm (see Chapter 2). The parameters to be estimated are SSHH's state transition and heuristic emission probability matrices, and the training observations are subsequences of effective heuristics. The results of executing the trained SSHH on unseen HyFlex problems are compared with those of an untrained SSHH, in order to demonstrate empirically that the offline trained SSHH is able to outperform the untrained SSHH across a number of problems and domains.

Although the primary motivation for combining offline and online learning is to improve optimisation performance in general, such hybrid learning could also be useful when applying hyper-heuristics to large computationally expensive problems. In these cases, a hyper-heuristic could be trained offline on a number of small, time efficient instances of a problem, before being applied to larger problems.

This chapter contains five hybrid learning experiments. In the first experiment the SSHH hyper-heuristic's hidden Markov model (HMM) is trained offline with the Baum-Welch learning algorithm, using effective subsequences of low level heuristics, selected using the γ-ratio. In the second experiment, the concept of *Pareto efficiency* is used to select effective subsequences of low level heuristics that are also time efficient. The first two experiments depend on statistics calculated over a large database of heuristic selections and objective function values. The third experiment assesses the effect on performance when offline learning with small data sets. The fourth experiment attempts to separate out and quantify the effects of online and offline learning, and to examine what has been learned by the Baum-Welch algorithm. All of the preceding experiments employ a run length of 150 heuristic

selections for reasons of computational feasibility, 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, by optimising each problem for 10 minutes of wall clock time.

The experiments in this chapter are conducted on the larger DB_2 offline learning database (see Chapter 3) and follow the methodology employed in Chapter 4. However, the method used here differs in two respects. Firstly, the VRP and TSP domains are used. Secondly, the DBGen hyper-heuristic which is used to generate DB_2, and the SSHH hyper-heuristic which is used to evaluate hybrid learning, employ a modified cross-over mechanism. Specifically, the cross-over pool uses five solutions as opposed to one, and this larger pool size improves the performance of the cross-over heuristics. The pool size of five solutions was taken from AdapHH [38]. The inclusion of the two unseen domains, and the changes in subsequence statistics due to the change to DBGen provide a test of the robustness of the methodology introduced in Chapter 3.

The abbreviations for the hyper-heuristic parameterisations used throughout this chapter together with their descriptions are summarised in table 5.1.

Table 5.1: Hyper-heuristic parameterisation abbreviations and descriptions.

Abbr.	Description	Sec.
DBGen	Hyper-heuristic used to generate the learning database.	C2–2.5
SSHH	The untrained hyper-heuristic used for comparisons.	C2–2.5
T-SSHH	Trained with effective subsequences (by domain).	5.1
T-SSHH_N	As above, but with no online learning.	5.4
P-SSHH	Trained with a Pareto set of subsequences (by domain).	5.2
P-SSHH_N	As above, but with no online learning.	5.4
S-SSHH	Trained using a single sequence (by domain).	5.3
S-SSHH_N	As above, but with no online learning.	5.4
A-SSHH	The adverse hyper-heuristic to T-SSHH (by domain).	5.4
A-SSHH_N	As above, but with no online learning.	5.4
NONE	No offline or online learning.	5.4

5.1 An Effective Set of Subsequences

In this experiment, the SSHH hyper-heuristic is trained offline with the Baum-Welch learning algorithm on effective subsequences chosen using the γ-ratio. The objective is to demonstrate that the Baum-Welch algorithm is able to learn and generalise from these well chosen subsequences of low level heuristics across a number of problem domains. In this context, generalisation means that the trained SSHH hyper-heuristic is able to significantly outperform the untrained SSHH hyper-heuristic when evaluated on unseen example problems.

The methodology employed here to select effective subsequences, and evaluate their potential as training inputs to the Baum-Welch algorithm is based on the method introduced in Chapter 3, and is virtually identical to that used to select and evaluate the TOP-DH subsequences in Chapter 4. The method here differs in that the training subsequences are evaluated using Baum-Welch and the SSHH hyper-heuristic instead of the EvalHH hyper-heuristic.

The training sets consists of the 10 subsequences of low level heuristics in DB_2 with the largest γ-ratio in each domain. The number of 10 subsequences was chosen to ensure that the subsequences contained sufficient heuristic "diversity", that is the subsequences consist of more that just one or two low level heuristics. The subsequences themselves are chosen solely according to their domain and γ-ratio. The training sets are constructed from these effective subsequences using a *leave-one-out* cross-validation methodology [9]. Recall that there are 10 problem instances of interest in each of the six HyFlex domains. For each target problem in a domain, γ is calculated from the sequences of low level heuristics from the remaining nine problems. The subsequences are then used to train SSHH which is evaluated on the target problem. This ensures that the training subsequences are always evaluated on a problem that is unseen. The methodology is illustrated in figure 5.1.

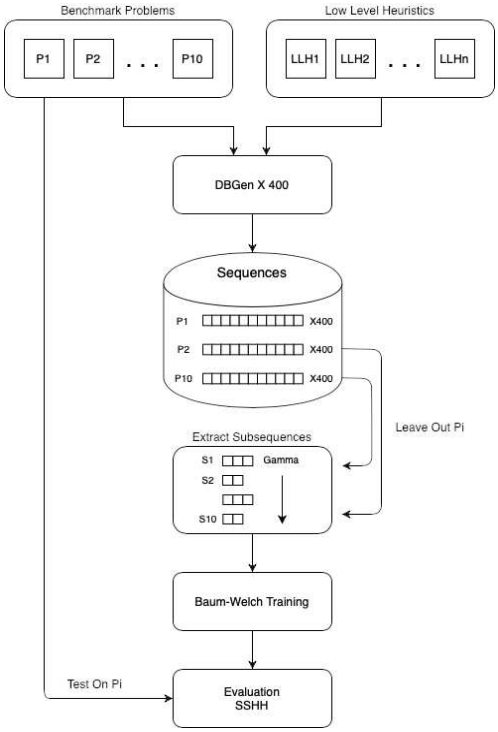


Figure 5.1: A flow chart of the leave-one-out cross-validation methodology used to evaluate hybrid learning in a given domain.

This methodology gives rise to 60 training sets, one for each problem instance in each HyFlex domain. As the γ statistics are quite stable, the selected training subsequences for the problems in a domain are very similar, usually differing by at most one or two subsequences. Some example training subsequence sets for each domain are shown in table 5.2.

Table 5.2: The effective training subsequences of low level heuristics for problem 5 in each domain of DB_2. The subsequences are listed left to right in descending γ-ratio order.

Dom.	Subsequences
BP	R_2R_2, M_3R_2, R_2L_4, R_2L_6, L_6R_2, R_2M_3, L_4R_2, R_2C_7, C_7R_2, M_3C_7
PFS	L_8L_7, $L_{10}L_8$, $L_{10}L_7$, L_8R_6, L_7L_{10}, R_6L_8, R_6L_{10}, M_1L_8, L_7L_7, $C_{14}L_8$
SAT	M_1M_0, M_0M_1, L_8L_7, M_0M_0, L_7M_0, M_1M_1, M_0L_7, L_7L_8, L_7M_1, L_7L_7
VRP	L_4L_8, L_8L_4, L_8L_8, C_6L_4, L_4L_4, M_0L_4, L_8L_9, M_0L_8, $C_6L_4L_8$, C_5L_8
TSP	R_5L_8, L_6L_8, L_7L_7, L_7L_8, L_6L_7, R_5L_6, L_7L_6, $C_{12}L_8$, R_5L_7, L_8L_6
PS	L_4L_4, L_0L_4, L_3L_4, L_4L_2, L_2L_4, L_3L_2, L_3L_0, L_3L_3, L_4C_9, L_0L_3

The untrained SSHH hyper-heuristic is initialised with an equiprobable state transition matrix and an identity heuristic emission matrix. This ensures that, initially, each hidden state has an equiprobable chance of being selected, and can only emit a single low level heuristic. For the trained SSHH, before Baum-Welch training, SSHH's hidden Markov model's state transition and heuristic emission probability matrices are randomised. The randomisation improves Baum-Welch learning which often fails when using equiprobable state transition and identity heuristic emission matrices. After training, the matrices are edited to ensure that every state transition and heuristic emission has a (small) non-zero probability. The trained SSHH hyper-heuristic is initialised with these edited matrices. In both cases, the initial state distribution, and the parameter and acceptance check emission matrices are initialised to be equiprobable.

The results of evaluating the offline trained and untrained SSHH hyper-heuristic, 40 times on each HyFlex problem for 150 iterations, are denoted T-SSHH and SSHH respectively, and are shown in table 5.3. It should be noted that both T-SSHH and SSHH employ online learning during optimisation.

Table 5.3: The mean final log return $\overline{\alpha}_f$, 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), over the HyFlex domains. The averages are calculated over 2400 runs.

	$\overline{\alpha}_f$	%	Min. Sel.	Obj. Eval.	Min. T	Total T
T-SSHH	**-0.5495**	**-50.5216**	**106.3154**	**48.7646**	59291	98306
SSHH	-0.5051	-47.9762	109.3271	54.4071	**28714**	**39601**

Table 5.4: A domain by domain comparison of the mean final log return $\overline{\alpha}_f$ and standard deviation SD of T-SSHH and SSHH. The averages are calculated over 400 runs. The boldface type indicates a win.

Dom.	T-SSHH	SD	SSHH	SD
BP	**-0.3820**	0.2342	-0.3396	0.2186
PFS	**-0.0091**	0.0041	-0.0078	0.0036
SAT	**-0.8701**	0.1886	-0.6739	0.1752
VRP	**-0.3247**	0.0711	-0.3071	0.0870
TSP	**-0.0728**	0.0192	-0.0678	0.0196
PS	**-1.6382**	0.4193	-1.6344	0.4265

The results demonstrate a marked improvement in the mean final log return $\overline{\alpha}_f$ for the T-SSHH hyper-heuristic at the cost of a significant increase in run time, even though the numbers of heuristic selections and objective function evaluations are lower. The results, broken down by domain, are shown in table 5.4. They demonstrate that T-SSHH is superior to SSHH on each of the six domains.

The differences in final log returns are tested for statistical significance using the Wilcoxon signed rank test as explained in Chapter 3, Section 3.5, and the results are shown in table 5.5.

Table 5.5: The sample median difference $\hat{d}$, the sample mean difference $\bar{d}$, the standard deviation SD, the p-value, and the interval within which the population median falls with 99% confidence for T-SSHH and SSHH. Statistically significant results are shown in boldface.

Dom.	$\hat{d}$	$\bar{d}$	SD	p-value	Conf. Int.
BP	-0.0277	-0.0424	0.1456	**0.0000**	$[-\infty, -0.0129]$
PFS	-0.0012	-0.0013	0.0018	**0.0000**	$[-\infty, -0.0010]$
SAT	-0.1958	-0.1961	0.1312	**0.0000**	$[-\infty, -0.1800]$
VRP	-0.0282	-0.0176	0.0842	**0.0000**	$[-\infty, -0.0186]$
TSP	-0.0043	-0.0051	0.0078	**0.0000**	$[-\infty, -0.0035]$
PS	-0.0041	-0.0038	0.1404	0.0177	$[-\infty, \ 0.0005]$
All	-0.0232	-0.0444	0.1251	**0.0000**	$[-\infty, -0.0187]$

The differences in median are significant overall and for five of the six domains. In the PS domain, a two tailed test confirms that the difference in hyper-heuristic performance is not statistically significant. Thus the hyper-heuristics can be stochastically ordered so that

$$\overline{\alpha}_f(\text{T-SSHH}) < \overline{\alpha}_f(\text{SSHH})$$

with 99% confidence. That is, the optimisation performance of the offline trained hyper-heuristic T-SSHH is significantly better than the untrained hyper-heuristic SSHH on the HyFlex domains.

This result is important because it demonstrates that the Baum-Welch algorithm is able to learn and generalise from well chosen subsequences of low level heuristics across five of the six HyFlex domains, including the VRP and TSP domains. The process of generalisation across the problems of a domain has generated a hyper-heuristic that is able to perform better than an untrained hyper-heuristic on unseen test problems. This shows that useful information can be learned about the problems in a domain from the sequences of heuristic selections used to optimise them.

5.2 A Pareto Set of Subsequences

Although the T-SSHH hyper-heuristic demonstrates a significant improvement in optimisation performance over SSHH, it is at the cost of a significant increase in run time. The run time increases because the effective subsequences used to offline train T-SSHH typically contain the low level heuristics with the highest computational costs. In this experiment the training observations are taken from Pareto fronts of heuristic subsequences chosen to be effective *and* time efficient.

Each subsequence s of length n is evaluated according to its negative γ-ratio[2] and its mean unit execution time. In symbols,

$$-\gamma(s) \quad \text{and} \quad \frac{\overline{T}(s)}{n}.$$

Following Section 5.1, the training sets are constructed using a leave-one-out cross-validation methodology. For each target problem in a domain, the training set consists of the 10 subsequences on the Pareto front with the largest γ-ratio, constructed from the subsequences of the remaining nine problems. Examples of the resulting training sets are shown in table 5.6. Notice that in each domain,

Table 5.6: The P-SSHH training subsequences of low level heuristics for problem 5 in each domain.

Dom.	Subsequences
BP	R_2R_2, R_1R_2, R_2R_1, $M_3R_2R_1$, $M_5R_2R_2$, $R_1M_0R_2R_1$, $R_2M_3R_2R_1$, $L_6R_2R_1L_6R_2$, $R_2R_1L_6R_2R_2$, $R_2M_5M_0R_1R_1$
PFS	L_8L_7, $L_{10}L_8$, L_7L_{10}, $L_{10}L_7$, $L_{10}L_9$, $C_{14}L_{10}$, R_5R_5, $R_5R_5M_1$, $M_4C_{12}C_{11}$, $R_5R_5M_1C_{12}$
SAT	M_1M_0, M_0M_1, L_8L_7, $C_{10}L_7$, L_8L_8, C_9L_8, L_8C_{10}, C_9M_3, $C_{10}M_4$, $C_9C_9M_3$
VRP	L_4L_8, L_8L_8, L_8L_9, M_0L_8, L_9L_8, M_1L_8, L_8M_0, L_8M_1, C_6M_0, M_0C_6
TSP	R_5L_8, L_7L_7, L_7L_6, $C_{11}L_7$, L_6C_{11}, $L_8C_9C_9$, $C_{10}M_2L_7$, $C_9C_{10}L_6$,$L_6C_{12}M_4$, $R_5L_6C_{12}M_4$
PS	L_4L_4, L_0L_4, L_4L_0, L_0L_1, L_1L_0, L_0R_5, R_5L_0, L_0L_0, $C_9C_9C_8C_9$, $C_8C_9R_5C_{10}$

the first subsequences (with the largest γ-ratio) are the same as the first T-SSHH subsequence shown in table 5.2. The Pareto fronts, illustrated in figure 5.2, show the trade-offs between subsequence run time and optimisation performance, and demonstrate that, in general, better performing subsequences take longer to execute. However, it should be noted that for some subsequences, such as $C_{14}L_{10}$, R_5R_5

[2]In this case the preferred direction of criteria values is "less than" and so the negative γ-ratio is used.

in PFS, and $\mathtt{C_9M_3}$, $\mathtt{C_{10}M_4}$ in SAT, the increase in performance is small when compared to the increase in run time.

These subsequences are used to offline train the SSHH hyper-heuristic's HMM. The overall results of evaluating the trained and untrained SSHH hyper-heuristic, 40 times on each HyFlex problem for 150 iterations, denoted P-SSHH and SSHH respectively, are shown in table 5.7. It should be noted that both P-SSHH and SSHH employ online learning during optimisation.

Table 5.7: The mean final log return $\overline{\alpha}_f$, 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), over the HyFlex domains. The averages are calculated over 2400 runs.

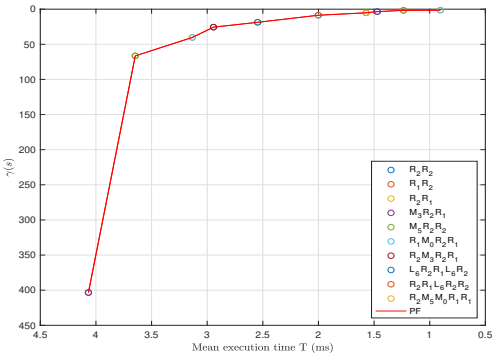
	$\overline{\alpha}_f$	%	Min. Sel.	Obj. Eval.	Min. T	Total T
SSHH	**-0.5051**	-47.9762	109.3271	54.4071	28714	39601
P-SSHH	-0.5040	**-48.3740**	**100.4454**	**49.3379**	**10741**	**13556**

The results demonstrate that the trained hyper-heuristic P-SSHH is significantly faster than the untrained SSHH with only a slight reduction in overall mean final log return $\overline{\alpha}_f$. Table 5.8 shows these results broken down by domain. They demonstrate that the P-SSHH hyper-heuristic's performance is inferior to SSHH on four out of the six domains.

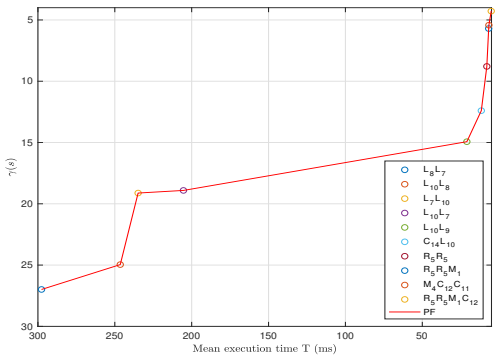
Table 5.8: A domain by domain comparison of the mean final log return $\overline{\alpha}_f$ and standard deviation SD of SSHH and P-SSHH. The averages are calculated over 400 runs. The boldface type indicates a win.

Dom.	SSHH	SD	P-SSHH	SD
BP	-0.3396	0.2186	**-0.3654**	0.2398
PFS	**-0.0078**	0.0036	-0.0077	0.0035
SAT	-0.6739	0.1752	**-0.7718**	0.1800
VRP	**-0.3071**	0.0870	-0.2859	0.0834
TSP	**-0.0678**	0.0196	-0.0669	0.0203
PS	**-1.6344**	0.4265	-1.5264	0.4064

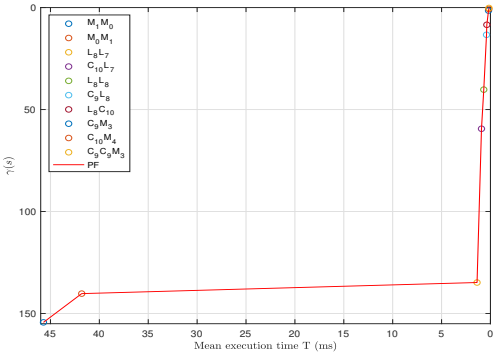
(a) BP



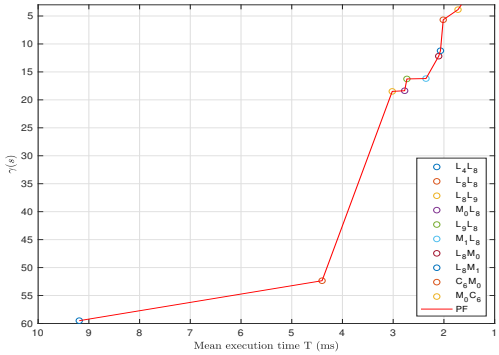
(b) PFS



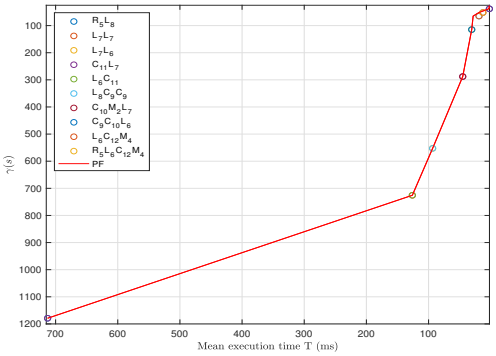
(c) SAT



(d) VRP



(e) TSP



(f) PS

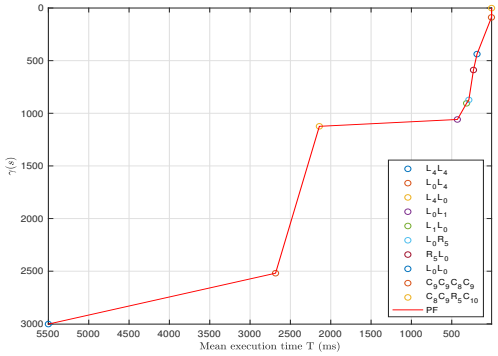


Figure 5.2: The Pareto fronts of the offline training subsequences for problem 5 in each domain.

The Wilcoxon signed rank test is used to establish whether the differences observed in the mean final log returns $\overline{\alpha}_f$ are statistically significant. The results are shown in table 5.9.

Table 5.9: The sample median difference $\hat{d}$, the sample mean difference $\bar{d}$, the standard deviation SD, the p-value, and the interval within which the population median falls with 99% confidence for SSHH and P-SSHH. Statistically significant results are shown boldface.

Dom.	$\hat{d}$	$\bar{d}$	SD	p-value	Conf. Int.
BP	0.0120	-0.0258	0.1240	0.9981	$[-\infty,\ 0.0238]$
PFS	-0.0001	0.0001	0.0019	0.1651	$[-\infty,\ 0.0001]$
SAT	0.1015	-0.0979	0.1262	1.0000	$[-\infty,\ 0.1160]$
VRP	-0.0167	0.0211	0.0578	**0.0000**	$[-\infty, -0.0105]$
TSP	-0.0007	0.0008	0.0077	**0.0067**	$[-\infty,\ 0.0000]$
PS	-0.0690	0.1080	0.2338	**0.0000**	$[-\infty, -0.0552]$
All	-0.0006	0.0011	0.1363	0.1412	$[-\infty,\ 0.0005]$

For the PFS domain, the result of a two tailed test yields a p-value of 0.0134 and so the differences in median for this domain is not significant. The P-SSHH hyper-heuristic outperforms SSHH on the BP and SAT domains while on the VRP, TSP, and PS domains the situation is reversed. The result of a two tailed test on the overall performance yields a p-value of 0.2824 and so the SSHH and P-SSHH hyper-heuristics are statistically similar. In symbols

$$\overline{\alpha}_f(\text{P-SSHH}) = \overline{\alpha}_f(\text{SSHH})$$

with 99% confidence.

This result demonstrates that the Baum-Welch algorithm is also able to learn and generalise from subsequences chosen to optimise a target problem and, at the same time, reduce run time. The process of generalisation across the problems of a domain can generate a hyper-heuristic that is able to perform (almost) as well as an untrained hyper-heuristic on unseen test problems while requiring approximately 60% less run time.

5.3 Generalisation with Small Data Sets

The T-SSHH and P-SSHH experiments employ a leave-one-out methodology and depend on statistics calculated over 400 sequences for each problem instance. For many real world applications such large data sets are not available. With this in mind, in the following experiment, the SSHH hyper-heuristic is trained on a *single* sequence of 150 heuristic selections and objective function values. For each run of a target problem, a single sequence of heuristic selections is chosen at random from the nine remaining

problems in that domain. The sequence is broken down into subsequences, and the γ-ratio is calculated for these subsequences. The 10 subsequences with the largest γ-ratio are then used to offline train the SSHH hyper-heuristic. The results of evaluating the trained SSHH hyper-heuristic are denoted S-SSHH, and are shown in table 5.10.

Table 5.10: The mean final log return $\overline{\alpha}_f$, 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), over the HyFlex domains. The averages are calculated over 2400 runs.

	$\overline{\alpha}_f$	%	Min. Sel.	Obj. Eval.	Min. T	Total T
T-SSHH$_N$	**-0.5526**	**-50.7784**	108.1142	53.1367	52490	89594
T-SSHH	-0.5495	-50.5216	106.3154	**48.7646**	59309	98473
S-SSHH	-0.5341	-48.8615	**100.0025**	49.2575	59990	92629
S-SSHH$_N$	-0.5301	-48.6753	103.0500	50.6963	54461	80558
SSHH	-0.5051	-47.9762	109.3271	54.4071	28714	39601
P-SSHH	-0.5040	-48.3740	100.4454	49.3379	**10741**	**13556**
P-SSHH$_N$	-0.4908	-47.1289	103.6579	51.1042	21395	29056
A-SSHH	-0.4846	-46.2657	108.3900	52.4358	23174	31452
NONE	-0.4802	-45.9636	106.6500	52.6696	24789	33486
A-SSHH$_N$	-0.4652	-44.8768	106.3429	52.3975	20723	28006

Unsurprisingly, the S-SSHH hyper-heuristic performance is inferior to T-SSHH as it employs far less statistical information. However, it is significantly superior to the untrained SSHH hyper-heuristic (see table 5.11). In symbols

$$\overline{\alpha}_f(\text{S-SSHH}) < \overline{\alpha}_f(\text{SSHH})$$

with 99% confidence. This demonstrates the utility of the γ-ratio for selecting effective subsequences of low level heuristics, even when there are only relatively small amounts of offline data available.

5.4 An Analysis of Learning

In this section, the results of evaluating an untrained SSHH hyper-heuristic on unseen HyFlex problems are compared with nine distinct parameterisations: T-SSHH, P-SSHH, S-SSHH, T-SSHH$_N$, P-SSHH$_N$, S-SSHH$_N$, A-SSHH, A-SSHH$_N$, and NONE (see tables 5.1 and 5.10). The objective of these experiments is to separate out and quantify the effects of online and offline learning, and then to examine what has been learned online by SSHH, and offline by the Baum-Welch learning algorithm.

5.4.1 Hyper-heuristic Parameterisation

The nine hyper-heuristic parameterisations considered in this section have the following characteristics. The T-SSHH$_N$, P-SSHH$_N$, and S-SSHH$_N$ parameterisations denote the results of offline training the SSHH hyper-heuristic with the T-SSHH, P-SSHH, and S-SSHH subsequences, and evaluated *without* online learning. Here, the N subscript indicates that no online learning takes place. The A-SSHH hyper-heuristic uses the offline trained T-SSHH probability matrices after they have been transformed using the function

$$x'_{ij} = \frac{1 - x_{ij}}{n - 1}$$

where n is the dimension of the matrix under consideration. This transformation maps high probabilities to low probabilities and *vice versa*. The intention is to produce a model that is the adverse of T-SSHH, that is a model that tends to do the "opposite" of T-SSHH by *not* selecting effective subsequences. The A-SSHH$_N$ hyper-heuristic uses the same probability matrices as A-SSHH but is evaluated without online learning. The NONE hyper-heuristic is a version of SSHH with no offline or online learning which selects heuristics, their parameters, and makes acceptance checks equiprobably. The NONE hyper-heuristic is used as a baseline for comparison purposes.

5.4.2 Comparing Optimisation Performance

The results of evaluating SSHH and the nine hyper-heuristic parameterisations on unseen HyFlex problems using a leave-one-out methodology are shown in table 5.10, and illustrated in figure 5.3. The results of the Wilcoxon signed rank test are shown in table 5.11.

Table 5.11: The sample median difference $\hat{d}$, sample mean difference $\bar{d}$, the standard deviation SD, the p-value, and the interval within which the population median difference falls with 99% confidence.

	$\hat{d}$	$\bar{d}$	SD	p-value	Conf. Int.
T-SSHH$_N$ - T-SSHH	-0.0003	-0.0031	0.0736	0.0649	$[-\infty,\ 0.0002]$
T-SSHH - S-SSHH	-0.0066	-0.0153	0.1041	**0.0000**	$[-\infty, -0.0048]$
S-SSHH - S-SSHH$_N$	-0.0006	-0.0040	0.0647	0.0175	$[-\infty,\ 0.0001]$
S-SSHH$_N$ - SSHH	-0.0063	-0.0250	0.1105	**0.0000**	$[-\infty, -0.0043]$
SSHH - P-SSHH	-0.0005	-0.0011	0.1363	0.1412	$[-\infty,\ 0.0005]$
P-SSHH - P-SSHH$_N$	-0.0021	-0.0132	0.1314	**0.0002**	$[-\infty, -0.0007]$
P-SSHH$_N$ - A-SSHH	-0.0037	-0.0062	0.0955	**0.0000**	$[-\infty, -0.0016]$
A-SSHH - NONE	-0.0005	-0.0045	0.0841	0.0555	$[-\infty, -0.0003]$
NONE - A-SSHH$_N$	-0.0055	-0.0149	0.0759	**0.0000**	$[-\infty, -0.0039]$

Using the results in table 5.10 and 5.11, the hyper-heuristic parameterisations can be stochastically

ordered so that when online learning is enabled

$$\overline{\alpha}_f(\text{T-SSHH}) < \overline{\alpha}_f(\text{S-SSHH}) < \overline{\alpha}_f(\text{P-SSHH}) < \overline{\alpha}_f(\text{A-SSHH})$$

and when online learning is disabled

$$\overline{\alpha}_f(\text{T-SSHH}_\text{N}) < \overline{\alpha}_f(\text{S-SSHH}_\text{N}) < \overline{\alpha}_f(\text{P-SSHH}_\text{N}) < \overline{\alpha}_f(\text{NONE}) < \overline{\alpha}_f(\text{A-SSHH}_\text{N})$$

with 99% confidence (for each comparison).

These inequalities, together with the results of table 5.10 and table 5.11, demonstrate that the T-SSHH and T-SSHH_N parameterisations are the best performing optimisers in terms of objective function minimisation. A two tailed Wilcoxon test indicates that these hyper-heuristics are statistically similar. In symbols

$$\overline{\alpha}_f(\text{T-SSHH}) = \overline{\alpha}_f(\text{T-SSHH}_\text{N})$$

with 99% confidence. Furthermore, the T-SSHH and S-SSHH parameterisations which combine online and offline learning, and the T-SSHH_N and S-SSHH_N parameterisations which employ only offline learning, significantly outperform SSHH. The P-SSHH parameterisation and SSHH are statistically similar (see Section 5.2), while P-SSHH_N is inferior to SSHH. These results demonstrate that, with the exception of P-SSHH, offline learning is able to significantly improve optimisation performance with or without online learning. The results for the adverse hyper-heuristics A-SSHH and A-SSHH_N demonstrate that the performance of T-SSHH and T-SSHH_N are not just a consequence of some fortuitous non-equiprobable parameterisation. The result that A-SSHH and NONE are statistically similar, while NONE outperforms A-SSHH_N shows that SSHH's online learning algorithm is able to adapt to poor initial parameterisations.

Figure 5.3 provides an alternative view of the results presented in table 5.10 which illustrates the performance trade-offs between SSHH and the nine parameterisations. The figure shows that, with the exception of the adverse parametrisation A-SSHH, offline learning decreases the number of selections required to find a minimum, and the number of objective function evaluations. However, with the exception of P-SSHH, the reductions in selections and objective function evaluations does not lead to a reduction in the time to a minimum, or overall run time. This is because the offline training subsequences are chosen to be effective, and so tend to consist of time expensive heuristics. In contrast, the adverse parameterisation A-SSHH is constructed to select ineffective subsequences which tend to consist of computationally inexpensive heuristics. The reduction in time to a minimum and overall run time for the P-SSHH parameterisation is because it is trained with subsequences that are effective *and* time efficient. Figure 5.3 also shows that, with the exception of T-SSHH, online learning always improves optimisation performance.

The dashed lines in figure 5.3 denote the Pareto front that each hyper-heuristic parameterisation belongs to, where Pareto efficiency decreases from left to right. If the Pareto fronts in each plot are numbered from 0 to n, where front-0 is the most efficient, and front-n is the least, then each

parameterisation can be evaluated by calculating its *Pareto rank* which is the sum of its Pareto front indices. For example, SSHH belongs to Pareto front 2 for minimum selections, Pareto front 3 for objective evaluations, and Pareto front 0 for time to a minimum and total run time, which gives an overall rank of 2+3+0+0 = 5. The Pareto rank for each parameterisation in figure 5.3 is shown in table 5.12. The ranking demonstrates that when the four criteria of selections to minima, objective function evaluations, run time to minima, and overall run time are taken into consideration, the offline trained hyper-heuristics T-SSHH$_N$ and S-SSHH$_N$, and the hybrid learning hyper-heuristics T-SSHH, P-SSHH, and S-SSHH outperform SSHH. Furthermore, SSHH outperforms the adverse parameterisation A-SSHH, with or without online learning, the offline trained P-SSHH$_N$, and the no learning hyper-heuristic NONE. This ranking demonstrate the potential for offline and hybrid learning to improve hyper-heuristic performance.

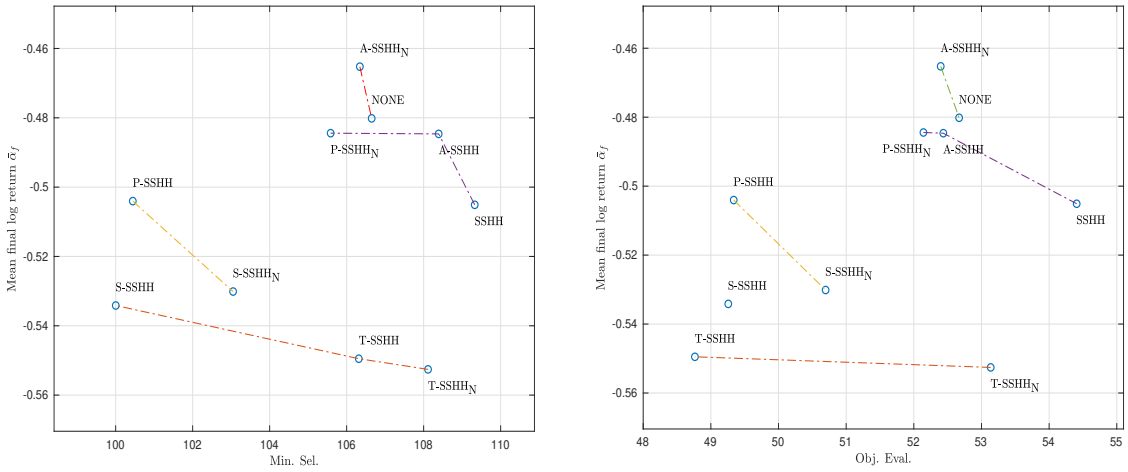
Table 5.12: The Pareto rank of each hyper-heuristic.

Hyper-heuristic	Min. Sel.	Obj. Eval.	Min. T	Total T	Score
T-SSHH$_N$	0	0	0	0	0
T-SSHH	0	0	1	1	2
P-SSHH	1	2	0	0	3
S-SSHH	0	1	2	1	4
S-SSHH$_N$	1	2	1	0	4
SSHH	2	3	0	0	5
A-SSHH	2	3	1	1	7
P-SSHH$_N$	2	3	1	1	7
A-SSHH$_N$	3	4	1	1	9
NONE	3	4	2	2	11

Figure 5.4 plots the mean log return of the best solution found against the number of iterations for run 0 of problem 5 in each domain for SSHH, T-SSHH, and P-SSHH. The plots show that, in this example, T-SSHH converges faster than P-SSHH and SSHH in all six domains, leading to the best solutions in all but the PS domain. Furthermore, for the BP and SAT domains, and to a lesser extent PFS, both offline trained hyper-heuristics converge faster than the untrained SSHH.

(a) $\overline{\alpha}_f$ versus Min. Sel.

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



(c) $\overline{\alpha}_f$ versus Min. T.

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

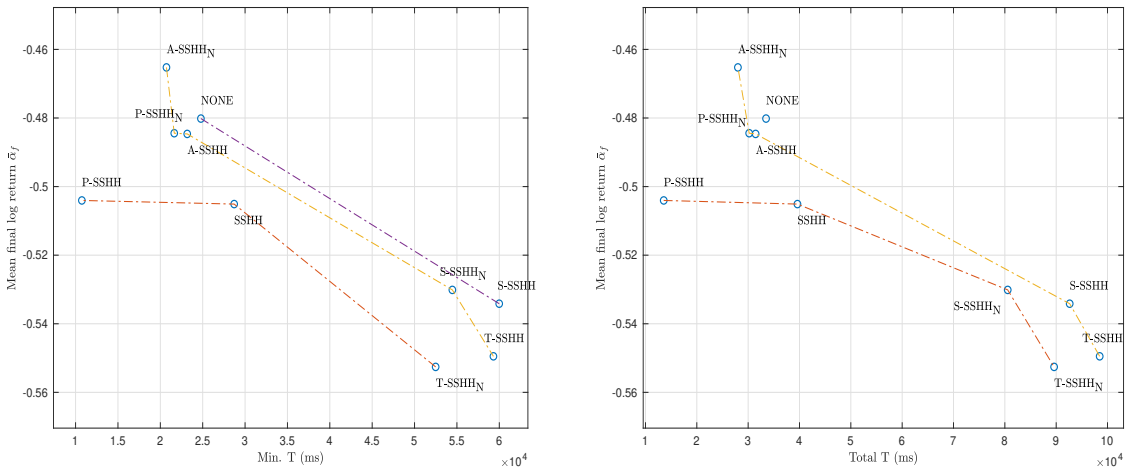
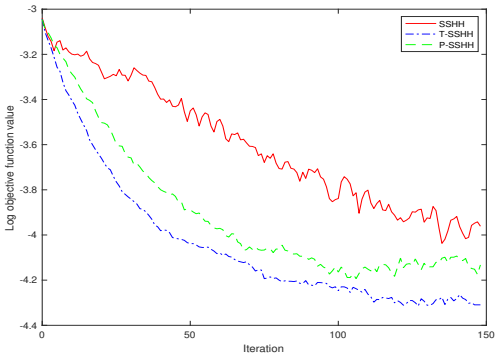
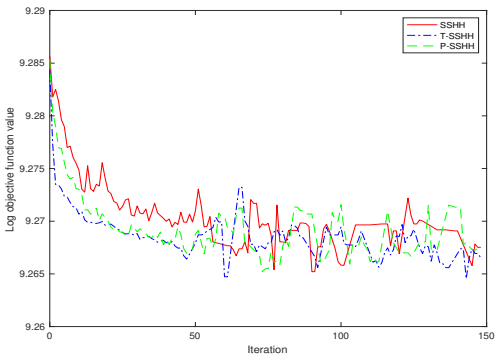


Figure 5.3: The mean final log return $\overline{\alpha}_f$ plotted against (a) the mean number of selections Min. Sel., (b) the mean number of objective function evaluations Obj. Eval. required to find a minimum, (c) the mean time Min. T required to find a minimum, and (d) the mean overall run time Total T.

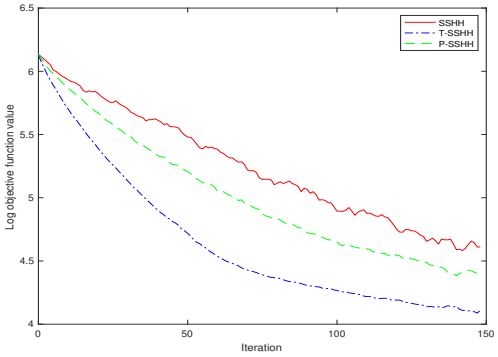
(a) BP



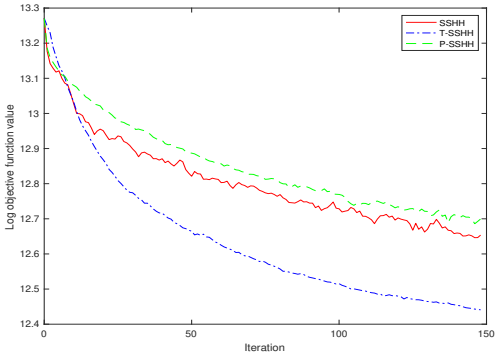
(b) PFS



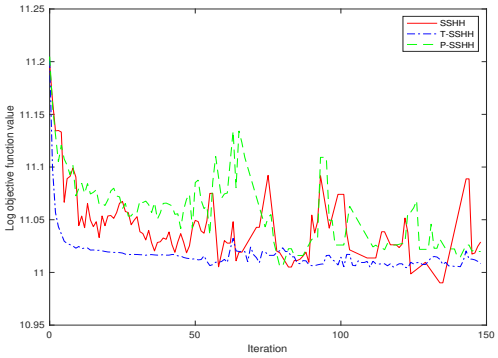
(c) SAT



(d) VRP



(e) TSP



(f) PS

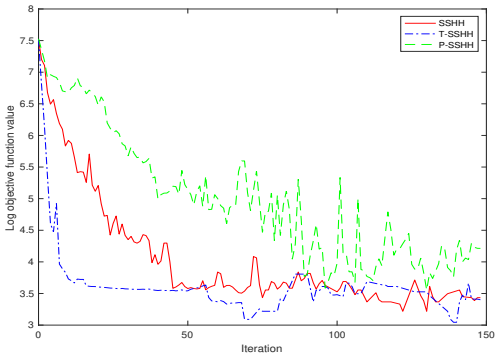


Figure 5.4: The best mean log objective function value of SSHH, T-SSHH, and P-SSHH for run 0 of problem 5 for each domain.

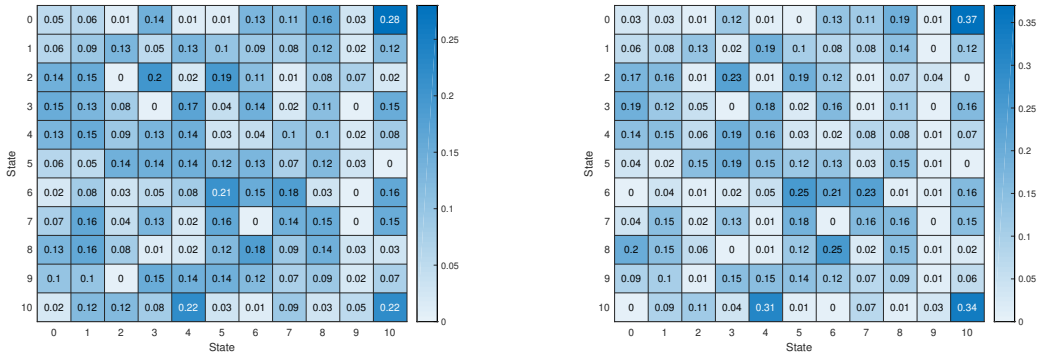
5.4.3 Heat Map Analysis

Section 5.4.2 presented a comparative analysis of SSHH and nine parameterisations that demonstrates that offline learning is able to significantly improve optimisation performance. This section presents an examination *what* has been learned.

An insight into what the Baum-Welch algorithm has learned can be gained by examining SSHH's state transition and heuristic emission matrices. Figure 5.5 shows heat maps of the transition and emission matrices for the T-SSHH trained hyper-heuristic for problem 5 in the SAT domain, before and after optimisation with online learning.

(a) Transition matrix before.

(b) Transition matrix after.



(c) Emissions matrix before.

(d) Emissions matrix after.

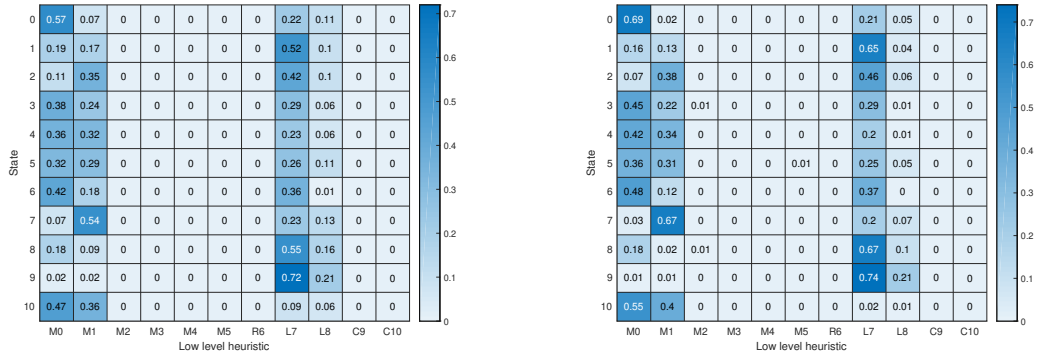


Figure 5.5: A heat map of the average transition and emission probability matrices for T-SSHH on problem 5 in the SAT domain before and after 150 iterations. The probabilities are averaged over 40 runs.

As expected, the emissions matrix contains relatively high probabilities for the $\mathtt{M}_0$, $\mathtt{M}_1$, $\mathtt{L}_7$, and $\mathtt{L}_8$ low level heuristics that occur in the T-SSHH training subsequences for the SAT domain (see table

5.2). This demonstrate that the Baum-Welch algorithm is able learn from (or load) the training subsequences. Furthermore, notice that the before and after matrices are very similar; they have not been significantly altered by the online learning algorithm. This is because the Baum-Welch algorithm has found a probabilistic maxima from which the online learning algorithm is unable to escape. These maxima can also be observed in the probability matrices learned in the other HyFlex domains.

5.5 Increasing the Run Length

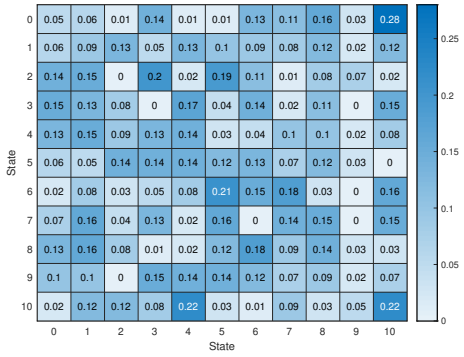
The previous experiments all use a run length of 150 low level heuristic selections. For some problems with computationally efficient heuristics, 150 iterations is a relatively small number over which to evaluate a hyper-heuristics's performance. Thus, the question arises as to whether the observed gains in performance are still present after longer execution times. In this section, the SSHH hyper-heuristic, and the offline trained hyper-heuristic T-SSHH are evaluated (using a leave-one-out methodology) for 10 times on each of the 60 HyFlex problems instances for 10 minutes of wall clock time. The results are shown in table 5.13.

Table 5.13: A domain by domain comparison of the mean final log return $\overline{\alpha}_f$ of SSHH and T-SSHH over 10 minutes. The averages are calculated over 10 and 100 runs. The boldface type indicates a win.

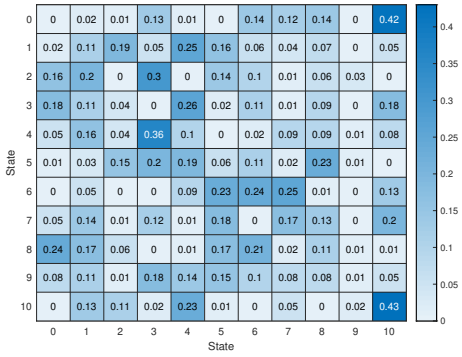
Dom.	SSHH	T-SSHH
BP	-0.9823	**-1.0294**
PFS	**-0.0143**	-0.0139
SAT	-1.0452	**-1.1004**
VRP	-0.4701	**-0.4705**
TSP	-0.0892	**-0.0893**
PS	**-1.7045**	-1.6549
All	-0.7176	**-0.7264**

The T-SSHH hyper-heuristic outperforms the SSHH hyper-heuristic overall, and on four of the six HyFlex domains. However, the improvements in mean final log return $\overline{\alpha}_f$ are modest, particularly for the VRP and TSP domains where the gains are so small that they are unlikely to be significant. This demonstrates that although offline training can lead to improvements in performance, especially early on in the optimisation process, the online learning algorithm can, given enough time, perform (almost) as well. Figure 5.6 shows heat maps of the transition and emission matrices for the T-SSHH trained hyper-heuristic for problem 5 in the SAT domain, before and after 10 minutes of optimisation with online learning. Notice that the before and after matrices are very similar, and are also very similar to the matrices produced after 150 iterations shown in figure 5.5. This strengthens the conclusion that the Baum-Welch algorithm has found a probabilistic maxima.

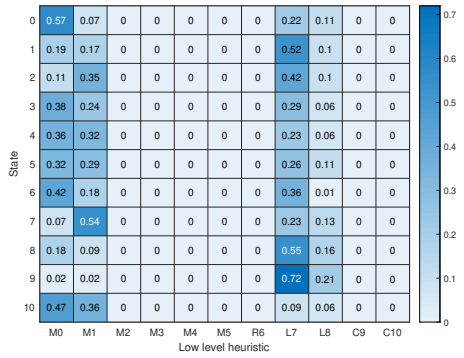
(a) Transition matrix before.



(b) Transition matrix after.



(c) Emissions matrix before.



(d) Emissions matrix after.

Figure 5.6: A heat map of the average transition and emission probability matrices for T-SSHH on problem 5 in the SAT domain before and after a 10 minute run. The probabilities are averaged over 10 runs.

5.6 Conclusions

This chapter has presented a methodology for combining the online learning capabilities of the SSHH hyper-heuristic, with offline learning of effective subsequences of heuristics using the Baum-Welch algorithm.

The methodology, based on the γ-ratio, is able to significantly improve the optimisation performance of the SSHH hyper-heuristic on the HyFlex problems with 99% confidence; a problem set on which SSHH is known to perform well. The same methodology can also produce a hyper-heuristic that when trained with subsequences taken from a Pareto front, has similar optimisation performance to the untrained SSHH hyper-heuristic, but with more than a 60% improvement in run time.

The two preceding experiments demonstrate that it is possible to learn and generalise from effective sets of subsequences. However, these experiments rely on large amounts of offline data. The γ-ratio can also be used to significantly improve performance in situations where there is only relatively small amounts of offline data available.

An analysis of a number of SSHH parameterisations when online learning is enabled or disabled shows that the performance of the SSHH hyper-heuristic is significantly affected by the initial configuration of the probability matrices. Furthermore, although online learning generally improves optimisation performance, the offline learning algorithm produces parameterisations that are local maxima from which the online learning algorithm is unable to escape. Further research into balancing the contribution of offline and online learning so that neither algorithm overwhelms the other is needed.

Finally, the observed improvements in optimisation performance due to offline learning can also be seen in longer runs.

These results demonstrate the utility of combining online and offline subsequence learning. One application of such hybrid learning could be in real-world scenarios where multiple problem instances exist within a problem domain. In these cases SSHH could be trained offline on computationally simple instances of a problem to determine the initial biases towards certain heuristics. Once trained, the SSHH hyper-heuristic could then be used on computationally expensive, real-world instances, benefiting from the domain-specific training, but using online learning to adapt to the specifics of such instances.

Chapter 6

A Case Study: Water Distribution Networks

In this chapter, the methodology developed and evaluated on the HyFlex problems in Chapters 3, 4 and 5 is applied to a novel problem domain; the optimisation of water distribution networks (WDN). Optimising the design and rehabilitation of water distribution networks is an important real-world problem. A WDN delivers water from reservoirs, tanks, and water treatment facilities to consumers via a network of pipes and makes use of pumps and valves to meet consumer demand. The WDN optimisation problem is characterised as a discrete NP-hard combinatorial optimisation problem with large scale multi-modal search landscapes [69].

A variety of meta-heuristics [10] have been successfully applied to this problem such as simulated annealing [28], shuffled frog leaping algorithms [42], harmony search [52], honey-bee mating optimisation [83], differential evolution [129], particle swarm optimisation [43], multi-objective evolutionary algorithms [120], and ant colony optimisation [105]. To date, hyper-heuristics have received relatively little attention in the WDN optimisation literature [69] [128].

This chapter investigates the optimisation of the 12 WDN problems presented in [120] with the sequence-based selection hyper-heuristic SSHH described in Chapter 2. The optimisation performance of the SSHH hyper-heuristic is compared with the five multi-objective evolutionary algorithms (or MOEAs) used in [120]. Following Chapter 5, the SSHH hyper-heuristic is then trained offline using the Baum-Welch learning algorithm [95] and an appropriate training set of heuristic subsequences. The training subsequences are chosen from an offline learning database using the statistical methodology introduced in Chapter 3. The statistical methodology is then used to analyse the results of offline learning, which leads to an improved offline learning strategy. Finally, the potential for *scalable learning*, where knowledge learned from a small problem is usefully transferred to a second larger problem, is explored.

This chapter presents four results. Specifically,

1. a hyper-heuristic with online learning is competitive with the state-of-the-art across 12 WDN problems of varying size,
2. offline learning can improve on online learning performance,
3. the effectiveness of the heuristics changes markedly during the optimisation process for WDN problems, and
4. knowledge learned from small WDN problems can be transferred to larger ones, raising the potential for high performance algorithms trained on benchmarks and deployed on large, real-world problems.

The structure of this chapter is as follows. Section 6.1 contains an overview of the water distribution network problems presented in [120] and considered here. Section 6.2 describes the methodology used to conduct the optimisation experiments. In Section 6.3, the SSHH hyper-heuristic is used to optimise the 12 WDN problems, and these results are compared with those produced by the MOEA's employed in [120]. In Section 6.4, in order to improve optimisation performance the SSHH hyper-heuristic's hidden Markov model (HMM) is trained offline with the Baum-Welch learning algorithm. However, in this case, offline learning fails to improve optimisation performance. With this in mind, Section 6.5 presents an analysis of the results of offline learning, which leads, in Section 6.6, to an improved offline learning methodology. In Section 6.7 the concept of probabilistic length, and the Kullback-Leibler distance are used to visualise the changes in the probability distribution encoded in the HMM due to online and offline learning. Finally, in Section 6.8, the potential for scalable learning is explored.

6.1 Water Distribution Networks

Water distribution networks are an important element of urban infrastructure as they convey clean water from reservoirs, tanks and water treatment works to homes and businesses via a set of pipes. The design for a WDN aims to ensure a supply of clean water to the demand nodes at sufficient pressure and for minimum monetary cost. Although cost minimisation is the primary design objective, there are many other objectives that can also be considered such as minimisation of water age, adherence to water pressure and velocity constraints, and increasing the robustness of the network to reduce supply outages.

In this chapter a simple WDN optimisation problem is considered. The discrete decision variables are the diameters of the pipes in a network. The objectives are to minimise the network's overall cost and maximise the network's reliability, while meeting all demand constraints (or loading conditions), and maintaining the minimum required head (pressure) throughout the network.

This section is structured as follows. Section 6.1.1 describes the 12 networks, Section 6.1.2 specifies the design objectives, and Section 6.1.3 defines the single valued objective function used in this case study. Section 6.1.3 also defines the solution points that are used to compare optimisation performance. Lastly, in Section 6.1.4, the six low level heuristics that are used to optimise the WDN problems are presented.

6.1.1 The Networks

The 12 WDN problems considered here are taken from [120]. Table 6.1 shows a summary of the problems including the number of loading conditions, water sources, decision variables, and pipe diameter options. The problems are categorised into four groups; small (S), medium (M), intermediate (I), and large (L) according to the size of search space, and range from small benchmark instances with tens of pipes to large-scale city-wide networks comprised of thousands of pipes.

The problems differ from one another in a number of other respects. Of the 12 problems, 11 are based on real world networks, while the TLN network [3] is an example of a hypothetical network. The TRN [53], BAK [73], NYT [102] and EXN [45] networks are expansion problems, where the task is to extend an existing network by modifying *some* of the pipes in the network. In addition to selecting pipe diameters such problems can sometimes make use of extra options such pipe cleaning, pipe duplication, or "leaving alone". The remaining problems TLN, BLA [104], HAN [47], GOY [70], FOS, PES, MOD [13], and BIN [96], are pure design problems where the diameters of any or all of the pipes in a network can be modified.

Each problem has minimum head pressure requirements for the demand nodes. The BLA, FOS, PES, and MOD networks also have maximum pressure requirements, and upper bounds on water velocities in the pipes. The TRN network differs from the other problems in that it has three sets of loading conditions, while the BIN network, unlike a typical WDN, has a fixed level of water consumption across all demand nodes.

Figure 6.1 shows an example schematic of the Blacksburg distribution network.

6.1.2 The Design Objectives

In this case study, the water distribution network design problem is specified by three objective functions: the cost C, the head pressure deficit H, and the network's *resilience* I_n. The cost and head pressure deficit are to be minimised while the resilience is to be maximised.

The monetary cost is usually expressed in millions and is defined by

$$C = \sum_{i=1}^{np} U_c(D_i)L_i \tag{6.1}$$

where U_c is the unit pipe cost which depends on the diameter D_i selected, and the length L_i of pipe $i = 1, \ldots, np$.

Table 6.1: The problem name, acronym, the number of loading conditions (LC), number of water sources (WS), number of decision variables (DV), number of pipe diameter options (PD), for the water distribution network problems. For the TRN problem, three existing pipes have eight diameter options for duplication and the two extra options of cleaning or leaving alone.

Problem	Acronym	LC	WS	DV	PD	Search Space	Size
Two-Reservoir	TRN	3	2	8	8^*	3.28×10^7	S
Two-Loop	TLN	1	1	8	14	1.48×10^9	S
BakRyan	BAK	1	1	9	11	2.36×10^9	S
New York	NYT	1	1	21	16	1.93×10^{25}	M
Blacksburg	BLA	1	1	23	14	2.30×10^{26}	M
Hanoi	HAN	1	1	34	6	2.87×10^{26}	M
GoYang	GOY	1	1	30	8	1.24×10^{27}	M
Fossolo	FOS	1	1	58	22	7.25×10^{77}	I
Pescara	PES	1	3	99	13	1.91×10^{110}	I
Modena	MOD	1	4	317	13	1.32×10^{353}	L
Balerma	BIN	1	4	454	10	1.00×10^{455}	L
Exeter	EXN	1	7	567	11	2.95×10^{590}	L

The head pressure deficit is defined by

$$H = \sum_{j=1}^{nn} \Big(\max(H_j - H_j^{\max}, 0) + \max(H_j^{\text{req}} - H_j, 0) \Big) \tag{6.2}$$

where H_j is the actual head pressure, H_j^{req} is the minimum required head pressure, and $H_j^{\max}$ is the maximum required head pressure (if any) for each demand node $j = 1, \ldots, nn$.

Network resilience measures the redundancy of a pipe network, and maximising this indicator can improve network reliability. It has been shown that using a network resilience index as an additional objective reduces the occurrence of nonviable networks, and yields solutions which are more robust under pipe failure conditions [94]. There are however many network resilience measures in the literature, and each has its own particular advantages and disadvantages (see for example [7]). Each resilience measure attempts to mimic the design goal of designing reliable loops with practicable pipe diameters, while providing excess head pressure above the minimum required at all nodes. In this chapter, following [120], a network's resilience is defined by

$$I_n = \frac{\sum_{j=1}^{nn} C_j Q_j (H_j - H_j^{\text{req}})}{\Big(\sum_{k=1}^{nr} Q_k H_k + \sum_{i=1}^{npu} \frac{P_i}{\gamma} \Big) - \sum_{j=1}^{nn} Q_j H_j^{\text{req}}}. \tag{6.3}$$

where Q_j is the demand, nr is the number of reservoirs, Q_k is the discharge, and H_k is actual head of reservoir k, npu is the number of pumps, P_i is the power of pump i (if any), and γ is the specific

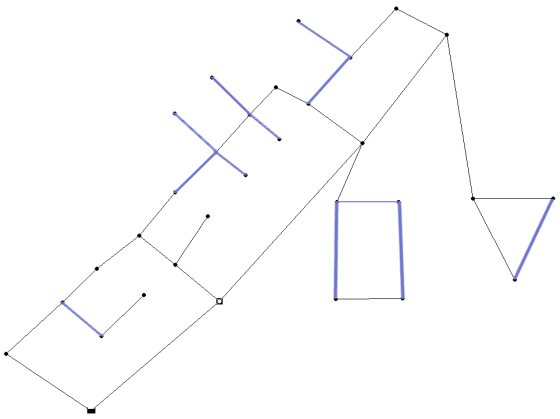


Figure 6.1: The layout of the Blacksburg network (BLA). The network consists of thirty-five pipes of which twelve have fixed diameters, one reservoir with a fixed head of 715.56m, and thirty demand nodes. Fixed pipes are shown as blue lines.

weight of water. The term C_j is the *uniformity* which is defined by

$$C_j = \frac{\sum_{i=1}^{npj} D_i}{npj \max\{D_i\}} \tag{6.4}$$

where npj is the number of pipes connected to node j and D_i is the diameter of pipe i connected to node j. The *EPANET2*[1] software library [100] is used to run hydraulic simulations in order to obtain the variables required for the calculation of a network's resilience.

6.1.3 Comparing Solutions

The solutions produced by the SSHH hyper-heuristic are compared with those of the five multi-objective evolutionary algorithms (or MOEAs) used in [120]. It should be emphasised that the objective of this study is not to demonstrate that SSHH is a superior optimiser to the MOEAs. Rather, it is to show that SSHH is a computationally efficient optimisation algorithm capable of producing high quality solutions comparable to the state-of-the-art, and that these solutions can be improved upon with offline learning.

As the SSHH hyper-heuristic employed in this case study is a single objective optimiser, the three quantities of cost, head pressure deficit, and resilience are combined to define the single value objective function

$$f = aC + bH + (-cI_n) \tag{6.5}$$

[1]The EPANET2 software and manual can be downloaded from:
`https://www.epa.gov/water-research/epanet/`

that is to be minimised. The constants $a = 200$, $b = 1000$ and $c = 5$ are chosen to ensure that the objective function f is always positive across *all* 12 problems, as opposed to "tuning" them for each problem. The objective function favours low cost networks with low head pressure deficits, as solutions with non-zero deficits are considered to be nonviable.

An MOEA is a multi-objective optimiser that operates on a population of solutions. Such algorithms naturally generate a Pareto front of "best" solutions, which make the trade-offs between the conflicting objectives of cost and resilience explicit. Although SSHH operates on a single solution, it can also generate a Pareto front. However, the use of a single objective value forces the algorithm to explore a smaller region of the solution space; the region of low cost, and therefore lower resilience networks. As a result, comparing the Pareto fronts produced by the two methods is unhelpful. Instead, the SSHH and MOEA optimisers are compared on a single point on the published Pareto front for each problem; the solutions with the cheapest monetary cost for each problem.

Table 6.2 shows the cheapest viable solutions, their resilience, and the number of objective function evaluations used to generate them, taken from the Pareto fronts of C and I_n presented[2] in [120]. The Pareto fronts were generated by executing the five MOEAs 30 times, on each problem for the number of iterations shown.

Table 6.2: The minimum cost C (M) and resilience I_n for the viable solutions of the WDN problems found by the MOEAs, and the number of objective function evaluations used to generate them [120].

Prob.	C	I_n	Evals.
TRN	1.7501	0.1490	100,000
TLN	0.4190	0.1535	100,000
BAK	0.9036	0.4978	100,000
NYT	38.8142	0.3906	600,000
BLA	0.1183	0.4267	600,000
HAN	6.1952	0.2041	600,000
GOY	0.1770	0.3262	600,000
FOS	0.0296	0.5239	1,000,000
PES	1.8134	0.2655	1,000,000
MOD	2.5394	0.3619	2,000,000
BIN	1.9986	0.3935	2,000,000
EXN	16.2722	0.3772	2,000,000

[2]Implementations of the problems together with their Pareto fronts can be downloaded from `https://emps.exeter.ac.uk/engineering/research/cws/resources/benchmarks/` under Design/Resilience.

6.1.4 The Low Level Heuristics

In order to optimise a WDN problem the SSHH hyper-heuristic requires a number of low level heuristics. The five heuristics used in [69] and a two-point crossover heuristic $\texttt{C}_5$ are used in these experiments. The low level heuristics are:

1. $\texttt{M}_0$ – change one pipe diameter randomly,
2. $\texttt{S}_1$ – swap two pipe diameters at random,
3. $\texttt{M}_2$ – increase or decrease a randomly selected pipe diameter by one size,
4. $\texttt{R}_3$ – "ruin" several pipes and rebuild randomly where the number of pipes to be changed is a parameter in the range $[1, 5]$,
5. $\texttt{S}_4$ – shuffle several pipes (i.e. makes several swaps) where the number of pipes to be changed is a parameter in the range $[1, 5]$, and
6. $\texttt{C}_5$ – two-point crossover of two vectors of network pipe diameters.

One of the objectives of this case study is to evaluate the performance of the additional crossover heuristic.

The low level heuristics can be grouped into four classes, mutational `M`, swap `S`, ruin and recreate `R`, and crossover `C`.

6.2 Experimental Methodology

The random, unbiased, single selection hyper-heuristic DBGen (see Chapter 3) employing the low level heuristics described in Section 6.1.4 is used to generate a database of heuristic selections and objective function values. Specifically, the DBGen hyper-heuristic is executed 40 times on each of the 12 WDN problem instances for 10,000, 20,000, 50,000, and 100,000 iterations for the small, medium, intermediate and large problems, respectively. The number of iterations for each size were chosen for computational feasibility. The number of 40 runs was chosen so as to ensure that robust statistics could be calculated for each problem instance. For each problem, each DBGen run r is seeded by a unique number

$$seed = 1000 + r$$

which is used to initialise the pseudorandom number generator used by DBGen, and to randomly initialise a WDN problem instance.

Although the computational cost of constructing such a database is significant, the objective of this study is not to compare optimisation performance using equivalent computational resources, but rather (as mentioned in Section 6.1.2) it is to investigate what can be learned from a pre-existing offline

learning database that has been constructed using a method that is independent of the optimisation algorithm under consideration.

Following Chapter 5, the Baum-Welch learning algorithm is used to offline train the SSHH hyper-heuristic. The Baum-Welch algorithm is used to estimate a set of parameters for SSHH's HMM that maximise the probability of generating a given sequence of training observations. Here, the parameters to be estimated are the HMM's state transition and heuristic emission matrices, while the training observations are subsequences of low level heuristics that are selected from the offline learning database. Specifically, the training observations consist of the 10 subsequences of length two and three with the largest γ-ratios in the database. Subsequences of length two and three are used as they occur more frequently in the database than longer subsequences, and as a result, the statistics calculated for these subsequences are more reliable. The number of 10 subsequences was chosen to ensure that the subsequence set contained sufficient heuristic "diversity", that is the subsequences consist of more that just one or two low level heuristics.

The results of executing the offline trained SSHH hyper-heuristic, denoted T-SSHH, on the WDN problems are compared with those of an untrained SSHH hyper-heuristic using a *leave-one-out* cross-validation methodology [9]. Recall that there are 12 problem instances in WDN domain. For each target problem, the training set consists of the 10 subsequences with the largest γ-ratios, chosen from the subsequences of the remaining 11 problems. The subsequences are used to train the HMM of the T-SSHH hyper-heuristic which is then evaluated on the target problem. This ensures that T-SSHH is always evaluated on a problem that is "unseen". The objective is to demonstrate empirically that an offline trained T-SSHH hyper-heuristic is able to learn and generalise from training subsequences selected across a number of problems. In this context, generalisation means that the trained T-SSHH hyper-heuristic is able to significantly outperform the untrained SSHH hyper-heuristic when evaluated on unseen problem instances.

Before Baum-Welch training, T-SSHH's hidden Markov model's state transition and heuristic emission probability matrices are randomised. The randomisation improves Baum-Welch learning which often fails when using equiprobable state transition and identity heuristic emission matrices. After Baum-Welch training, the probability matrices are edited to ensure that every state transition and heuristic emission has a non-zero probability of at least 0.0001. The offline trained T-SSHH hyper-heuristic is initialised with these edited matrices.

6.3 Online Learning

In this section the SSHH hyper-heuristic is used to optimise the 12 WDN problems. This experiment extends the work presented in [69] by evaluating the SSHH hyper-heuristic, with an additional crossover operator, on multiple WDN problems. The objective is to compare the performance of the SSHH algorithm with that of the evolutionary algorithms, and to determine the utility of the extra crossover

heuristic.

The SSHH hyper-heuristic is executed 40 times on each of the 12 problems in the WDN domain. For each problem, each run is seeded by a unique number

$$seed = 100 + r$$

that is distinct to the seeds used to generate the offline learning database. The number of iterations used by SSHH varies with the problem size. Specifically, SSHH is executed for 10,000, 20,000, 50,000, and 100,000 iterations for the small, medium, intermediate and large problems, respectively. The number of iterations for each size were chosen for computational feasibility. The number of 40 runs was chosen so as to ensure that robust statistics could be calculated for each problem instance. The SSHH hyper-heuristic is compared to the MOEAs on a single point on the published Pareto fronts; the solutions with the cheapest monetary cost. The cheapest solutions found by each algorithm for each problem are shown in table 6.3.

Table 6.3: The lowest cost C (M), resilience I_n, and the number of objective function evaluations (Evals.) for the solutions of the WDN problems produced by SSHH and MOEA. The result R indicates an equal (E), non-dominant (ND), dominant (D) or nonviable (NV) solution.

Prob.	SSHH			MOEAs			R
	C	I_n	Evals.	C	I_n	Evals.	
TRN	1.7501	0.1110	**8876**	1.7501	0.1490	100000	ND
TLN	0.4200	0.1579	**5850**	0.4190	0.1535	100000	ND
BAK	0.9036	0.4978	**9257**	0.9036	0.4978	100000	E
NYT	38.8142	0.3906	**18282**	38.8142	0.3906	600000	E
BLA	0.1186	0.4804	**17375**	0.1183	0.4267	600000	ND
HAN	6.1350	0.1797	**18580**	6.1952	0.2041	600000	ND
GOY	0.1781	0.4498	**17334**	0.1770	0.3262	600000	ND
FOS	0.0296	0.5249	**49124**	0.0296	0.5239	1000000	ND
PES	1.8319	0.2210	**48438**	**1.8134**	**0.2655**	1000000	D
MOD	2.5754	0.2739	**81101**	**2.5394**	**0.3619**	2000000	D
BIN	2.1366	0.3280	**56005**	**1.9986**	**0.3935**	2000000	D
EXN	7.6130	0.1912	**67841**	16.2722	0.3772	2000000	NV

The cheapest solutions found by SSHH and the MOEAs for the problems BAK and NYT are identical. For the small problem TRN and TLN, the medium sized problems BLA, HAN, GOY, and FOS no solution dominates. For the intermediate problem PES, and the large problems MOD and BIN, the MOEA solutions (shown in boldface) dominate those found by SSHH. As SSHH only evaluates the objective function after an acceptance check, rather than every iteration, the number of objective function evaluations (Evals.) for SSHH is always less than the number of iterations.

The solution produced by SSHH for EXN has a head deficit of $H = 2.6940$. The head penalties are calculated by summing the pressure violations over the whole network. When the violations are small, and spread evenly across a network, the solution can be viewed as semi-viable, or approaching viability. For the EXN network, the pressure violation occurs at a single pipe, and so the SSHH solution is deemed nonviable. However, by choosing the alternative constants $a = 2$, $b = 5000$, and $c = 0.1$ for the objective function, SSHH can find a viable solution for EXN where $C = 17.0886$ and $I_n = 0.3373$, using 49552 objective function evaluations.

As SSHH uses less iterations (and objective function evaluations) than the MOEA's the question arises as to whether its optimisation performance could be improved on the larger problems with a comparable number of iterations. Table 6.4 shows the cheapest cost solutions found by executing SSHH on PES, MOD, BIN, and EXN for additional iterations. Specifically, SSHH is run 40 times for 1,000,000 iterations on PES and 2,000,000 iterations on MOD, BIN and EXN. As extending the number of iterations for EXN using the original objective function constants also yields a nonviable solution, the result shown in table 6.4, denoted EXN*, uses the alternative constants defined above. While the MOD and BIN solutions remain dominant after the additional iterations, the solutions for PES and EXN* are now non-dominated. For PES, SSHH actually finds a lower cost solution than any of the MOEAs.

Table 6.4: The lowest cost C (M), head deficit H, and resilience I_n for the solutions of the PES, MOD, BIN, and EXN* problems produced by SSHH and MOEA. The result R indicates a non-dominant (ND) or dominant (D) solution.

Prob.	SSHH			MOEA			R
	C	I_n	Evals.	C	I_n	Evals.	
PES	1.8125	0.2546	**557549**	1.8134	0.2655	1000000	ND
MOD	2.5485	0.2792	**1457855**	**2.5394**	**0.3619**	2000000	D
BIN	2.0919	0.3440	**1014035**	**1.9986**	**0.3935**	2000000	D
EXN*	15.5632	0.3425	**681349**	16.2722	0.3772	2000000	ND

As the constants used to define the objective function are chosen to ensure that the function is positive for *all* the problems in the WDN domain, one would expect a multi-objective algorithm to outperform a single objective function optimiser. However, the differences in cost and network resilience are modest, while SSHH employs significantly less objective function evaluations than the MOEAs. This is important because, for many large problems (such as EXN), evaluating the objective function is computationally expensive.

These results demonstrate that the SSHH hyper-heuristic is a computationally efficient alternative to a multi-objective evolutionary algorithm for the optimisation of WDN problems. Furthermore, it should be noted that the SSHH algorithm can be extended to optimise multiple objectives [117].

6.4 Offline Learning

In order to improve optimisation performance the SSHH hyper-heuristic's HMM is trained offline with the Baum-Welch learning algorithm on an effective set of heuristic subsequences selected from the offline learning database. Following Chapter 5, the training subsequences are the 10 subsequences of heuristics of length two and three with the largest γ-ratios in the offline learning database. The training sets are constructed from these effective subsequences using a leave-one-out cross-validation methodology. Specifically, for each target WDN problem, the training set consists of the 10 subsequences with the largest γ-ratios, chosen from the subsequences of the 11 remaining problems. These subsequences are used to offline train the HMM of the SSHH hyper-heuristic which is then used to optimise the "unseen" target problem. This methodology gives rise to 12 training sets, one for each of the WDN problems.

The offline trained hyper-heuristic, denoted T-SSHH, is executed 40 times on each of the 12 problems in the WDN domain. The mean objective function values for each problem are shown in table 6.5.

Table 6.5: A problem by problem comparison of the mean final objective function value and standard deviation for SSHH and T-SSHH. Winning scores are shown in boldface.

Prob.	SSHH	SD	T-SSHH	SD
TRN	**357.4068**	16.9400	361.0980	18.6944
TLN	**87.2512**	2.6345	87.9132	2.4008
BAK	**179.0267**	0.9965	179.2223	1.4257
NYT	**8116.0158**	536.2183	8169.6525	645.3373
BLA	**22.2911**	1.2971	22.5268	2.4840
HAN	**1286.6305**	37.4569	1294.9970	42.7782
GOY	33.3999	0.1606	**33.3900**	0.0628
FOS	**4.2449**	0.6641	4.4674	0.9202
PES	**401.5132**	31.3177	409.5692	50.8569
MOD	**555.0569**	27.6637	564.3754	100.4345
BIN	**494.2631**	65.7123	537.6853	115.3677
EXN	4789.5943	402.1191	**4738.0677**	810.7313

The problem by problem results demonstrate that the SSHH hyper-heuristic outperforms the offline trained hyper-heuristic T-SSHH on 10 of the 12 WDN problems. The overall results are shown in table 6.6. Although the mean final log return $\overline{\alpha}_f$ is slightly better for T-SSHH than SSHH, the mean percentage change in objective function value is worse. The differences in mean final log returns are tested for statistical significance and the results are shown in table 6.7. The results of the one tailed Wilcoxon test together with the results contained in table 6.5 and table 6.6 indicate that T-SSHH is an inferior optimiser when compared with SSHH, and that offline learning has failed to improve optimisation performance.

Table 6.6: The mean final log return $\overline{\alpha}_f$, the mean percentage change, the mean number of iterations to a minimum, and the mean number of objective function evaluations.

	$\overline{\alpha}_f$	%	Min. Sel.	Obj. Eval.
SSHH	-2.9104	-88.7454	32916.5854	32238.0979
T-SSHH	-2.9134	-88.7435	32695.6125	30071.8729

Table 6.7: The sample pseudo-median difference $\hat{d}$, the sample median absolute deviation MAD, the sample mean difference $\bar{d}$, the standard deviation SD, the p-value, and the 99% confidence interval for $\overline{\alpha}_f(\text{T-SSHH}) - \overline{\alpha}_f(\text{SSHH})$. Statistically significant results are shown in boldface.

$\hat{d}$	MAD	$\bar{d}$	SD	p-value	Conf. Int.
0.0017	0.0225	-0.0030	0.2011	0.9029	$[-\infty, 0.0055]$

6.5 An Analysis of Heuristic Effectiveness

In Chapter 4, the γ-ratio is used to select effective subsequences of heuristics from a number of distinct problem domains. The γ-ratio can also be used, in certain circumstances, to select effective subsequences *across* a number of problem domains. For example, consider two comparable domains, and define the $\overline{\alpha}$-*order* of a domain to be the order of its low level heuristics (or heuristic classes) when ranked by their mean log return $\overline{\alpha}$. If the difference between the $\overline{\alpha}$-orderings of the two domains is small, then a subsequence that is effective in one domain is likely to be effective in the other (see Chapter 4, Section 4.6). In this section, the failure of offline training to improve optimisation performance in the previous section is investigated by considering the $\overline{\alpha}$-order of the low level heuristics of the WDN domain *during* the course of the optimisation process.

Consider the LOW and HIGH subsets of low level heuristics defined in Chapter 3, Section 3.3.2. The mean log return $\overline{\alpha}$ of the low level heuristics is calculated from the LOW and HIGH sets. The heuristics are then ranked by their $\overline{\alpha}$ values. The resulting LOW and HIGH $\overline{\alpha}$-orderings for the smallest problem in each size category, the EXN problem, and overall, are shown in table 6.8. The change in rankings is quantified by using the Spearman's Footrule distance [33].

The results in table 6.8 show some large differences in the orderings of the LOW and HIGH heuristic sets. For example, notice how the $\mathtt{M}_2$ heuristic changes from being one of the least effective heuristics in the HIGH sets, to one of the most effective heuristics in the LOW sets. These large differences in rank indicate that different individual heuristics are effective at different points in the optimisation process. Figure 6.2 shows the effectiveness of the low level-heuristics for each percentile over all 12 problems. The plot illustrates the changes in heuristic effectiveness as solution optimality increases. Notice that the two-point crossover heuristic $\mathtt{C}_5$ is the best performing low level heuristic in all but the last two percentiles P^i_{20} and P^i_{10}. Figure **??** shows the heuristic effectiveness for each heuristic class. The bars

Table 6.8: The low level heuristics in the HIGH and LOW subsets ordered by ascending mean log return $\bar{\alpha}$ from left to right, the Spearman's Footrule distance d, and the normalised Footrule distance $d' = \frac{d}{m}$.

Prob.	Set	$\overline{\alpha}$-order	d	d'
TRN	HIGH	C_5, R_3, S_4, M_0, S_1, M_2	10	0.5556
	LOW	C_5, M_2, M_0, R_3, S_1, S_4		
NYT	HIGH	C_5, S_4, R_3, S_1, M_0, M_2	12	0.6667
	LOW	C_5, M_2, M_0, R_3, S_1, S_4		
FOS	HIGH	S_4, C_5, R_3, S_1, M_0, M_2	16	0.8889
	LOW	M_2, M_0, C_5, S_1, R_3, S_4		
MOD	HIGH	C_5, R_3, M_0, M_2, S_1, S_4	6	0.3333
	LOW	M_2, C_5, M_0, R_3, S_1, S_4		
EXN	HIGH	C_5, R_3, M_0, M_2, S_1, S_4	6	0.3333
	LOW	C_5, M_2, M_0, S_1, R_3, S_4		
ALL	HIGH	C_5, S_4, R_3, S_1, M_0, M_2	14	0.7778
	LOW	M_2, C_5, M_0, S_1, R_3, S_4		

show the maximum and minimum $\bar{\alpha}$ values.

Figure 6.3 shows the normalised Spearman's Footrule metric between the HIGH order (percentile P^i_{100}) and the 10 local percentiles for the WDN and HyFlex domains. Notice that the WDN domain exhibits the largest change in heuristic order, and although the WDN problems are evaluated for more iterations than the HyFlex problems, the change in order occurs early on in the optimisation process starting at percentile P^i_{70}. This large change in heuristic order could explain why offline learning is effective in the HyFlex domains but not in WDN.

Such large changes in heuristic order and effectiveness are particularly relevant to SSHH as the efficient optimisation of such problems require online learning strategies, as any offline learned heuristic subsequences can only be effective at particular points during optimisation. The result of this analysis is also important for other optimisation techniques. For example, a vanilla genetic algorithm will typically execute a mutation operation and crossover operation at a given fixed rate for every iteration of the algorithm. For problems where the effectiveness of the crossover and mutate operation varies significantly during optimisation, optimisation performance can be improved by varying this rate accordingly.

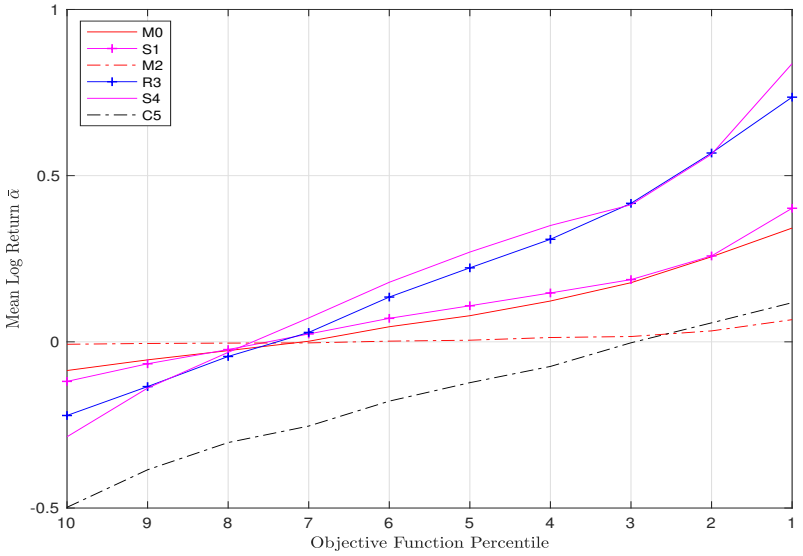


Figure 6.2: The mean log return $\overline{\alpha}$ of the low level heuristics in the WDN domain, averaged over the 10 local percentiles. Optimality increases from left to right, while negative $\overline{\alpha}$ values correspond to reductions in the objective function value.

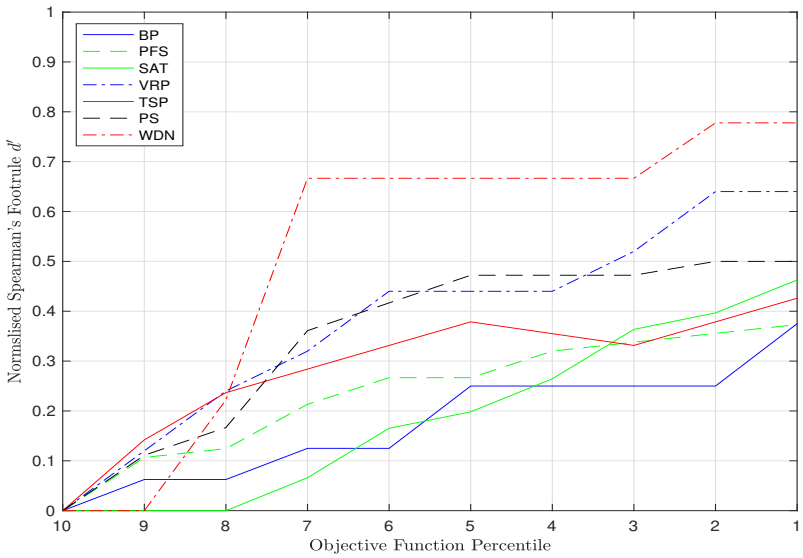


Figure 6.3: The normalised Spearman's Footrule metric between the HIGH order (percentile 10) and the 10 local percentiles for all domains. The optimisation process proceeds in time from left to right.

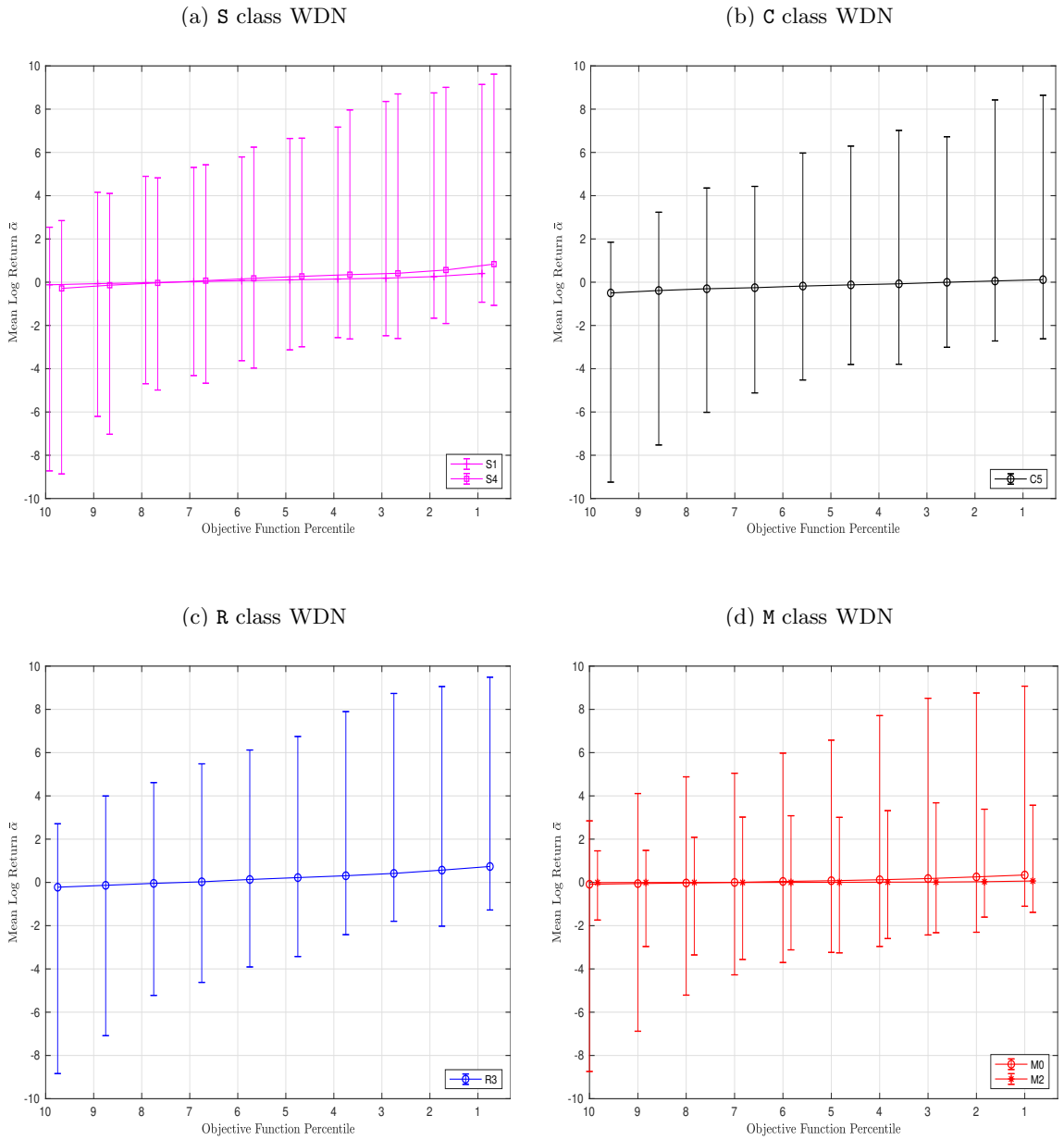


Figure 6.4: The mean log return $\overline{\alpha}$ of the low level heuristics in the WDN domain, averaged over the 10 local percentiles. The optimisation process proceeds in time from left to right, while negative $\overline{\alpha}$ values correspond to reductions in the objective function value. The bars show the maximum and minimum $\overline{\alpha}$ values.

6.6 Offline Learning with LOW Subsequences

In order to test the hypothesis that it is the large variance in heuristic performance during optimisation that prevents the Baum-Welch algorithm from finding a suitable set of HMM parameters, the experiment in Section 6.4 is repeated using subsequences that are effective when the objective function value is relatively low. Specifically, the 10 subsequences of length two and three with the largest γ-ratios are selected from the LOW set defined in Section 6.5. The Baum-Welch learning algorithm is used to train SSHH's HMM with the effective LOW subsequences, using a leave-one-out cross-validation methodology.

The SSHH hyper-heuristic is initialised with an identity heuristic emission matrix and equiprobable transition, parameter and acceptance matrices. The optimisation process with online learning is started, and when half of the specified iterations have been performed, the current HMM is switched for the offline trained HMM. The optimisation process, again with online learning, is then resumed. Although the method of switching to the LOW trained HMM is simple, it is trivial to implement, and requires no information regarding the objective function. The results for the SSHH hyper-heuristic when it is trained offline using effective subsequences chosen from the LOW set are denoted T-SSHH-L. The results for the offline trained hyper-heuristic with *no switching* mechanism, where the offline trained HMM is used from the outset of the optimisation process, are included for comparison purposes, and are denoted T-SSHH-L-NS.

The offline trained hyper-heuristic T-SSHH-L and T-SSHH-L-NS are each executed 40 times on each of the 12 problems in the WDN domain. Table 6.9 contains the mean final objective function value for each problem in the WDN domain. The T-SSHH-L hyper-heuristic outperforms SSHH on all 12 problems.

The results in table 6.10 show that, overall, the offline trained hyper-heuristic T-SSHH-L outperforms SSHH. The T-SSHH-L hyper-heuristic also outperforms T-SSHH-L-NS which demonstrates the importance of applying the effective subsequences at an appropriate point in the optimisation process.

The overall differences in the mean final log returns are tested for statistical significance as before, and the results are shown in table 6.11. They demonstrate that the differences, while small, are statistically significant, and so the offline trained hyper-heuristic T-SSHH-L outperforms the SSHH hyper-heuristic with 99% confidence.

These results show that, despite large variations in heuristic performance over the optimisation process, it is possible to significantly improve the optimisation performance of the SSHH hyper-heuristic on unseen WDN problems with offline learning. The result for EXN is particularly encouraging, as it shows that a training set constructed from a number of smaller problems can lead to improved optimisation on a larger, more complex, problem.

Table 6.12 contains a comparison of the cheapest monetary cost solutions found by SSHH, T-SSHH-L, and the MOEAs for each problem. Dominant solutions are shown in boldface. The performance of

Table 6.9: A problem by problem comparison of the mean final objective function value and standard deviation for SSHH and T-SSHH-L. Winning scores are shown in boldface.

Prob.	SSHH	SD	T-SSHH-L	SD
TRN	357.4068	16.9400	**355.4825**	14.4080
TLN	87.2512	2.6345	**86.7689**	2.4960
BAK	179.0267	0.9965	**178.8902**	0.9436
NYT	8116.0158	536.2183	**7985.0690**	448.5106
BLA	22.2911	1.2971	**22.0291**	1.1858
HAN	1286.6305	37.4569	**1279.6965**	36.1578
GOY	33.3999	0.1606	**33.3729**	0.0081
FOS	4.2449	0.6641	**4.1938**	0.6468
PES	401.5132	31.3177	**393.3359**	27.0505
MOD	555.0569	27.6637	**545.1349**	24.2674
BIN	494.2631	65.7123	**474.7110**	34.7078
EXN	4789.5943	402.1191	**4667.1560**	272.0995

Table 6.10: The mean final log return $\overline{\alpha}_f$, the mean percentage change, the mean number of iterations to a minimum, and the mean number of objective function evaluations.

	$\overline{\alpha}_f$	%	Min. Sel.	Obj. Eval.
SSHH	-2.9104	-88.7454	32916.5854	32238.0979
T-SSHH-L	**-2.9248**	**-88.8740**	34693.5292	**30437.8688**
T-SSHH-L-NS	-2.9105	-88.7656	**32384.0750**	32212.7813

T-SSHH-L is markedly inferior to the MOEAs on the large problems MOD, BIN and EXN. This could be due to the relatively low number of iterations employed by T-SSHH-L on these problems. Another cause could be that the 100000 iterations employed by DBGen is insufficient to discover the subsequences of heuristics necessary to construct effective training sets for the large problems.

It should be noted that although the values of C, H and I_n vary considerably across the WDN problems, the objective function constants a, b, and c are identical for each task. The EXN* result demonstrates that optimisation performance can be improved by "tuning" the constants for a particular problem. However, preliminary experiments suggest that using the same objective function for each problem facilitates learning and generalisation.

Table 6.11: The sample median difference $\hat{d}$, the sample median absolute deviation MAD, the sample mean difference $\bar{d}$, the standard deviation SD, the p-value, and the 99% confidence interval for $\overline{\alpha}_f(\text{T-SSHH-L}) - \overline{\alpha}_f(\text{SSHH})$. Statistically significant results are shown in boldface.

$\hat{d}$	MAD	$\bar{d}$	SD	p-value	Conf. Int.
-0.0056	0.0015	-0.0144	0.1941	**0.0000**	$[-\infty, -0.0036]$

Table 6.12: The lowest cost C (M) and resilience I_n for the solutions of the WDN problems produced by SSHH, T-SSHH-L, and the MOEAs. The solutions for EXN produced by SSHH and T-SSHH-L are nonviable with a head deficit H of 2.6940 and 7.4360 respectively.

Prob.	SSHH		T-SSHH-L		MOEAs	
	C	I_n	C	I_n	C	I_n
TRN	1.7501	0.1110	1.7501	0.1110	**1.7501**	**0.1490**
TLN	0.4200	0.1579	0.4200	0.1579	0.4190	0.1535
BAK	0.9036	0.4978	0.9036	0.4978	0.9036	0.4978
NYT	38.8142	0.3906	38.8142	0.3906	38.8142	0.3906
BLA	0.1186	0.4804	0.1186	0.4804	0.1183	0.4267
HAN	6.1350	0.1797	6.1177	0.1764	6.1952	0.2041
GOY	0.1781	0.4498	0.1783	0.4607	0.1770	0.3262
FOS	0.0296	0.5249	**0.0296**	**0.5249**	0.0296	0.5239
PES	1.8319	0.2210	1.8276	0.2171	**1.8134**	**0.2655**
MOD	2.5754	0.2739	2.5980	0.2775	**2.5394**	**0.3619**
BIN	2.1366	0.3280	2.1668	0.2967	**1.9986**	**0.3935**
EXN	7.6130	0.1912	2.7001	0.1891	**16.2722**	**0.3772**

6.7 Visualising Learning

The results of Section 6.6 demonstrate that the offline trained T-SSHH-L hyper-heuristic significantly outperforms the SSHH hyper-heuristic on the WDN problems. The SSHH hyper-heuristic employs a HMM to generate sequences of low level heuristic selections, their parameters, and acceptance check decisions. The HMM encodes a stationary probability distribution defined over these emissions. The objective of this section is to visualise the changes in the probability distribution encoded in the HMM that occur during online learning, and for the T-SSHH-L hyper-heuristic, the changes due to switching to the offline trained HMM.

During the optimisation process, every time SSHH or T-SSHH-L finds a new, best solution, the online learning mechanism updates the hyper-heuristic's HMM state transition matrix, and the parameter and decision check emission matrices, thus altering the encoded probability distribution. The identity heuristic emission matrix is not updated so that each hidden state always corresponds to a unique low

level heuristic.

The effects of online learning on the HMM can be visualised using the concept of probabilistic length and the Kullback-Leibler divergence (see Chapter 2). Specifically, the effects of online learning on SSHH's HMM can be illustrated by plotting the log number of iterations performed by SSHH against:

1. the probabilistic length of each state transition vector in the HMM,
2. the Kullback-Leibler distance between the current HMM and the initial equiprobable HMM,
3. the Kullback-Leibler distance between the HMM before and after online learning, and
4. the objective function value.

The probabilistic length of a state transition n-vector measures uncertainty; the shortest length $1/\sqrt{n}$ corresponds to maximum uncertainty, while the longest length of 1 corresponds to minimum uncertainty (or equivalently maximum certainty). As SSHH employs an identity heuristic emission matrix, increasing the certainty of the next hidden state to be selected is equivalent to increasing the certainty of the next heuristic selection.

The Kullback-Leibler divergence can be used to define a "distance" between probability distributions, and therefore HMMs [65]. Plotting the Kullback-Leibler distance between the current HMM and the initial, equiprobable HMM shows how much learning has taken place overall, while plotting the Kullback-Leibler distance between the current HMM before and after online learning shows how much has been learned during a particular learning episode.

These three plots are compared with a plot of the log number of iterations against the objective function value of the best solution found so far. This plot shows the overall progress of the optimisation process. As online learning episodes tend to occur less frequently as the optimisation process proceeds, a log scale for iterations is used to improve plot clarity.

The figures 6.6, and 6.7 show a typical run of SSHH and T-SSHH-L on the NYT problem. In this example, the SSHH hyper-heuristic is executed for 20000 iterations, and finds the minimum solution of

$$C = 44.8854,\ H = 0.0800,\ \text{and}\ I_n = 0.4147$$

after 3699 iterations, and 68 online learning episodes (see table 6.13). After this point no further online learning takes place and so the traces for probabilistic length, the KL-distances, and the objective function value remain flat (see figure 6.6).

The T-SSHH-L hyper-heuristic, which switches to the offline trained HMM after 10000 iterations (indicated in figures 6.6 and 6.7 by the vertical dashed line) finds the improved solution

$$C = 41.9999,\ H = 0.0022,\ \text{and}\ I_n = 0.4117$$

at iteration 13886, after 10 further online learning episodes. The abscissa of figure 6.7 has been rescaled so that the changes in the probability distribution after switching to the offline trained HMM

are clearer. It should be noted that in this case, offline learning not only improves optimisation performance, but also reduces the number of objective function evaluations (see table 6.13).

Table 6.13: The final log return α_f, the percentage change, the number of iterations to a minimum, and the number of objective function evaluations for run 2 of the NYT problem.

	$\overline{\alpha}_f$	%	Min. Sel.	Obj. Eval.
SSHH	-0.6051	-75.1720	**3699**	19467
T-SSHH-L	**-0.6377**	**-76.9676**	13886	**14660**

Plot 6.6a demonstrates that although online learning tends to increases the certainty of the next heuristic selection as one would expect, some hidden states such as $S1$ and $S5$ can "forget" and revert to less certain selection probabilities. The general increase in certainty is reflected in plot 6.6d that shows that, overall, online learning consistently increases the HMM distance from the initial equiprobable state. As the distance from the equiprobable state increases, so the propensity of the HMM to produce certain subsequences of heuristics over others increases. The changes between HMM states evident in figure 6.6b) indicate that although the largest changes between states occur early on in the optimisation process, significant changes also occur later on. This supports the conclusion of Section 6.5, that the effectiveness of heuristics and subsequences of heuristics changes during the optimisation process.

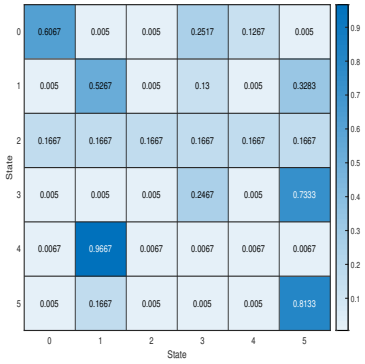
Figure 6.7b shows that the change in probability distribution between HMM states after switching to the offline trained HMM is an order of magnitude larger than any of the changes observed due to online learning. Figure 6.7d also demonstrates a significant increase in the KL-distance between the initial, equiprobable state, and the offline trained HMM. This increase in KL-distance indicates increased certainty regarding the selection of heuristics, and is reflected in the changes to the probabilistic lengths of the hidden states. Notice that with the exception of hidden state $S4$ the remaining states have all become more certain. These plots suggest that switching to the offline trained HMM moves SSHH to a more "certain" region of the heuristic search space. However, although the region is more certain, it allows another 10 online learning episodes to occur (see figure 6.7c).

The effect of learning can also be visualised with heat maps of the HMM's probability matrices. Figure 6.5 shows heat map plots of the transition and heuristic emission probability matrices for T-SSHH-L, before and after the switch to the offline trained HMM, and after optimisation has finished. The changes in probability due to switching the online trained HMM for the offline trained version are evident in figures 6.5a, and 6.5b and figures 6.5d and 6.5e, while the magnitude of this change can be seen in figure 6.7b.

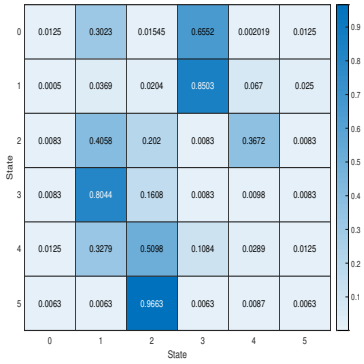
The effect of online learning on the offline trained HMM can be seen in figures 6.5b and 6.5c and figures 6.5e and 6.5f, Despite 10000 iterations, and 10 further online learning episodes, the offline trained heat maps do not change a great deal. However, although the changes are small, with many disappearing due to rounding, transition states $S1$ and $S3$ (row one and row three), and heuristic

emission state $H3$ (row three) are more certain of their next state. In this case, the 10 additional online learning episodes have reinforced the results of offline learning. This supports the hypothesis proposed in Chapter 5, that the Baum-Welch algorithm has found a "probabilistic maxima", from which the online learning algorithm cannot escape.

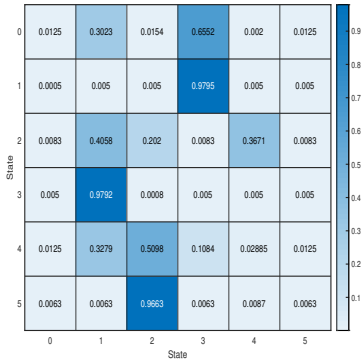
(a) Transitions before switch.



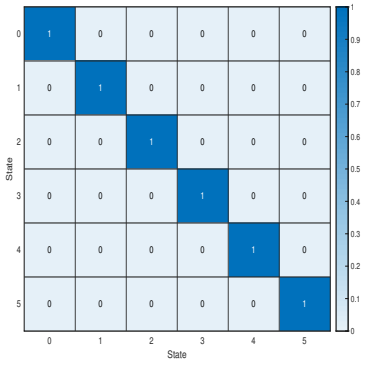
(b) Transitions after switch.



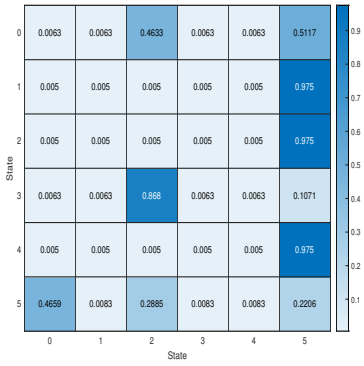
(c) Transitions at end.



(d) Emissions before switch.



(e) Emissions after switch.



(f) Emissions at end.

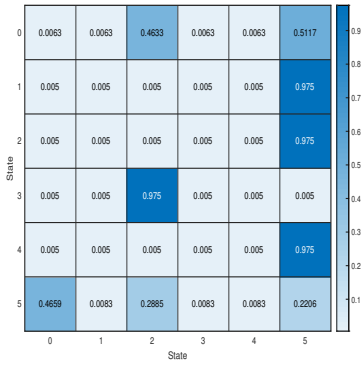
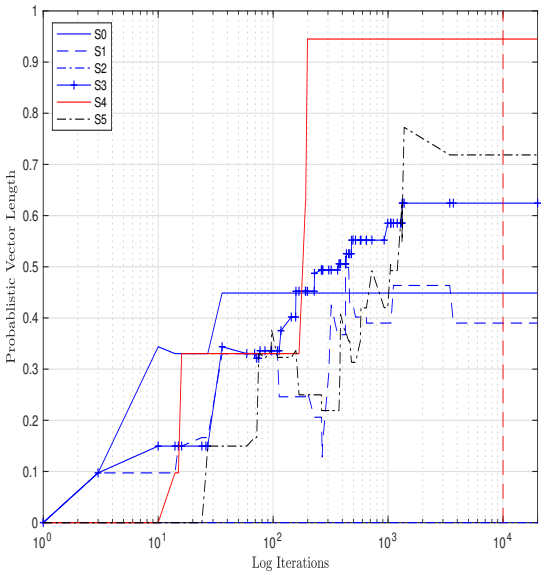
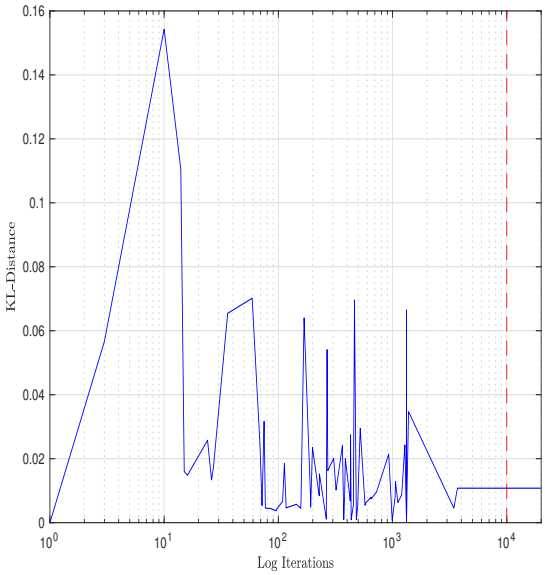


Figure 6.5: A heat map of the transition and emission probability matrices for T-SSHH-L for run 2 of the NYT problem after 10000, 10001, and 20000 iterations.

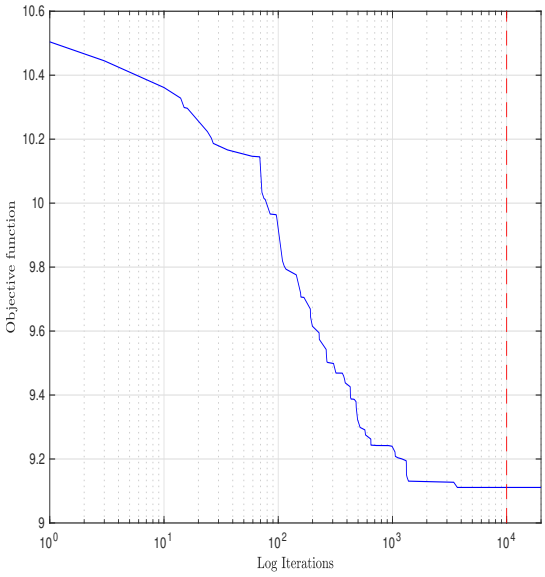
(a) Probabilistic length of each state vector.



(b) KL-distance between HMM states.



(c) Log objective function value.



(d) KL-distance from initial HMM state.

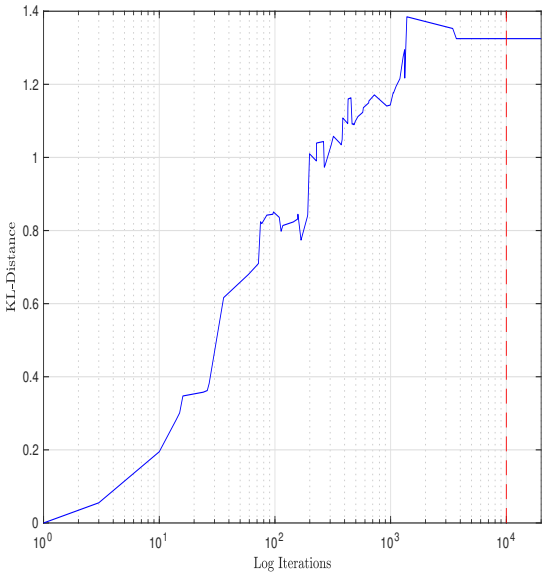
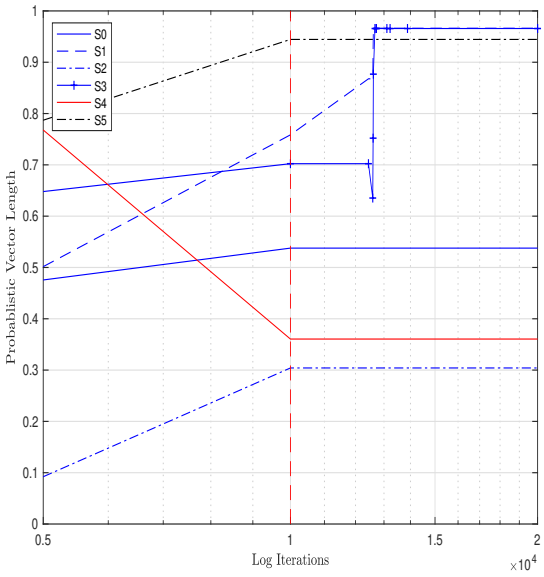
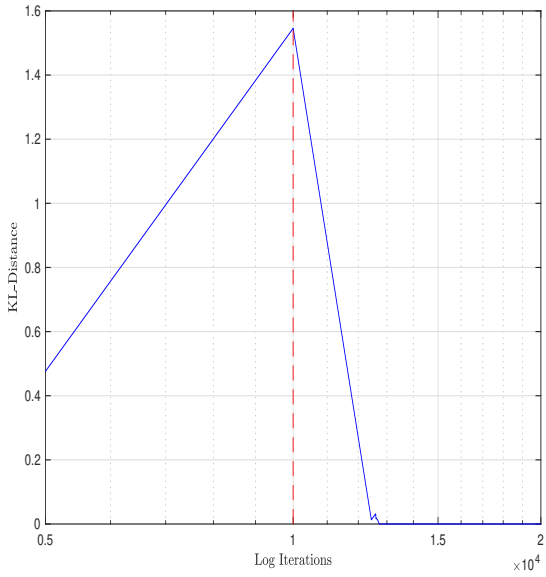


Figure 6.6: The online learning traces generated by SSHH when optimising NYT (run 2) resulting in a cost of 44.8854 and resilience 0.4147.

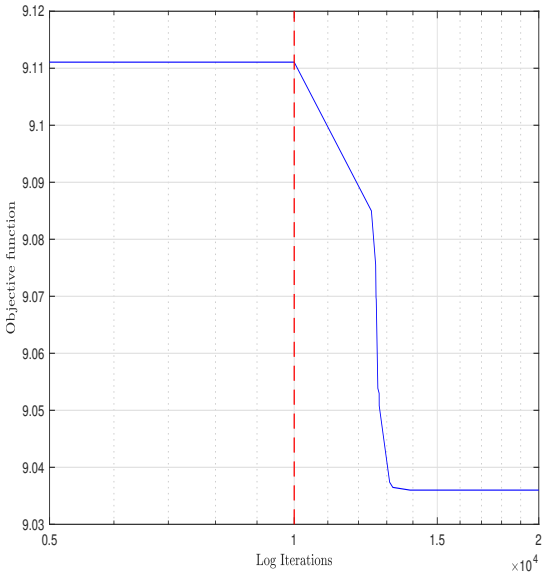
(a) Probabilistic length of each state vector.



(b) KL-distance between HMM states.



(c) Log objective function value.



(d) KL-distance from initial HMM state.

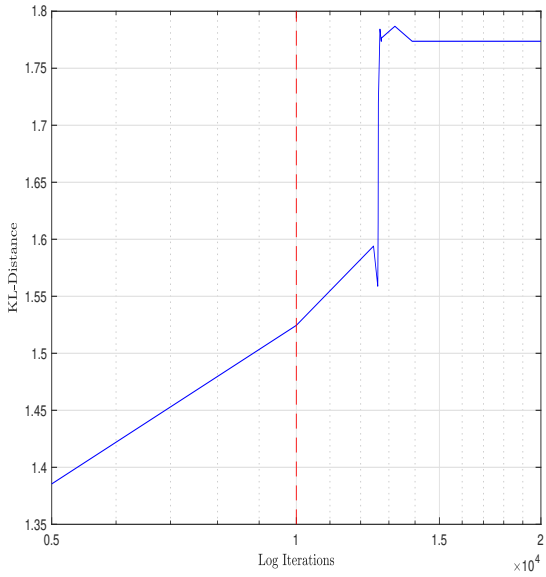


Figure 6.7: The online learning traces generated by T-SSHH-L when optimising NYT (run 2) resulting in a cost of 44.8854 and resilience 0.4147.

6.8 Scalable Learning

In this section, the potential for *scalable learning* is explored. Scalable learning is where a model developed for a small, computationally tractable task, is reused as the starting point of a model for a second, larger task. The concept of scalable learning is appropriated from [17] where the authors use evolutionary algorithms to generate novel heuristics for small problems which are then shown to perform well when tested on larger problems.

The idea is to generate training subsequences from a small problem which are then used, offline, to improve optimisation performance on a larger, more computationally expensive problem. With this in mind, a training set is constructed from the LOW selections and objective function values of the TRN problem. Specifically, the 10 subsequences of length two and three, with the largest γ-ratios are selected from the LOW set for TRN, and this training set is used to offline train SSHH. The offline trained hyper-heuristic, denoted T-SSHH-L-TRN, uses the same switching mechanism as T-SSHH-L described in Section 6.6. As all the training subsequences are drawn from one problem and evaluated on the remaining 11 problems, the leave-one-out cross validation methodology is not required. The results for the TRN problem are included for completeness.

The offline trained hyper-heuristic T-SSHH-L-TRN is executed 40 times on each of the 12 problems in the WDN domain. Table 6.14 contains the mean final objective function values for each problem in the WDN domain. The results demonstrate that T-SSHH-L-TRN outperforms SSHH on every problem (see table 6.9), and outperforms T-SSHH-L on five out of 11 problems.

The overall optimisation results shown in table 6.15 indicate that T-SSHH-L-TRN outperforms SSHH, and is slightly less effective than T-SSHH-L.

Table 6.14: A problem by problem comparison of the mean final objective function value and standard deviation for T-SSHH-L and T-SSHH-L-TRN. Winning scores are shown in boldface.

Prob.	T-SSHH-L	SD	T-SSHH-L-TRN	SD
TRN	**355.4825**	14.4080	355.6056	13.5715
TLN	**86.7689**	2.4960	87.0160	2.5482
BAK	178.8902	0.9436	**178.8715**	0.9204
NYT	**7985.0690**	448.5106	8011.2947	444.8608
BLA	22.0291	1.1858	**22.0194**	1.0362
HAN	1279.6965	36.1578	**1276.8813**	27.1250
GOY	33.3729	0.0081	**33.3721**	0.0079
FOS	**4.1938**	0.6468	4.2177	0.6845
PES	**393.3359**	27.0505	396.5199	29.9579
MOD	**545.1349**	24.2674	549.7243	25.2320
BIN	**474.7110**	34.7078	487.6116	39.1680
EXN	4667.1560	272.0995	**4632.3642**	333.9616

Table 6.15: The mean final log return $\overline{\alpha}_f$, the mean percentage change, the mean number of iterations to a minimum, and the mean number of objective function evaluations.

	$\overline{\alpha}_f$	%	Min. Sel.	Obj. Eval.
SSHH	-2.9104	-88.7454	**32916.5854**	32238.0979
T-SSHH-L	**-2.9248**	**-88.8740**	34693.5292	30437.8688
T-SSHH-L-TRN	-2.9233	-88.8643	33901.3896	**29479.8042**

The overall differences in final log returns are tested for statistical significance. The results, shown in table 6.16, indicate that the offline trained hyper-heuristic T-SSHH-L-TRN outperforms the SSHH hyper-heuristic on 11 of the WDN problems with 99% confidence.

Table 6.16: The sample median difference $\hat{d}$, the sample median absolute deviation MAD, the sample mean difference $\bar{d}$, the standard deviation SD, the p-value, and the 99% confidence interval for $\overline{\alpha}_f$(T-SSHH-L-TRN)$-\overline{\alpha}_f$(SSHH) (excluding the TRN problem). Statistically significant results are shown in boldface.

$\hat{d}$	MAD	$\bar{d}$	SD	p-value	Conf. Int.
-0.0039	0.0023	-0.0138	0.1977	**0.0000**	$[-\infty, -0.0022]$

The result that subsequences drawn from TRN, which is the smallest problem in the WDN domain, can be used to the improve the optimisation of EXN which is the largest, is notable. However, the improvement in performance is perhaps less surprising when one notes that the heuristic orderings in the LOW sets are very similar for TRN and EXN (see table 6.8).

These experimental results demonstrate that it is possible to learn offline from a small, computationally inexpensive problem, and then use this knowledge to improve optimisation performance on larger, more computationally expensive WDN problems with the same objective function.

6.9 Conclusions

The sequence based selection hyper-heuristic SSHH is able to produce viable solutions for the 12 WDN problems that are comparable in monetary cost and resilience to the cheapest solutions presented in [120] but using significantly less computational resources. In addition, the two-point crossover heuristic is shown to perform well when compared with the five low level heuristics employed in [69].

The selection hyper-heuristic DBGen is used to generate an offline learning database of low level heuristic selections and their objective function values across the 12 problems in the WDN domain. By employing the framework presented in Chapter 3, low-level heuristics and subsequences of heuristics can be identified in the database as being either effective or disruptive. Effective subsequences tend

to decrease the objective function value, while disruptive subsequences tend to increase the objective function value. The most effective heuristic subsequences are used to offline train the HMM of the SSHH hyper-heuristic using the Baum-Welch learning algorithm following the methodology presented in Chapter 5. However, in this case, the Baum-Welch algorithm is unable to learn an effective optimisation strategy because the performance of the effective subsequences varies considerably during the optimisation process, and this variation can be quantified by the Spearman's footrule metric. In order to test this hypothesis, subsequences that are effective when the objective function value is relatively low are selected from the database. These subsequences are used to train another HMM which is employed by the SSHH hyper-heuristic after the midpoint of the optimisation process. Although the method of switching between optimisation strategies is simple, it produces a small, but statistically significant improvement in performance, with 99% confidence. The final experiment, demonstrates that scalable learning is possible, that is, it is possible to learn offline from a small WDN problem which is computationally inexpensive, and transfer this learning to larger, more computationally expensive problems. Furthermore, this improvement in optimisation performance is also statistically significant, with 99% confidence.

Chapter 7

Conclusions

This thesis has presented a novel statistical framework for the offline identification and analysis of effective subsequences of low level heuristics. Effective subsequences of heuristics can be used to improve the optimisation of NP-hard problems. Such problems are characterised by the absence of any known polynomial-time algorithmic solution, and occur in a wide range of real-world applications. It has been shown that optimisation performance can be improved either directly by using effective subsequences to construct a sequence-based selection hyper-heuristic, or indirectly as training patterns for some offline learning algorithm. It has also been demonstrated that effective subsequences of heuristic classes can, in some cases, also be useful in constructing optimisers for problems from novel or unseen domains.

A Framework for Offline Learning

In Chapter 3 the unbiased, single selection hyper-heuristic DBGen is repeatedly executed on the HyFlex set of benchmark problems [84], in order to construct a number of offline learning databases from the resulting heuristic selections, and objective function values. The HyFlex benchmark set is a well known set of discrete optimisation problems that has been used in a number of studies. It is an implementation of six computationally hard problem domains: 1D bin packing (BP), permutation flow shop (PFS), Max Boolean Satisfiability (SAT), Vehicle Routing (VRP), Travelling Salesman (TSP), and Personnel Scheduling (PS). The use of a preexisting benchmark set significantly reduces the time required to develop and implement computational experiments, and facilitates the comparison of results between this and other studies.

The statistical framework is based on the concept of logarithmic returns [61]. Logarithmic returns are used widely in finance where they are employed to compare two or more variables when the originating price series consist of highly unequal values. In this thesis, logarithmic returns are used to normalise subsequences of objective function values, and their application in this context is novel.

The framework is used to analyse and categorise heuristic selections such as

1. individual heuristics,
2. subsets of heuristics, and
3. subsequences of heuristics,

based on their associated objective function values. Statistics are calculated over sets of occurrences of heuristic selections, and these statistics are used to quantify heuristic behaviours such as optimisation performance, and the changes in optimisation performance that occur during the optimisation process. In general, heuristic selections can be categorised as either effective selections that tend to decrease the objective function value, or disruptive selections that tend to increase the objective function value. By quantifying heuristic behaviour it becomes possible to employ visualisation techniques as an aid to the analysis of heuristic effectiveness.

In keeping with the literature, a non-parametric statistical test is used to validate experimental results [48] [49] [32] [116]. Specifically, the non-parametric Wilcoxon signed-rank test is used to establish stochastic orderings on a number of hyper-heuristics that are parameterised with sets of heuristic selections. The objective is demonstrate that certain hyper-heuristic parameterisations perform better than others, and that this improvement in performance is statistically significant, with 99% confidence.

It should be emphasised that the proposed framework for offline learning does not depend on the problem domain, the number or type of heuristics, or the process used to generate the heuristic selections.

An Analysis of Heuristic Subsequences

In Chapter 4, the framework is used to identify and quantify, well known heuristic behaviours, such as the expected exploration-exploitation behaviour that occurs in some heuristic subsequences, the effect of the choice and order of heuristics on optimisation performance, and the changes in heuristic performance that occur *during* the optimisation process. This analysis is novel, and demonstrates 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 a general purpose offline learning framework.

The γ-ratio, introduced in Chapter 3, Section 3.3.4 is defined as the ratio of the sum of the negative and positive changes of the objective function value over a set of occurrences. It is employed to select subsequences from an offline learning database, and its definition and use in this context are novel. Several sets of heuristic subsequences are selected using the γ-ratio, and these subsequences are evaluated with the unbiased, sequence based selection hyper-heuristic EvalHH using a leave-one-out cross validation methodology. The EvalHH hyper-heuristic is used to demonstrate empirically that well chosen subsequences of heuristics can produce better optimisation results than individual heuristic selections

when tested on unseen problems from the BP, PFS, SAT, and PS domains, with 99% confidence. This result is important because it underpins the thesis that subsequences of heuristics are important structural components of the optimisation process. The EvalHH hyper-heuristic, when parameterised with well chosen subsequences of heuristics can also outperform the SSHH hyper-heuristic. The SSHH hyper-heuristic is a published sequenced based hyper-heuristic that employs online learning of heuristic subsequences, and is known to perform well on the HyFlex problems [67]. These results demonstrate that subsequences can be chosen that *generalise* across the benchmark training problems leading to improvements in optimisation performance on unseen test problems. Furthermore, it demonstrates the utility of offline learning in general, and the proposed statistical framework in particular.

Having demonstrated that it is possible to extract useful information from the heuristic selections used to optimise the problems of a domain, the question arises as to whether it is possible to extract information from heuristic selections taken across a number of domains. Such cross-domain selections must necessarily consist of heuristic classes, and could be used to construct optimisers for novel, unseen domains. The γ-ratio can also be used to select effective cross-domain subsequences of heuristic classes. These sets of cross-domain subsequences, when evaluated using EvalHH, can generalise across three out of the four HyFlex domains tested. The performance of cross-domain subsequences can be significantly improved by remapping or reordering the classes according to their mean log return $\overline{\alpha}$. By mapping subsequences of heuristic selections to and from subsequences of abstract heuristic classes it is possible to construct cross-domain subsequences that are almost as effective on an unseen target domain, as subsequences selected using information that is specific to that domain[1]. These results indicates that cross-domain generalisation is possible for some, but not all, problem domains, and that heuristic order or context is crucial to optimisation performance.

The preceding results concern short subsequences of length two and three, which are evaluated using the EvalHH hyper-heuristic over a run of 150 iterations. The importance of subsequence length is examined by using the γ-ratio to select subsequences of heuristic classes of unrestricted lengths. The resulting long subsequence set outperforms a similarly constructed subsequence set of length two and three, albeit slightly, with 99% confidence. For many problems, especially those with computationally efficient heuristics, 150 iterations is a relatively small number over which to evaluate a hyper-heuristic's performance. The issue of evaluation run length is addressed by evaluating some subsequence sets for 10 minutes of wall clock time which is a more realistic execution time. The results show that the improvements in optimisation performance observed in the previous experiments are still present after significantly larger numbers of heuristic selections. These results demonstrates that the framework can also be applied to longer subsequences and longer evaluation runs.

[1]It should be noted that some domain knowledge is required to define the heuristic reordering

Hybrid Learning

In Chapter 5 a methodology for combining the online learning capabilities of the SSHH hyper-heuristic, with offline learning of effective subsequences using the Baum-Welch algorithm is presented. The use of the Baum-Welch algorithm to offline train the SSHH hyper-heuristic is novel.

The methodology, using subsequences chosen with the γ-ratio and leave-one-out cross-validation, is able to significantly improve the optimisation performance of the SSHH hyper-heuristic on five of the six HyFlex problem domains tested, with 99% confidence. The same methodology can also produce a hyper-heuristic that when trained with subsequences taken from a Pareto front, has similar optimisation performance to the untrained SSHH hyper-heuristic, but with more than a 60% improvement in run-time. The ability to offline train SSHH so that it is significantly more time efficient, at little or no cost to optimisation performance is important when optimising large, computationally expensive or time-critical problems. Moreover, in this case, the time savings are large enough to be commercially useful.

The previous experiments rely on large amounts of offline data. The γ-ratio can also be used to significantly improve performance in situations where there is only relatively small amounts of offline data available. This becomes important when dealing with novel or computationally expensive problems where large amounts of offline data may not be available.

An insight into what the Baum-Welch algorithm has learned can be gained by examining SSHH's state transition and heuristic emission probability matrices. An analysis of heat maps constructed from these matrices before and after optimisation shows that they have not been significantly altered by the online learning algorithm. This is because the Baum-Welch algorithm appears to have found a probabilistic maxima from which the online learning algorithm is unable to escape. These maxima can also be observed in the probability matrices learned in the other HyFlex domains.

These results show that it is possible to learn and generalise from sets of effective subsequences in order to improve the optimisation performance or time-efficiency of the SSHH hyper-heuristic. This demonstrates the utility of combining offline and online subsequence learning.

One application of such hybrid learning could be in real-world scenarios where multiple problem instances exist within a problem domain. In these cases SSHH could be trained offline on computationally simple instances of a problem to determine the initial biases towards certain subsequences of heuristics. Once trained, the SSHH hyper-heuristic could then be used on computationally expensive, real-world instances, benefiting from the domain-specific training, but using online learning to adapt to the specifics of such instances.

A Case Study: Water Distribution Networks

In Chapter 6 the offline learning framework is applied a novel problem domain; the optimisation of water distribution networks (WDN) [128].

The sequence based selection hyper-heuristic SSHH is used to produce viable solutions for 12 WDN problems that are comparable in monetary cost and resilience to the cheapest solutions produced by five multi-objective evolutionary algorithms (MOEA) [120] but using significantly less computational resources. In addition, a two-point crossover heuristic is shown to perform well when compared with the five low level heuristics employed in a previous study [69].

By employing the framework presented in Chapter 3, effective subsequences of heuristics are identified in an offline learning database. A set of effective heuristic subsequences, selected using the γ-ratio, is used to offline train the hidden Markov model (HMM) of the SSHH hyper-heuristic using the Baum-Welch learning algorithm. However, in this case, the Baum-Welch algorithm is unable to learn an effective optimisation strategy because the performance of the effective subsequences varies considerably during the optimisation process. In order to test this hypothesis, subsequences that are effective when the objective function value is relatively low are selected from the database. These subsequences are used to train another HMM which is employed by the SSHH hyper-heuristic after the midpoint of the optimisation process. Although the method of switching between optimisation strategies is simple, it produces a small, but statistically significant improvement in performance, with 99% confidence.

The last experiment in this case study, demonstrates that it is possible to learn offline from a small WDN problem which is computationally inexpensive, and transfer this learning to larger, more computationally expensive problems in order to improve optimisation performance with 99% confidence.

The Kullback-Leibler divergence presented in Chapter 2 is used to define a "distance" between probability distributions, and therefore HMMs [65]. Plotting the KL-distance between SSHH's current HMM and its initial, equiprobable HMM shows how much learning has taken place overall, while plotting the KL-distance between the current HMM before and after online learning shows how much has been learned during a particular learning episode. The concept of probability vector can also be used to visualise the effects of offline and online learning. This method of analysing the effects of hybrid learning is novel.

These results 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. Although the offline learning gains are modest, it should be possible to improve on these results.

Summary

This thesis has demonstrated the importance of heuristic context, and the generation of specific heuristic orderings to achieve significant improvements in hyper-heuristic performance. It has explored the capabilities of offline learning of heuristic subsequences and the possible synergies with online learning across a variety of problem domains for the first time. Furthermore, experimentation has shown that generalisation is possible in many cases, leading to the potential for algorithms trained on small prob-

lem instances which can then be applied to larger real-world instances, thus raising the prospect of a "library" of potential optimisers to be deployed on computationally expensive real-world problems. This work, and its application to a real-world problem taken from the water industry demonstrates that these methods can achieve comparable performance to state-of-the-art optimisation approaches. However, the main discoveries in this thesis demonstrate the limits of heuristic subsequences and what can be learned offline, hopefully providing some guiding principles to subsequent researchers in this field.

Limitations

This thesis, in common with all studies, is subject to a number of limitations.

The main constraint on this study has been the modest computer resources available. As all the experiments have been conducted on a modern desk top computer, the initial choice of 40 runs of 150 iterations for each problem in each HyFlex domain was dictated by statistical best practice and computational feasibility. The longer runs of 10-minutes of wall-clock time, while useful, could not be tested for statistical significance because of the small number of 10 runs per problem.

The replicability of computational experiments has long been a concern in the machine learning literature [88]. The PS domain of the HyFlex problems is not replicable, in the sense that although initialising a PS problem with the same random seed produces the same initial problem, identical optimisation runs, that is, application of the same low level heuristics and heuristic parameters, does not produce identical results. As the implementation details of all the HyFlex domains are hidden it was not possible to rectify this issue.

Some of the low level heuristics employed in this study are parameterised, and these heuristic parameters are not directly employed in the offline learning framework. The framework uses sets of heuristic occurrences to calculate performance estimates. When heuristic parameters are considered, these sets become very small, and the performance estimates become unreliable. Grouping all parametrised occurrences together results in estimates that are averaged over all choices of a parameter, and this requires a learning algorithm to generalise over the parameters. The role of heuristic parameters could be investigated by constructing a larger offline learning databases.

The analysis of SSHH parameterisations when online learning is enabled or disabled in Chapter 5, shows that the performance of the SSHH hyper-heuristic is significantly affected by the initial configuration of the probability matrices. Although offline learning generally improves optimisation performance, the offline learning algorithm produces parameterisations that appear to be local maxima from which the online learning algorithm is unable to escape. Further research into balancing the contribution of offline and online learning so that neither algorithm overwhelms the other is needed.

In Chapter 6, the SSHH hyper-heuristic and the MOEAs are only compared at a single point; the cheapest viable solution. This weakness could be addressed by employing the multi-objective version of SSHH described in [117]. This would allow SSHH to generate a Pareto front of solutions during

optimisation, enabling a direct comparison with the Pareto fronts presented in [120], and facilitating an analysis of the trade-offs between the conflicting design objectives of cost and resilience.

Future Work

This thesis could be extended in a number of ways. For example, the framework could be applied to the three extra HyFlex domains presented in [1]. The additional domains are implementations of the Knapsack, Quadratic Assignment, and Max-Cut problems. In order to define a minimisation problem the Knapsack problem employs a negative objective function, with a maximum value of zero, and some finite, but unspecified, negative minimum value. The use of a negative objective function (with zero) requires changes to the framework as it is not possible to calculate logarithmic returns in this case. Switching the sign of the objective function creates a maximisation problem which presents problems when calculating cross-domain statistics with other minimisation problems.

The Knapsack, Quadratic Assignment, and Max-Cut problems are discrete problem domains. The framework could also be tested on continuous problem domains.

All of the experiments in this thesis employ sets of 10 heuristic subsequences. The number of 10 subsequences was chosen to ensure that a subsequence set contained sufficient heuristic "diversity", that is the subsequences consist of more that just one or two low level heuristics, and to simplify the comparison of different subsequence sets. The Pareto fronts of the subsequence sets shown in figure 5.2, in Chapter 5 demonstrate the trade off between heuristic optimisation performance and execution time. This information could be used to refine the choice, and number of the subsequences in a set.

The γ-ratio which is used to select heuristic subsequences is defined as the ratio of the sum of the negative and positive changes of the objective function value over a set of occurrences. This definition could be improved. For example, the γ-ratio makes no use of the variance (or standard deviation) of the changes in the objective function value. Preliminary experiments suggest that employing variance could improve the effectiveness of heuristic selections.

In Chapter 6, one obvious task for future research would be to improve the mechanism employed to switch between the optimisation strategies encoded in the initial and (possibly various) offline trained HMMs. Although switching at the midpoint of optimisation is trivial to implement, a lot of information regarding the performance of heuristic subsequences over the whole optimisation problem is discarded.

It is also clear from the EXN* result in Section 6.3 that the optimisation of water distribution networks can be improved by "tuning" the objective function parameters for each problem. However preliminary experiments suggest that using the same objective function parameters facilitates generalisation, and the transfer of learning from small problems to larger ones. A fuller investigation into learning when objective function parameters vary for each problem is left to future research.

The framework has also been applied to the study of *sequences* of heuristic selections, that is, individual runs of heuristics generated, in this case, by the DBGen hyper-heuristic [123] [124]. This work could be extended. For example, the analysis of the differences and similarities of sequences within,

and across domains begun in [123] could be applied to other types of selection hyper-heuristic, and the notion of similarity employed could also be refined. Unpublished experimental work demonstrates that the remapping process described in Chapter 4 can also be successfully applied to the heuristic sequences used to train the Elman network in [124]. Further research into the identification of effective training sequences within and across domains could complement the study of heuristic subsequences.

Finally, this thesis is solely concerned with single objective optimisation problems. The framework could be extended to deal with multiple objective problems, and tested using the multi-objective version of SSHH presented in [117].

Appendix A

Clustering of Hyper-heuristic Selections

A.1 Introduction

Selection hyper-heuristics are methods that are typically used to solve computationally hard optimisation problems (see [15]). A selection hyper-heuristic selects heuristics from a given set of low level heuristics, deciding which heuristic to apply at a given point during the optimisation process. The sequences of low level heuristic selections and objective function values that result from the application of a simple selection hyper-heuristic to the HyFlex problem set (see [84]) are used to construct an offline learning database. The intention is to select effective subsequences of heuristics from this database and use them as inputs to machine learning algorithms in order to improve optimisation.

The purpose of this study is to algorithmically identify and analyse the similarities and dissimilarities that occur between the sequences of the database. By employing a suitable measure of similarity, the sequences of the offline database can be grouped or clustered according to the view of the similarity algorithm. It can be shown that by using a well-known algorithm from bioinformatics more commonly used to explore the conserved regions of DNA sequences, the Smith-Waterman algorithm (see [106]), it is possible to characterise problems using only the sequence of heuristic choices made by the hyper-heuristic. The Smith-Waterman algorithm is able to provide a measure of the level of similarity between two strings operating over any alphabet, and is used here to define a distance function between sequences of heuristics which is then used to perform a cluster analysis. The results presented here show that the Smith-Waterman algorithm is able separate the offline database into distinct problem domains.

The automatic separation and identification of problem domains from sequences of heuristics is

important because it demonstrates that there are subsequences of heuristics that are common to each problem domain, and that these subsequences vary between domains. This strengthens the thesis that subsequences of heuristics play an important role in the optimisation process. In addition, the identification of a (similar) problem domain can improve the choice of learning algorithm, learning algorithm parameterisation, and training data. For example, in [113] a k-nearest neighbour classifier is used to identify problems in an offline database that are similar to a target problem based on a set of measurable problem characteristics. This *metaknowledge* is then used to retrieve further problem specific information which is used to optimise the performance of a planning algorithm. The method described here differs from conventional metalearning approaches to algorithm selection in that, as only sequences of low level heuristic classes are employed, no problem specific information is required, preserving the domain barrier.

A.2 HyFlex and the Offline Learning Database

The Hyper-heuristics Flexible framework (or HyFlex, see [84]) is an implementation of 4 computationally hard benchmark problem domains:

1. 1D bin packing (BP),
2. permutation flow shop (PFS),
3. boolean satisfiability (SAT), and
4. personnel scheduling (PS).

Each 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 HyFlex domain has four general classes of low level heuristic:

1. mutation (`M`) which perturbs a solution randomly,
2. crossover (`C`) which constructs a new solution from two or more existing solutions,
3. ruin and recreate (`R`) which destroys a given solution partially and then rebuilds the deleted parts, and
4. local search (`L`) that incorporates an iterative improvement process and returns a non-worsening solution.

The actual number and implementation of the low level heuristics differs between problem domains.

A simple hyper-heuristic is executed for 150 selections, 40 times on each of the 10 HyFlex problems in each domain. The resulting 1600 sequences of heuristic selections and objective function values are

used to construct the offline database. The number of 40 trials was chosen because for a sufficiently large number (say $n > 30$) the central limit theorem ensures that the arithmetic mean of any results will be approximately normally distributed, regardless of the underlying distribution. This allows robust statistics to be calculated for each problem. The number of 150 selections was chosen after experimental observations indicated that no major improvements in objective function occurred beyond this point.

A.3 The Smith-Waterman Algorithm

The goal is to compare sequences of heuristic classes to obtain an understanding of the problem space from an algorithmic perspective. However, the comparison of sequences is not straightforward. For example, using the Hamming distance, two otherwise identical binary strings will appear dissimilar, that is score a high Hamming distance, if one string is shifted by one character in either direction. The Smith-Waterman algorithm (see [106]) is intended to overcome this because it attempts to identify similar regions of any given pair of strings. In bioinformatics, the Smith-Waterman algorithm is used to analyse the arrangement of DNA/RNA or protein sequences. The algorithm performs a *local sequence alignment* by use of dynamic programming; instead of looking at the whole sequence, the Smith-Waterman algorithm compares subsequences of all possible lengths and optimises a similarity measure. A large similarity score produced by the algorithm implies that the strings are very similar. A similarity score of 0 implies that the two strings have no symbols in common. The similarity measure is defined by a *similarity matrix* and a set of *gap penalties*. The similarity matrix defines the positive score for matching two symbols or the cost of mismatching two symbols. The gap penalties specify the score or cost of opening up a gap in a string and extending that gap in order to improve the fit with another string. Although the similarity matrix and gap penalties can be adjusted to alter the behaviour of the algorithm, in general it is not known which values are best suited for optimisation problems. In this study the similarity matrix is

	L	C	R	M
L	3	−2	−2	−2
C	−2	3	−2	−2
R	−2	−2	3	−2
M	−2	−2	−2	3

while the gap open and gap extend penalties are -3 and -1 respectively.

In this paper two distance functions, defined on sequences of heuristic selections, are considered: a distance function based on the Smith-Waterman algorithm and for comparison purposes, the Hamming distance.

The Smith-Waterman algorithm can be used to construct a simple notion of distance d between

sequences. In symbols

$$d(s_1, s_2) = \max_{SW} - sw(s_1, s_2)$$

where $\max_{SW}$ is the maximum value that can be attained by the Smith-Waterman function sw on the subsequences under consideration. A low d value indicate that two sequences are similar or close. In this study the maximum Smith-Waterman score over the 1600 sequences of the database is 357. This function should only be loosely interpreted as a distance function as it is not a metric in the formal sense.

A.4 Cluster Analysis

A k-medoid clustering algorithm employing the Smith-Waterman and Hamming distances is used to separate the entire offline learning database of 1600 sequences into 4 clusters corresponding to the 4 HyFlex domains. For clustering purposes, only the sequence selections up to and including the minimum objective function value are used, as these are the selections that are used as learning algorithm inputs. The accuracy of the resulting clusters are evaluated using the four commonly used measures: *purity*, *normalised mutual information (NMI)*, *Rand index*, and the F_5 (see [78]). For each measure, the worst clusterings have values close to 0 while a perfect clustering has a value of 1. The results shown in table A.1 demonstrate that the Smith-Waterman distance is superior in each measure.

Table A.1: A comparison of clustering accuracy.

Distance	Purity	NMI	Rand	F_5
S-W	0.8269	0.5954	0.7951	0.8001
Hamming	0.5350	0.2955	0.6185	0.7320

A.5 Conclusions

This experiment demonstrates that the sequences of heuristic selections produced by a simple hyper-heuristic on the HyFlex problems contain subsequences that are common to each problem domain and that differ significantly between problem domains. These similarities and differences can be identified automatically using the Smith-Waterman algorithm. Specifically, the clusters produced by a k-medoid cluster algorithm using Smith-Waterman are more accurate than those produced using the Hamming distance across 4 standard accuracy measures. The existence of discernible subsequences of heuristics in the database lends weight to the argument that the ordering of a subsequence is crucial to search efficacy

and this ordering varies with problem domain. The ability to identify (similar) problem domains from a sample of heuristic selections can also be used to guide the choice of learning algorithm and learning algorithm parameters for unseen problems or those with novel heuristic sets without requiring problem specific information. These results demonstrate the suitability of the Smith-Waterman algorithm as a measure of sequence similarity for offline learning applications.

Appendix B

Offline Learning with Elman Networks

B.1 Introduction

Hyper-heuristics are heuristic methods that are employed to solve computationally hard problems for which no known effective algorithmic solution exists. Typically such problems are presented as optimisation problems where the goal is to minimise an *objective function* defined on a space of solutions. Such methods have proved effective on a number of real world problems (see [15]).

A *selection hyper-heuristic* selects heuristics from a given set of low level heuristics and applies them sequentially to optimise a particular problem. Many hyper-heuristics employ learning algorithms in order to improve optimisation performance, and this learning may be classified as either *online* or *offline*. Online learning is based on the low level heuristic selections and resulting objective function values computed during the execution of a hyper-heuristic. 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. This paper is concerned with offline learning for selection hyper-heuristics.

A variety of machine learning algorithms have been proposed for offline learning (see for example [99], [20], and [111]). In [99] *classifier systems* are applied to the 1D bin packing problem. Here the system learns a set of rules which associate characteristics of the current problem state with specific heuristics. Heuristics are selected and applied sequentially, thus gradually altering the characteristics of the problem. The system when trained on several problems, generalises by also performing well on unseen problems. In [20] *case based reasoning* (CBR) is applied successfully to exam timetabling problems. The assumption underlying CBR is that "similar problems will have similar solutions". 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. Here the authors employ a two-stage learning process, one for the case representation (or feature selection) and another for source case selection. In [111], *messy genetic algorithms* are used to evolve combinations of condition-action rules which represent problem states and associated heuristics. Each chromosomes represents a hyper-heuristic and contains the set of rules that determine which heuristic should be applied to which problem state. When tested, these hyper-heuristics generalised well and solved many of the test problems efficiently.

In each case, learning is used to improve optimisation performance by improving the selection of *individual* heuristics at particular points in the search process across a number of training problems. In contrast, recent research (see [67] and [123]) has argued that heuristic selections should be understood as part of a *sequence* of selections. The concept of heuristic sequences is intuitive, certain heuristic orderings make sense (e.g. an explorative mutation followed by an exploitative local search) whereas others (e.g. the reverse of the previous example) do not.

The objective of this study is to test the thesis that subsequences of heuristics can be found in the offline learning database that are effective across a number of problems and (it is hoped) problem domains. A selection hyper-heuristic is executed on the well known HyFlex set of benchmark problems (see [84]) and the resulting sequences of low level heuristic selections and objective function values are used to construct an offline learning database. An Elman network (see [40]) is used to extract effective subsequences of heuristics automatically by learning from suitable sets of sequences chosen from the offline database. Elman networks are recurrent neural networks which naturally learn from, process and produce sequences of data. After training, the Elman network is used to compute new sequences of heuristics which are then evaluated on unseen HyFlex example problems. The aim is to determine if the network has generalised from the training sequences. In this context, generalisation means that the network is able to produce a sequence of heuristic selections which, when evaluated on the unseen examples, outperform the training sequences.

The benchmark problems are drawn from 4 distinct *problem domains*. Offline learning can be classified as either *intra-domain* or *inter-domain*. In intra-domain learning, the training sequences and the test optimisation problem are drawn from the same problem domain. In inter-domain learning, the training sequences and test problem can be drawn from different domains.

The results presented here demonstrate that an Elman network is capable of intra-domain learning and generalisation with 99% confidence when trained on suitable sequences of heuristic selections. When trained using an inter-domain training set, the Elman network did not exhibit generalisation indicating that inter-domain generalisation is harder, and the methodology used to choose the training sets is unsuitable in this case.

This paper is structured as follows. Section B.2 details the methodology and describes the construction of the offline learning database, the structure of the Elman networks and their training sets,

and the hyper-heuristic used to evaluate the sequences produced by the trained Elman networks. Section B.3 contains the results of two experiments designed to test the suitability of Elman networks for offline intra-domain and inter-domain learning. Finally, Section B.4 presents the conclusions of this study.

B.2 Methodology

Section B.2.1 contains a description of the HyFlex benchmark problems and the DBGen hyper-heuristic used to generate the offline learning database. In Section B.2.2 the mathematical concept of a logarithmic return is introduced and used to quantify hyper-heuristic performance, and to select training sequences from the database. Section B.2.3 details the architecture of the Elman network used in this study, while Section B.2.4 describes the construction of the intra-domain and inter-domain training sets. Finally, in Section B.2.5, the BLIND hyper-heuristic that is used to evaluate the sequences produced by the trained Elman networks is presented.

B.2.1 HyFlex and the Offline Learning Database

The Hyper-heuristics Flexible framework (or HyFlex[1], see [84]) is a set of benchmark problems that has been used in a number of studies. See for example [118], [37], [82], [38], [67], and [31]. HyFlex contains an implementation of four computationally hard problem domains:

1. 1D bin packing (BP),
2. permutation flow shop (PFS),
3. boolean satisfiability (SAT), and
4. personnel scheduling (PS).

Each 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 HyFlex problem has four general *heuristic classes*:

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

[1] HyFlex, Cross-domain Heuristic Search Challenge (CHeSC 2011) is used in this study (see http://www.asap.cs.nott.ac.uk/chesc2011/).

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

The actual number and implementation of the low level heuristics in each class differs between problem domains. As a result, it is not possible to directly compare sequences of low level heuristics from different domains. Instead, sequences of heuristic classes are compared.

Algorithm 11 The DBGen hyper-heuristic in pseudocode.

```
 1. ITERATIONS ← 150;
 2. new-sol ← initialiseSolution();
 3. new-obj ← f(new-sol);
 4. cross-sol ← initialiseSolution();
 5. cross-obj ← f(new-sol);
 6. while (ITERATIONS−− > 0) do
 7.     cur-sol ← new-sol;
 8.     cur-obj ← new-obj;
 9.     Heuristic h ← selectHeuristic();
10.     new-sol ← apply( h, new-sol, cross-sol );
11.     new-obj ← f(new-sol);
12.     double r ← ran();
13.     if (new-obj < cross-obj or r < 0.5) then
14.         cross-sol ← new-sol;
15.         cross-obj ← new-obj;
16.     end
17.     if (new-obj ≥ cur-obj and r ≥ 0.5) then
18.         new-sol ← cur-sol;
19.         new-obj ← cur-obj;
20.     end
21. end
```

The random, unbiased, single selection hyper-heuristic DBGen used to generate the offline learning database is shown in listing 11. The function *select()* (line 9) selects a single low level heuristic class at random from the set $\{\mathtt{C}, \mathtt{L}, \mathtt{R}, \mathtt{M}\}$. The function *apply()* (line 10) takes the heuristic class and chooses, again at random, an actual low level heuristic and its parameters from the available heuristics of that class. The actual heuristic is then applied to the current solution *cur-sol*, and if the class is C, to the current crossover solution *cross-sol*. An objective function evaluation (line 11) and an acceptance check (lines 12–20) are then performed. The function *ran()* (line 12) 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 *cur-obj* 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 approximately 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.

The DBGen hyper-heuristic is executed 40 times, for 150 selections, on the 10 problems in each of the 4 HyFlex domains. The resulting 1600 sequences of low level heuristic selections and associated

objective function values are used to construct an offline learning database. The number of 40 trials was chosen because for a sufficiently large number (say $n > 30$) the central limit theorem ensures that the arithmetic mean of any observed values will be approximately normally distributed, regardless of the underlying distribution. This allows robust statistics to be calculated for each problem. The number of 150 selections was chosen after experimental observations indicated that no major improvements in objective function occurred beyond this point.

B.2.2 Final Log Returns and the BEST Sequences

In this study, logarithmic returns are used to measure the performance of a hyper-heuristic. The *final log return* α_f of a hyper-heuristic run or sequence s is the log return between the initial solution of a run x_0, which has an objective function value o_0, and the best final solution $x_{\min}$ found during the run, which has an objective function value of $o_{\min}$. In symbols

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

Logarithmic returns allows us to easily compare the objective function values produced by a hyper-heuristic executing on a number of distinct problems or problem domains.

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. The anti-log of the mean of the logs is equivalent to the geometric mean. In symbols

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

assuming the values x_i all have the same sign. The geometric mean is always less than or equal to the arithmetic mean, and is employed to average values which have very different ranges. The geometric mean normalises the ranges, so that no range dominates the average. Although the use 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, the arithmetic mean of the final log returns $\overline{\alpha}_f$ is used in preference to the arithmetic mean of the decimal returns.

The *final unit log return* β_f is the final log return α_f divided by the sequence's length up to (and including) the minimum objective function value. That is

$$\beta_f(s) = \frac{\alpha_f(s)}{\min}.$$

The length of a sequence 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 HyFlex benchmark problems set consists of 4 problem domains, each one containing 10 problems. The set of the 40 "best" sequences in the offline database, denoted BEST, consists of the sequences with the lowest final unit log return β_f for each problem. These sequences are the shortest sequences that produce the largest decrease in the objective function value for each problem. As the offline database was generated by executing the DBGen hyper-heuristic 40 times on each of the 40 HyFlex problems, the "best" sequence for each problem is selected from a pool of 40 sequences.

B.2.3 Elman Networks

Elman networks are examples of *simple recursive neural networks*. They are typically applied to problems which express themselves naturally as temporal sequences such as natural language processing applications (see [40] and [41]). Such networks learn from, process, and produce sequences of data.

The training sequences are sequences of low level heuristics selections chosen from the offline learning database. Each such sequence is encoded using a *field* representation so that it can be processed by the Elman network. Specifically, each low level heuristic selection $\{\mathtt{M}, \mathtt{C}, \mathtt{R}, \mathtt{L}\}$ is encoded as a vector in $\{0,1\}^4$ where

$$\begin{aligned} \mathtt{M} &= (1,0,0,0) \\ \mathtt{C} &= (0,1,0,0) \\ \mathtt{R} &= (0,0,1,0) \\ \mathtt{L} &= (0,0,0,1), \end{aligned}$$

and $\mathtt{X} = (0,0,0,0)$ denotes a missing or unknown selection. These vectors are then concatenated to form an input pattern. For example, given the sequence `MCRLR`, an input pattern of 4 low level heuristic selections, corresponding to the current selection `L` and the three past selections `MCR` is

$$(\overbrace{1,0,0,0}^{\mathtt{M}}, \overbrace{0,1,0,0}^{\mathtt{C}}, \overbrace{0,0,1,0}^{\mathtt{R}}, \overbrace{0,0,0,1}^{\mathtt{L}})$$

while the output pattern corresponding to the next selection in the sequence is

$$(\overbrace{0,0,1,0}^{\mathtt{R}}).$$

The number of selections to be used as an input is termed the memory length of a selection strategy (see [6]). Using the current heuristic selection and those prior to it as inputs provides context for the next selection.

Initial experiments with memory length show that Elman network learning improves significantly as the number of past selections increases. Figure B.1 shows the results of training an Elman network with a memory length of 1, 2, 3, 4 and 5, on the INTRA training sequences for each domain (see Section

B.2.4). It should be noted that increasing the number of past selections also increases the number of weights which also improves learning.

In this study, a memory length of 4 is used because, with this number, the Elman network learns 80% (or more) of each training set. Thus, the 3-layer Elman network used in this experiment has 16 input units, 16 hidden units (and therefore 16 context units), 4 output units, and 596 weights. The hidden and output units employ the sigmoid activation function. The number of 16 hidden units was chosen arbitrarily.

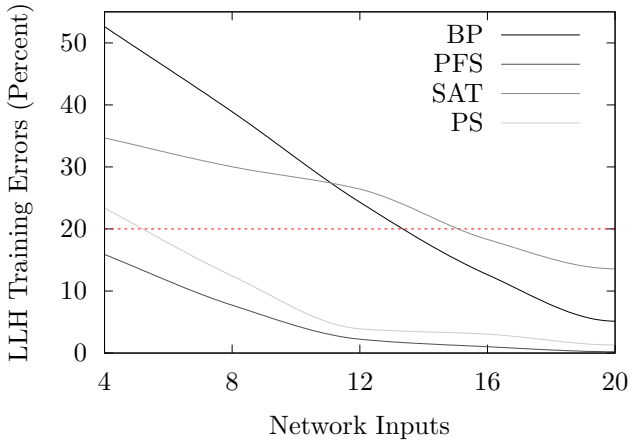


Figure B.1: The percentage of LLH training errors for an Elman network with 4, 8, 12, 16 and 20 inputs, 16 hidden units, and 4 output units, for each domain.

After training, given some initial input, an Elman network produces a sequence of outputs. The output sequence may converge to a single point, a limit cycle of repeating values, or produce a chaotic non-repeating sequence.

B.2.4 Training Sets

This study is concerned with offline intra-domain and inter-domain learning of heuristic classes. In intra-domain learning, the training sequences and the test optimisation problem are drawn from the same problem domain. This simplifies the learning task considerably as the low level heuristics in each class are identical for each problem and so the heuristic classes will have similar statistical characteristics across the problems of the domain. This is not generally the case for inter-domain learning where the training sequences and test problem can be drawn from different domains. These different domains will have different low level heuristic implementations and so the heuristic classes can have different statistical characteristics in each domain (see figure B.2). However, the general underlying

principles of each heuristic class should remain similar, for example a mutation operation should make small random changes, while a local search operation will greedily search the surrounding space.

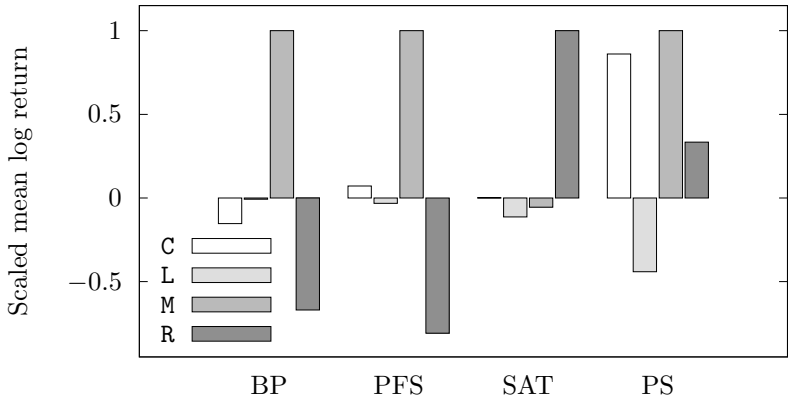


Figure B.2: The scaled mean log returns $\overline{\alpha}$ of the heuristic classes `C`, `L`, `M`, and `R` for each domain. In each domain the $\overline{\alpha}$ values have been scaled by the largest absolute $\overline{\alpha}$ value into the interval $[-1, 1]$.

The training sets for intra-domain and inter-domain learning are constructed from the BEST heuristic class sequences. As these sequences are the most efficient optimisations of each problem available they contain the most "useful information" regarding that problem and therefore they are prime candidates for inputs to a machine learning algorithm. In this study, *leave-one-out* cross-validation (see [9]) is employed to determine whether the Elman network sequences are able to outperform the BEST training sequences.

For intra-domain learning, the BEST subsequences are divided by domain into 4 sets of 10 sequences. For each problem in a domain, the sequence for that problem is left out of the training set and the remaining 9 sequences are used to train a network. The sequence produced by the trained network is then evaluated on the problem that was left-out. Thus the sequence generated by the network is always evaluated on a problem that the network has not been trained on. Applying this methodology gives rise to 40 training sets of 9 sequences, one for each problem, constructed from the 10 sequences selected for each domain.

For inter-domain learning, the BEST subsequences are again divided by domain into 4 sets of 10 sequences. For each domain, 3 sequences are selected from each of the 3 remaining domains. These sequences correspond to the problems with the lowest β_f in those domains. Applying this methodology gives rise to 4 training sets of 9 sequences, one for each domain, constructed from the 9 sequences selected from the other domains.

In each case, for each problem, the Elman network is trained with 9 sequences drawn from the

set BEST. It should be noted that for network training, only the accepted selections of each sequence up to (and including) the minimum objective function value are used. Rejected selections, and those selections that occur after the minimum objective function value are not used.

B.2.5 The BLIND Hyper-heuristic

The BLIND hyper-heuristic is used to evaluate sets of heuristic sequences on the HyFlex problems. It is intended to serve as a simple test bed and a "level playing field", in order to evaluate and compare the performance of sequences. The sequence based hyper-heuristic BLIND used in these experiments blindly applies a given sequence, one low level heuristic class after another to a HyFlex problem, accepting every selection. The actual low level heuristics and their parameters are chosen at random.

B.3 Results

Section B.3.1 presents the results of training the Elman networks with the intra-domain and inter-domain training sequences. In Section B.3.2 the sequences that are generated by the trained networks are evaluated on the HyFlex problems using the BLIND hyper-heuristic.

B.3.1 Network Training

An Elman network is trained with the intra-domain and inter-domain training sets using stochastic Backpropagation with early stopping over a maximum of 1000 epochs (see [9]) using the parameters shown in table B.1. The learning rate, momentum term, and the number of training epochs have not been optimised.

Table B.1: The Elman network structure and training parameters.

Input	Hidden	Out	Learn	Momentum	Epochs
16	16	4	0.1	0.25	1000

The results of network training are summarised in table B.2 and figure B.3. Table B.2 shows the results of training the Elman network with the 40 intra-domain training sets. The results are averaged over the 10 training sets in each domain. The columns show the average number of low level heuristics in each set, the average percentage of low level heuristics incorrect after training, the average network root mean square error, and the average number of epochs. Low level heuristic correctness is determined by applying a winner-take-all strategy to the network's output units and comparing the network's choice of heuristic with the target heuristic. Figure B.3a shows the percentage of low level heuristic errors during intra-domain training for 4 representative problems (number 7, 19, 34, and

Table B.2: The averaged training results of the Elman network on the intra-domain training sets.

Dom.	Num.	Wrong (%)	Error	Epochs
BP	369.0	12.6407	4.2958	907.7
PFS	94.5	1.0260	1.0491	328.9
SAT	288.2	18.3158	4.1991	918.9
PS	121.5	3.0474	0.9290	947.3

14) chosen from the BP, PFS, SAT and PS domains. These results demonstrate that the difficulty of learning intra-domain sequences of heuristic selections varies by domain. For example, the SAT domain sequences are much harder to learn than the training sequences of the other domains.

Table B.3: The averaged training results of the Elman network on the inter-domain training sets.

Dom.	Num.	Wrong (%)	Error	Epochs
BP	151	1.7391	1.0443	999
PFS	224	1.0638	1.3208	994
SAT	175	0.7194	0.8031	616
PS	221	1.0810	0.9290	739

Similarly, table B.3 and figure B.3b show the results of training the Elman network with the 4 inter-domain training sets. These results demonstrate that intra-domain learning is harder than inter-domain learning.

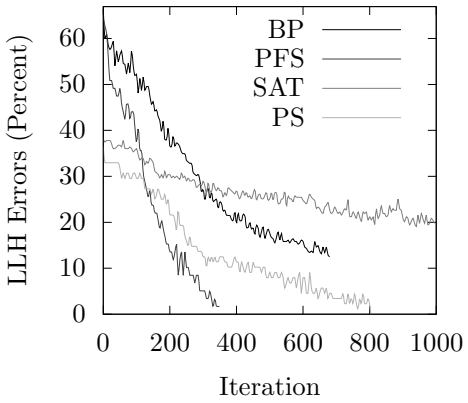
After training, the Elman network is then given the initial "blank" input `XXXX`. As Elman networks are deterministic, the intra-domain trained networks produces a set of 40 sequences, one for each problem, while the inter-domain trained networks produce a set of 4 sequences, one for each domain.

B.3.2 Evaluating the Elman Network Sequences

The BLIND hyper-heuristic is parameterised with three sets of sequences denoted BEST, INTRA, and INTER and then executed 40 times on each of the HyFlex problems. The INTRA sequence set is generated by the intra-domain trained Elman networks, while the INTER sequence set is generated by the inter-domain trained Elman networks. It should be noted that the pseudorandom number seeds and therefore the initial solutions used for the INTRA, INTER, and BEST evaluation runs presented here are identical and distinct to the pseudorandom number seeds used by DBGen to generate the offline database from which the BEST sequences are selected.

When parameterised with the BEST sequences the BLIND hyper-heuristic applies *all* the accepted

(a) Intra-domain training results.



(b) Inter-domain training results.

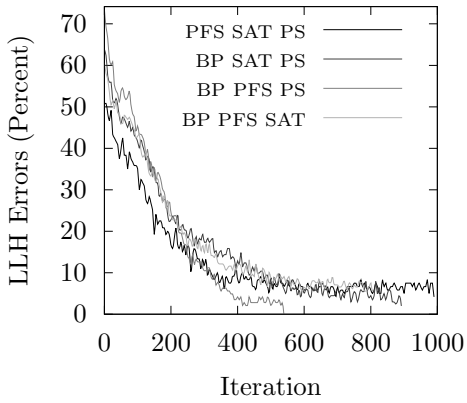


Figure B.3: The Elman network training results for the intra-domain and inter-domain sets. In figure (a) the training sequences are drawn from the BP, PFS, SAT and PS domains. In figure (b) the training sequences are drawn from the {PFS SAT PS}, {BP SAT PS}, {BP PFS PS}, and {BP PFS SAT} domains.

selections including those *after* the minimum objective function value. This is done because some sequences in BEST find a minimum quickly, in some cases after only 9 selections. Using all accepted selections gives the BLIND hyper-heuristic a larger number of iterations/selections to better optimise a problem. The length of the BEST sequences also dictate the number of selections used by the INTRA and INTER parameterisations. The results of evaluating the INTRA and INTER sequence sets on the HyFlex problems are compared to the BEST sequences (see table B.4). The intention of the comparison is to determine whether the network has learned anything over and above the information contained in the BEST sequences. The INTRA sequences outperform the BEST sequences overall and on each domain, while BEST outperforms INTER overall, and on each domain except the PFS domain. The best generalisation is observed between INTRA and BEST on the SAT domain (which was the hardest to learn). The overall averages are calculated over 1600 sequences, and the domain averages are calculated over 400 sequences.

A paired t-test is used to establish whether the difference observed in the mean final log returns of BEST and INTRA is statistically significant. Formally, the null hypothesis

$$\overline{\alpha}_f(\text{BEST}) \geq \overline{\alpha}_f(\text{INTRA})$$

is rejected if t lies outside the interval $[-2.3287, \infty)$ and the alternative hypothesis

$$\overline{\alpha}_f(\text{BEST}) < \overline{\alpha}_f(\text{INTRA})$$

Table B.4: A domain by domain and overall comparison of the mean final log return $\overline{\alpha}_f$ of BEST, INTRA and INTER.

Dom.	BEST	INTRA	INTER
BP	-0.2172	**-0.2202**	-0.0375
PFS	-0.0043	-0.0049	**-0.0051**
SAT	-0.4345	**-0.6919**	-0.2313
PS	-1.7912	**-1.8042**	-1.5560
All	-0.6118	**-0.6803**	-0.4575

is accepted with 99% confidence. The results of the t-test are shown in table B.5. The difference in mean is statistically significant overall, and for the PFS and SAT domains with 99% confidence. For the BP and PS domains the difference in mean is not statistically significant.

Table B.5: The domain, the sample mean difference, the standard deviation, the t-score, and the interval within which the population mean difference falls with 99% confidence.

Dom.	Diff.	SD	t-score	Conf. Int.
BP	-0.0030	0.0821	-0.7214	[-0.0136, 0.0077]
PFS	-0.0006	0.0024	-5.2796	[-0.0009, -0.0003]
SAT	-0.2573	0.1085	-47.4485	[-0.2714, -0.2433]
PS	-0.0130	0.1225	-2.1289	[-0.0289, 0.0028]
All	-0.0685	0.1424	-19.2384	[-0.0777, -0.0593]

B.4 Conclusions

The sequence set BEST consists of the sequences with the lowest final unit log return β_f for each HyFlex problem. An intra-domain training set INTRA and an inter-domain training set INTER are constructed from the BEST sequences and used to train an Elman network. In order to estimate the Elman network's capacity for generalisation the network is evaluated using a *leave-one-out cross-validation* methodology. The first result presented in this study demonstrates that the Elman network is capable of intra-domain generalisation with 99% confidence. This result is notable because the Elman network is able to significantly outperform the sequences on which it was trained. The process of generalisation across the training problems within a domain has generated a network that is able to perform better on unseen test problems in that domain. This shows that useful information can be learned about the problems in a domain from the sequences of heuristic selections used to optimise them. The second

result shows that the Elman network is not capable of inter-domain generalisation using the training set INTER in spite of the fact that the training sets are easier to learn. This suggests that inter-domain generalisation is harder than intra-domain generalisation, and that low training errors need not translate into good generalisations. This was generally to be expected, the sequences of heuristics learned on one domain are not expected to be applicable to another. However, there are exceptions, for example the performance on PFS domain from the INTER trained network performed well and indicates perhaps that a more general strategy for solving the PFS domain would be successful.

Overall, the Elman network proved to be able to generalise the training sequences for intra-domain learning which opens up the possibility of the use of bespoke learned algorithms for particular problems. Inter-domain generalisation was more difficult, as expected, and more work would need to be conducted to determine whether a different methodology would allow domains with similar sequences to be identified.

Appendix C

Database Structure

This chapter presents an overview of the structure of the offline learning database used in this thesis. The database is implemented using the MySQL[1] relational database management system, and consists of four main table schemas; a sequence table, two subsequence tables, and a results table. These tables are described in Section C.1, Section C.2, and Section C.3 respectively. For clarity, some minor tables and some table columns, such as those used to improve query efficiency, or to facilitate the calculation of certain values, have been omitted. The tables have not been normalised.

C.1 The Sequence Table

The offline learning database is generated by repeatedly executing the DBGen hyper-heuristic (see Chapter 3, Section 3.2) on a given set of benchmark problems. The sequence table is used to hold hyper-heuristic run data, that is, the data generated when a hyper-heuristic is executed on a particular problem instance. A hyper-heuristic run is stored as consecutively indexed rows in the sequence table, where each row corresponds to a single iteration of the hyper-heuristic. Table C.1 contains a description of the sequence table columns, while table C.2 contains some example run data. When extracting sequences from the sequence table it is important to distinguish between a whole sequence of accepted *and* rejected selections, and the sequence of accepted selections used to generate subsequences. All timings are given in milliseconds (ms).

[1] see `https://www.mysql.com`

Table C.1: The sequence table column names and descriptions.

Column	Description
Dom.	The problem domain.
Prob.	The problem instance identifier (1-10).
HH	The hyper-heuristic.
Run	The run identifier.
Seq.	The sequence index.
LLH	The low level heuristic selected.
P	The heuristic parameter selected.
Cur.	The current objective function value.
New	The new objective function value.
Log Ret.	The logarithmic return α.
A	The result of the acceptance test (True/False).
Evals.	The current number of objective function evaluations.
LLH T	The time taken to apply the selected low level heuristic (ms).
Obj. T	The time taken to evaluate the objective function (ms).

Table C.2: Some example values for the sequence table.

Dom.	Prob.	HH	Run	Seq.	LLH	P	Cur.	New	Log Ret.	A	Evals.	LLH T	Obj. T
BP	3	SSHH	0	0	$\mathtt{M}_3$	0.9	0.0480	0.0480	0.0000	T	1	14	1
BP	3	SSHH	0	1	$\mathtt{R}_2$	1.0	0.0480	0.0428	-0.0497	T	2	9	2
BP	3	SSHH	0	2	$\mathtt{R}_2$	0.3	0.0428	0.0422	-0.0061	T	3	2	1

C.2 The Subsequence Tables

Sequences of accepted selections are extracted from the sequence table, and broken down into consecutive subsequences of length $n = 2, 3, \ldots, 15$. The raw subsequence table holds data for each individual occurrences of a particular subsequence. This table schema is used to store low level heuristic data for domain specific learning, and, in a separate table, heuristic class data for cross domain learning. Table C.3 contains a description of the raw subsequence table columns, while table C.4 contains some example raw subsequence data for heuristic classes. The maximum subsequence length of 15 was chosen because beyond this length the numbers of subsequence occurrences are too small to calculate robust statistical estimates of performance. All timings are given in milliseconds (ms).

The data in the raw subsequence tables is then consolidated and the relevant averaged values for each subsequence is stored in the subsequence tables for low level heuristics and heuristic classes

Table C.3: Raw Subsequence Data Columns.

Column	Description
Dom.	The problem domain.
Prob.	The problem instance identifier.
HH	The hyper-heuristic.
Run	The run identifier.
Pos.	The position in a sequence where the subsequence occurs.
Subseq.	The subsequence.
Cur.	Current objective function value *before* applying the subsequence.
New	The new objective function value *after* applying the subsequence.
Unit Log Ret.	Unit log return β.
T	Execution time of subsequence (ms).

Table C.4: The Raw Subsequence Data Table.

Dom.	Prob.	HH	Run	Pos.	Subseq.	Cur.	New.	Unit Log Ret.	T
BP	3	DBGen	0	13	CC	0.0434	0.0434	0.0000	539
BP	3	DBGen	0	52	CC	0.0499	0.0381	-0.0586	551
BP	3	DBGen	0	52	CCL	0.0499	0.0421	-0.0246	576

respectively. Table C.6 contains a description of the subsequence table columns, while table C.5 contains some example subsequence data for heuristic classes. Note that when storing low level heuristic subsequence data the Doms. column has a fixed value of 1.

Table C.5: Subsequence Table Columns.

Column	Description
HH	The hyper-heuristic.
Subseq.	The subsequence.
Doms.	The number of problem domains where the subsequence occurs.
Mean	The mean unit log return $\overline{\beta}$.
P. Sum	The sum of the positive β values.
N. Sum	The sum of the negative β values.
N. Num.	The number of negative β values.
T. Num.	The total number of occurrences.
Min.	The minimum unit log return $\min \beta$.
Max.	The maximum unit log return $\max \beta$.
Var.	The variance of the mean unit log return $\overline{\beta}$.
T	Average execution time of subsequence (ms).

Table C.6: Subsquence Table.

HH	Subseq.	Doms.	Mean	P. Sum	N. Sum	N. Num.	T. Num.	Min.	Max.	Var.	T
DBGen	CC	4	0.1642	779.8570	50.1554	3346	4443	-1.9474	1.9339	0.2307	15.5772
DBGen	CCC	4	0.1323	120.7200	7.0897	588	859	-1.2648	1.2867	0.1242	26.5041
DBGen	CCCC	4	0.0907	16.0861	0.1168	111	176	-0.0354	0.9645	0.0562	38.5262

C.3 The Results Table

The results for each run of a hyper-heuristic are stored in the results table. Hyper-heuristic results, for each problem, each domain, or across a number of domains can be calculated by grouping and averaging the rows of the result table. Table C.7 contains a description of the result table columns, while table C.8 contains some example result data for the SSHH hyper-heuristic. All timings are given in milliseconds (ms).

Table C.7: Results Table Columns.

Column	Description
Dom.	The problem domain.
Prob.	The problem instance identifier.
HH	The hyper-heuristic.
Run	The run identifier.
Initial	Initial objective function value.
Final	Best, final objective function value.
Log Ret.	Final log. return α_f.
Percent	Percentage reduction in objective function value.
Seq.	The position of the best solution in the whole sequence.
Pos.	The position of the best solution in the sequence of accepted selections.
Evals.	The number of objective function evaluations.
T	Average execution time of the subsequence (ms).

Table C.8: Results Table.

Dom.	Prob.	HH	Run	Initial	Final	Log Ret.	Percent	Seq.	Pos.	Evals.	T
BP	3	SSHH	0	0.0480	0.0151	-0.5018	-68.5045	146	73	75	4053.5238
BP	3	SSHH	1	0.0497	0.0151	-0.5173	-69.6148	140	71	67	3853.3131
BP	3	SSHH	2	0.0496	0.0148	-0.5267	-70.2649	146	78	88	3892.3478

Bibliography

[1] Adriaensen, S., Ochoa, G., and Nowé, A. A benchmark set extension and comparative study for the HyFlex framework. In *IEEE Congress on Evolutionary Computation* (2015), CEC, pp. 784–791.

[2] Ahmadi, S., Barone, R., Cheng, P., Cowling, P., and McCollum, B. Perturbation based variable neighbourhood search in heuristic space for examination timetabling problem. pp. 13–16.

[3] Alperovits, E., and Shamir, U. Design of optimal water distribution systems. *Water Resources Research 13*, 6 (1977), 885–900.

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

[5] Asta, S., and Özcan, E. A tensor-based selection hyper-heuristic for cross-domain heuristic search. *Information Sciences 299* (2015), 412 – 432.

[6] Bai, R., Burke, E. K., Gendreau, M., Kendall, G., and McCollum, B. Memory length in hyper-heuristics: An empirical study. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling* (2007), pp. 173–178.

[7] Baños, R., Reca, J., Martínez, J., Gil, C., and Márquez, A. Resilience indexes for water distribution network design: A performance analysis under demand uncertainty. *Water Resources Management 25*, 10 (2011), 2351–2366.

[8] Bianchi, L., Dorigo, M., Gambardella, L. M., and Gutjahr, W. J. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing 8*, 2 (2009), 239–287.

[9] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer, 2006.

[10] Blum, C., and Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys 35* (2003), 268–308.

[11] Bonabeau, E., Dorigo, M., and Theraulaz, G. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, 1999.

[12] Boussaïd, I., Lepagnot, J., and Siarry, P. A survey on optimization metaheuristics. *Information Sciences 237* (2013), 82–117.

[13] Bragalli, C., D'Ambrosio, C., Lee, J., Lodi, A., and Toth, P. An minlp solution method for a water network problem. Tech. Rep. RC24495, IBM Research, 2008.

[14] Burke, E. K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Qu, R. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society 64*, 12 (2013), 1695–1724.

[15] Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. *A Classification of Hyper-heuristic Approaches*. Springer US, 2010.

[16] Burke, E. K., Hyde, M. R., Kendall, G., Ochoa, G., Özcan, E., and Woodward, J. R. *A Classification of Hyper-Heuristic Approaches: Revisited*. Springer International Publishing, Cham, 2019, pp. 453–477.

[17] Burke, E. K., Hyde, M. R., Kendall, G., and Woodward, J. R. The scalability of evolved on line bin packing heuristics. In *IEEE Congress on Evolutionary Computation* (2007), Society, IEEE Press, pp. 2530–2537.

[18] Burke, E. K., Kendall, G., and Soubeiga, E. A Tabu-search hyperheuristic for timetabling and rostering. *Journal of Heuristics 9*, 6 (2003), 451–470.

[19] Burke, E. K., McCollum, B., Meisels, A., Petrovic, S., and Qu, R. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research 176*, 1 (2007), 177–192.

[20] Burke, E. K., Petrovic, S., and Qu, R. Case-based heuristic selection for timetabling problems. *Journal of Scheduling 9*, 2 (2006), 115–132.

[21] Chakhlevitch, K., and Cowling, P. Choosing the fittest subset of low level heuristics in a hyperheuristic framework. In *Evolutionary Computation in Combinatorial Optimization* (2005), G. R. Raidl and J. Gottlieb, Eds., Springer Berlin Heidelberg, pp. 23–33.

[22] Chakhlevitch, K., and Cowling, P. Hyperheuristics: Recent developments. In *Adaptive and Multilevel Metaheuristics*, C. Cotta, M. Sevaux, and K. Sörensen, Eds. Springer Berlin Heidelberg, 2008, pp. 3–29.

[23] Chan, C. Y., Xue, F., Ip, W. H., and Cheung, C. F. A hyper-heuristic inspired by pearl hunting. In *Learning and Intelligent Optimization* (Berlin, Heidelberg, 2012), Y. Hamadi and M. Schoenauer, Eds., Springer Berlin Heidelberg, pp. 349–353.

[24] Cohn, D. L. *Measure Theory.* Birkhäuser, 1980.

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

[26] Creighton, H. B., and McClintock, B. A correlation of cytological and genetical crossing-over in Zea Mays. *Proc. Natl. Acad. Sci. 17* (1931), 492–497.

[27] Crowston, W. B., Glover, F., Thompson, G. L., and Trawick, J. D. *Probabilistic and parametric learning combinations of local job shop scheduling rules.* ONR Research memorandum. Carnegie Mellon University, 1963.

[28] Cunha, M., and Sousa, J. Water distribution network design optimization: Simulated annealing approach. *Journal of Water Resources Planning and Management 125*, 4 (1999), 215–221.

[29] Curtois, T., Ochoa, G., Hyde, M., and Vázquez-Rodríguez, J. A. A HyFlex module for the personnel scheduling problem. Tech. rep., Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, 2010.

[30] Darwin, C. *The origin of species.* John Murray, London, 1859.

[31] Dempster, P., and Drake, J. H. Two frameworks for cross-domain heuristic and parameter selection using harmony search. In *Harmony Search Algorithm: Proceedings of the 2nd International Conference on Harmony Search Algorithm (ICHSA2015)* (2016), H. J. Kim and W. Z. Geem, Eds., Springer Berlin Heidelberg, pp. 83–94.

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

[33] Diaconis, P., and Graham, R. L. Spearman's footrule as a measure of disarray. *Journal of the Royal Statistical Society. Series B 39*, 2 (1977), 262–268.

[34] Dietterich, T. G. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation 10* (1998), 1895–1923.

[35] Dorigo, M., and Stützle, T. *Ant Colony Optimization.* The MIT Press, 06 2004.

[36] Drake, J. H., Kheiri, A., Özcan, E., and Burke, E. K. Recent advances in selection hyper-heuristics. *European Journal of Operational Research* (2019).

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

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

[39] Eiben, A. E., and Smith, J. E. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer Berlin Heidelberg, 2007.

[40] Elman, J. L. Finding structure in time. *Cognitive Science 14*, 2 (1990), 179–211.

[41] Elman, J. L. Distributed representations, simple recurrent networks, and grammatical structure. *Machine Learning 7* (1991), 195–224.

[42] Eusuff, M. A., and Lansey, K. E. Optimization of water distribution network design using the shuffled frog leaping algorithm. *Journal of Water Resources Planning and Management 129*, 3 (2003), 210–225.

[43] Ezzeldin, R., Djebedjian, B., and Saafan, T. Integer discrete particle swarm optimization of water distribution networks. *Journal of Pipeline Systems Engineering and Practice 5*, 1 (2014), 04013013.

[44] Fang, H.-L., Ross, P., and 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* (1993), Morgan Kaufmann, pp. 375–382.

[45] Farmani, R., Savic, D. A., and Walters, G. A. EXNET benchmark problem for multi-objective optimization of large water systems. In *Modelling and Control for Participatory Planning and Managing Water Systems, IFAC Workshop, Venice, Italy* (2004).

[46] Fisher, H., and Thompson, G. Probabilistic learning combinations of local job-shop scheduling rules. In *Industrial scheduling*, J. Muth and G. Thompson, Eds. Prentice Hall, 1963, pp. 225–251.

[47] Fujiwara, O., and Khang, D. B. A two-phase decomposition method for optimal design of looped water distribution networks. *Water Resources Research 26*, 4 (1990), 539–549.

[48] García, S., Fernández, A., Luengo, J., and Herrera, F. A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. *Soft Computing 13*, 10 (2008), 959.

[49] García, S., Molina, D., Lozano, M., and Herrera, F. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 special session on real parameter optimization. *Journal of Heuristics 15*, 6 (2008), 617.

[50] Garrido, P., and Castro, C. Stable solving of CVRPs using hyperheuristics. In *Genetic and Evolutionary Computation Conference, GECCO 2009, Proceedings, Montreal, Québec, Canada, July 8-12, 2009* (2009), pp. 255–262.

[51] Garrido, P., and Riff, M. C. DVRP: a hard dynamic combinatorial optimisation problem tackled by an evolutionary hyper-heuristic. *Journal of Heuristics 16*, 6 (2010), 795–834.

[52] Geem, Z. W. Optimal cost design of water distribution networks using harmony search. *Engineering Optimization 38*, 3 (2006), 259–277.

[53] Gessler, J. Pipe network optimization by enumeration. In *Speciality Conference on Computer Applications / Water Resources* (1985), ASCE, pp. 572–581.

[54] Glover, F., and Laguna, M. *Tabu Search*. Kluwer Academic Publishers, 1997.

[55] Goldberg, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Professional, January 1989.

[56] Gratch, J., and Chien, S. Adaptive problem-solving for large-scale scheduling problems: A case study. *Journal of Artificial Intelligence Research 4*, 1 (1996), 365–396.

[57] Hansen, P., and Mladenović, N. *An Introduction to Variable Neighborhood Search*. Springer US, Boston, MA, 1999, pp. 433–458.

[58] Hodges, J. L., and Lehmann, E. L. Estimates of location based on rank tests. *The Annals of Mathematical Statistics 34*, 2 (06 1963), 598–611.

[59] Hooker, J. N. *Logic-Based Methods for Global Optimization*. Wiley-Interscience, 2003.

[60] Hsiao, P.-C., Chiang, T.-C., and Fu, L.-C. A VNS-based hyper-heuristic with adaptive computational budget of local search. In *IEEE Congress on Evolutionary Computation* (June 2012), CEC, pp. 1–8.

[61] Hudson, R. S., and Gregoriou, A. Calculating and comparing security returns is harder than you think: A comparison between logarithmic and simple returns. *International Review of Financial Analysis 38* (2015), 151 – 162.

[62] Hutter, F., Hoos, H. H., Leyton-Brown, K., and Stützle, T. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research 36* (October 2009), 267–306.

[63] Hyde, M., Ochoa, G., Curtois, T., and Vázquez-Rodríguez, J. A. A HyFlex module for the maximum satisfiability (MAX-SAT) problem. Tech. rep., Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, 2010.

[64] Hyde, M., Ochoa, G., Curtois, T., and Vázquez-Rodríguez, J. A. A HyFlex module for the one dimensional bin packing problem. Tech. rep., Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, 2010.

[65] Juang, B.-H., and Rabiner, L. R. A probabilistic distance measure for hidden Markov models. *AT&T Technical Journal 64*, 2 (1985), 391–408.

[66] Keller, R. E., and Poli, R. Cost-benefit investigation of a genetic-programming hyperheuristic. In *Artificial Evolution* (Berlin, Heidelberg, 2008), N. Monmarché, E.-G. Talbi, P. Collet, M. Schoenauer, and E. Lutton, Eds., Springer Berlin Heidelberg, pp. 13–24.

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

[68] Kheiri, A., and 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. *Evolutionary Computation 25*, 3 (2017), 473–501.

[69] Kheiri, A., Keedwell, E. C., Gibson, M. J., and Savic, D. Sequence analysis-based hyper-heuristics for water distribution network optimisation. *Procedia Engineering 119* (2015), 1269–1277.

[70] Kim, J. H., Kim, T. G., Kim, J. H., and Yoon, Y. N. A study on the pipe network system design using non-linear programming. *Journal of Korea Water Resources Association 27*, 4 (1994), 59–67.

[71] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. Optimization by simulated annealing. *Science 220*, 4598 (1983), 671–680.

[72] Koza, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[73] Lee, S. C., and Lee, S. I. Genetic algorithms for optimal augmentation of water distribution networks. *Journal of Korea Water Resources Association 34*, 5 (2001), 567–575.

[74] Lehre, P. K., and 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)* (2013), ACM, pp. 97–104.

[75] Lourenço, H. R., Martin, O. C., and Stützle, T. Iterated Local Search: Framework and applications. In *Handbook of Metaheuristics*, M. Gendreau and J.-Y. Potvin, Eds., vol. 146 of *International Series in Operations Research & Management Science*. Springer, 2010, ch. 9, pp. 363–397.

[76] Lourenço, H. R., Martin, O. C., and Stützle, T. *Iterated Local Search*. Springer US, Boston, MA, 2003, pp. 320–353.

[77] Lu, C., Schwier, J. M., Craven, R. M., Yu, L., Brooks, R. R., and Griffin, C. A normalized statistical metric space for hidden Markov models. *IEEE Transactions on Cybernetics 43*, 3 (2013), 806–819.

[78] Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[79] Marín-Blázquez, J. G., and Schulenburg, S. A hyper-heuristic framework with XCS: Learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In *Learning Classifier Systems* (Berlin, Heidelberg, 2007), T. Kovacs, X. Llorà, K. Takadama, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, Eds., Springer Berlin Heidelberg, pp. 193–218.

[80] Misir, M., Verbeeck, K., Causmaecker, P. D., and Berghe, G. V. The effect of the set of low-level heuristics on the performance of selection hyper-heuristics. In *Parallel Problem Solving from Nature* (2012), C. A. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, Eds., Springer Berlin Heidelberg, pp. 408–417.

[81] Misir, M., Verbeeck, K., Causmaecker, P. D., and Berghe, G. V. *An Intelligent Hyper-Heuristic Framework for CHeSC 2011*. LION 6. Springer Berlin Heidelberg, 2012, pp. 461–466.

[82] Misir, M., Verbeeck, K., Causmaecker, P. D., and Berghe, G. V. A new hyper-heuristic as a general problem solver: an implementation in HyFlex. *Journal of Scheduling 16*, 3 (2013), 291–311.

[83] Mohan, S., and Babu, K. S. J. Optimal water distribution network design with honey-bee mating optimization. *Journal of Computing in Civil Engineering 24*, 1 (2010), 117–126.

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

[85] Ortiz-Bayliss, J. C., Terashima-Marín, H., Özcan, E., Parkes, A. J., and 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)* (2013), pp. 3307–3314.

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

[87] Pareto, V. *Trattato di Sociologia Generale (The Mind and Society)*. Firenze, G. Barbéra, 1916.

[88] Partridge, D., and Yates, W. B. Replicability of neural computing experiments. *Complex Systems 10*, 4 (1996), 257–281.

[89] Petrovic, S., and Epstein, S. L. Random subsets support learning a mixture of heuristics. *International Journal on Artificial Intelligence Tools 17*, 3 (2008), 501–520.

[90] Pillay, N., and Banzhaf, W. A study of heuristic combinations for hyper-heuristic systems for the uncapacitated examination timetabling problem. *European Journal of Operational Research 197*, 2 (2009), 482–491.

[91] Poli, R., and Graff, M. There is a free lunch for hyper-heuristics, genetic programming and computer scientists. In *Genetic Programming* (Berlin, Heidelberg, 2009), L. Vanneschi, S. Gustafson, A. Moraglio, I. D. Falco, and M. Ebner, Eds., Springer Berlin Heidelberg, pp. 195–207.

[92] Qu, R., and Burke, E. K. Hybrid variable neighborhood hyperheuristics for exam timetabling problems. In *MIC2005: The Sixth Metaheuristics International Conference* (2005).

[93] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, 2018.

[94] Raad, D. N., Sinske, A. N., and van Vuuren, J. H. Comparison of four reliability surrogate measures for water distribution systems design. *Water Resources Research 46*, 5 (2010), W05524.

[95] Rabiner, L. R. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceddings of the IEEE 77*, 2 (1989), 257–286.

[96] Reca, J., and Martínez, J. Genetic algorithms for the design of looped irrigation water distribution networks. *Water Resources Research 42*, 5 (2006), W05416.

[97] Remde, S., Cowling, P., Dahal, K., and Colledge, N. Exact/heuristic hybrids using rVNS and hyperheuristics for workforce scheduling. In *Evolutionary Computation in Combinatorial Optimization*. Springer Berlin Heidelberg, 2007, pp. 188–197.

[98] Remde, S., Cowling, P., Dahal, K., Colledge, N., and Selensky, E. An empirical study of hyperheuristics for managing very large sets of low level heuristics. *Journal of the Operational Research Society 63*, 3 (2012), 392–405.

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

[100] Rossman, L. A. *EPANET2 users manual*. U.S. Environment Protection Agency, 2000.

[101] Sabar, N. R., and Kendall, G. Population based Monte Carlo tree search hyper-heuristic for combinatorial optimization problems. *Information Sciences 314*, C (2015), 225–239.

[102] Schaake, J. C., and Lai, F. H. Linear programming and dynamic programming application to water distribution network design. Tech. rep., M.I.T. Hydrodynamics Laboratory, 1969.

[103] Schumacher, C., Vose, M. D., and Whitley, L. D. The no free lunch and problem description length. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001* (2001), Morgan Kaufmann, pp. 565–570.

[104] Sherali, H. D., Subramanian, S., and Loganathan, G. V. Effective relaxations and partitioning schemes for solving water distribution network design problems to global optimality. *Journal of Global Optimization 19*, 1 (2001), 1–26.

[105] Shirzad, A. Shortening the search time in optimization of water distribution networks. *Urban Water Journal 14*, 10 (2017), 1038–1044.

[106] Smith, T. F., and Waterman, M. S. Identification of common molecular subsequences. *Journal of molecular biology 147*, 1 (1981), 195–197.

[107] Soria-Alcaraz, J. A., Ochoa, G., Sotelo-Figeroa, M. A., and Burke, E. K. A methodology for determining an effective subset of heuristics in selection hyper-heuristics. *European Journal of Operational Research 260*, 3 (2017), 972–983.

[108] Soria-Alcaraz, J. A., Ochoa, G., Swan, J., Carpio, M., Puga, H., and Burke, E. K. Effective learning hyper-heuristics for the course timetabling problem. *European Journal of Operational Research 238*, 1 (2014), 77 – 86.

[109] Srinivas, N., and Deb, K. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation 2*, 3 (1994), 221–248.

[110] Storn, R., and Price, K. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization 11*, 4 (1997), 341–359.

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

[112] Terashima-Marín, H., Ross, P., Farías-Zárate, C. J., López-Camacho, E., and Valenzuela-Rendón, M. Generalized hyper-heuristics for solving 2D regular and irregular packing problems. *Annals of Operations Research 179*, 1 (2010), 369–392.

[113] Tsoumakas, G., Vrakas, D., Bassiliades, N., and Vlahavas, I. Using the k nearest problems for adaptive multicriteria planning. In *in Proceedings of the 3rd Hellenic Conference on Artificial Intelligence, SETN04* (2004), Springer, pp. 132–141.

[114] Uludağ, G., Kiraz, B., Etaner-Uyar, A., and Özcan, E. A framework to hybridize PBIL and a hyper-heuristic for dynamic environments. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature - Volume Part II* (2012), Springer-Verlag, pp. 358–367.

[115] Vázquez-Rodríguez, J. A., Ochoa, G., Curtois, T., and Hyde, M. A HyFlex module for the permutation flow shop problem. Tech. rep., Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science, University of Nottingham, 2010.

[116] Veek, N., Repinek, M., and Mernik, M. On the influence of the number of algorithms, problems, and independent runs in the comparison of evolutionary algorithms. *Applied Soft Computing 54*, C (2017), 23–45.

[117] Walker, D. J., and Keedwell, E. C. Multi-objective optimisation with a sequence-based selection hyper-heuristic. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion* (2016), GECCO '16 Companion, ACM, pp. 81–82.

[118] Walker, J. D., Ochoa, G., Gendreau, M., and Burke, E. K. Vehicle routing and adaptive iterated local search within the HyFlex hyper-heuristic framework. In *Learning and Intelligent Optimization - 6th International Conference, LION 6, Paris, France, January 16-20, 2012, Revised Selected Papers* (2012), pp. 265–276.

[119] Wallace, R. J. Analysis of heuristic synergies. In *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2006, pp. 73–87.

[120] Wang, Q., Guidolin, M., Savic, D., and Kapelan, Z. Two-objective design of benchmark problems of a water distribution system via MOEAs: Towards the best-known approximation of the true Pareto front. *Journal of Water Resources Planning and Management 141*, 3 (2015), 04014060.

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

[122] Yang, S., and Yao, X. Population-based incremental learning with associative memory for dynamic environments. *IEEE Transactions on Evolutionary Computation 12*, 5 (2008), 542–561.

[123] Yates, W. B., and 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* (2017), GECCO, ACM, pp. 119–120.

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

[125] Yates, W. B., and Keedwell, E. C. Analysing heuristic performance for optimising water distribution networks. In *17th International Computing and Control for the Water Industry Conference (CCWI)* (2019), Extended Abstract.

[126] Yates, W. B., and Keedwell, E. C. An analysis of heuristic subsequences for offline hyper-heuristic learning. *Journal of Heuristics 25*, 3 (2019), 399–430.

[127] Yates, W. B., and Keedwell, E. C. Combining online and offline learning for a sequence-based selection hyper-heuristic. *In preparation* (2020).

[128] Yates, W. B., and Keedwell, E. C. Offline learning with a selection hyper-heuristic: An application to water distribution network optimisation. *Evolutionary Computation*, accepted (2020).

[129] Zheng, F., Zecchin, A. C., and Simpson, A. R. Self-adaptive differential evolution algorithm applied to water distribution system optimization. *Journal of Computing in Civil Engineering 27*, 2 (2013), 148–158.