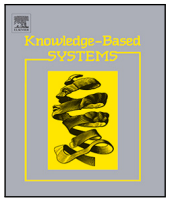
Knowledge-Based Systems 295 (2024) 111731

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys



Constructing selection hyper-heuristics for open vehicle routing with time delay neural networks using multiple experts

Raras Tyasnurita [a], Ender Özcan [a], John H. Drake [b,*], Shahriar Asta [c]

[a] Computational Optimisation and Learning Lab, University of Nottingham, UK
[b] School of Computing and Mathematical Sciences, University of Leicester, UK
[c] TD Bank Group, Canada

ARTICLE INFO

Keywords:
Combinatorial optimisation
Metaheuristics
Vehicle routing
Apprenticeship learning
Machine learning

ABSTRACT

Hyper-heuristics are general purpose search methods for solving computationally difficult problems. A selection hyper-heuristic is composed of two key components: a heuristic selection method and move acceptance criterion. Under an iterative single-point search framework, a solution is modified by selecting and applying a predefined low-level heuristic, with a decision then taken to accept or reject the resulting solution. Designing a selection hyper-heuristic is an extremely challenging task. In this study, we investigate computer-aided design of a selection hyper-heuristic for the open vehicle routing problem. A time delay neural network is used as an offline apprenticeship learning method. Our approach first observes the search behaviour of multiple expert human-designed selection hyper-heuristics on a selected sample of training instances, before automatically generating a selection hyper-heuristic capable of solving unseen instances effectively. The proposed approach is tested on open vehicle routing problem instances of different sizes to examine the performance and generality of the selection hyper-heuristics generated. Improved performance is demonstrated over a set of well-known benchmarks from the literature when compared to using the existing expert systems directly.

1. Introduction

In recent years, the Open Vehicle Routing Problem (OVRP) has received significant attention from researchers in the field of operational research. This is largely due to this variant of the vehicle routing problem capturing various real-world complexities of importance in practice [1]. OVRPs are often encountered when a business eliminates the routing cost of the return path to the depot, allowing a vehicle's route to terminate at the last customer served. As a result, the solution to an OVRP instance is a set of open routes, otherwise known as Hamiltonian paths, rather than cycles [2]. This feature in the problem formulation distinguishes the OVRP from the standard VRP. The OVRP has become increasingly important in recent years, particularly as the prevalence of the use of contractors with their own vehicles for deliveries has grown substantially [3].

There are a number of exact algorithms proposed for solving OVRPs, such as Branch-and-Cut [4] and improved Branch-Cut-and-Price [5]. However, despite their ability to obtain optimal solutions, such algorithms typically require extensive computational time to execute. The OVRP is proven to be an NP-hard optimisation problem [6]. Consequently, heuristic and metaheuristic optimisation algorithms are often deployed, producing high-quality solutions within acceptable computational times, but with no guarantee of returning an optimal solution. There are many previous studies applying various metaheuristic approaches to OVRPs with some success, including variable neighbourhood search [7] and evolutionary-based methods [8]. A comprehensive review of the algorithms used to solve OVRPs can be found in Li et al. [3].

Hyper-heuristics have emerged as general-purpose search methods which are capable of solving computationally difficult problems [9,10]. There are two main categories of hyper-heuristics: generation hyper-heuristics that create heuristics using existing algorithmic components, and selection hyper-heuristics that control a given set of predefined low-level heuristics. A typical selection hyper-heuristic consists of two main components that are critical to performance: the heuristic selection method and a move acceptance criterion. At each step of a search, the heuristic selection method chooses and applies a low-level heuristic from a pool of available heuristics, with the resulting solution considered to replace the original based on the move acceptance criterion. Designing these components effectively is a non-trivial process. Although in most cases the paradigms of heuristic selection

* Corresponding author.
E-mail addresses: raras.tyasnurita@nottingham.ac.uk (R. Tyasnurita), ender.ozcan@nottingham.ac.uk (E. Özcan), john.drake@leicester.ac.uk (J.H. Drake), shahriarasta@gmail.com (S. Asta).

https://doi.org/10.1016/j.knosys.2024.111731
Received 4 July 2023; Received in revised form 6 March 2024; Accepted 30 March 2024
Available online 18 April 2024

and heuristic generation are separate, some previous effort has been dedicated to the *generation* of selection hyper-heuristics [10]. In this paper we will present an approach to automatically generate these two key components of a selection hyper-heuristic.

There is evidence in the literature to support the improvement of optimisation algorithms using machine learning approaches [11]. There are many ways of doing so in hyper-heuristics, including using machine learning as a control mechanism, as a surrogate to accelerate objective value calculation, or to select the best optimisation algorithm for a given situation. However, fewer studies exist that create optimisation algorithms directly, and in many of these cases Genetic Programming is typically used to do so [12]. This work substantially extends our previous work [13], exploring the use of a data science technique for computer-aided design of a selection hyper-heuristic. Using the OVRP as a case study, we propose a novel offline-learning general-purpose approach, based on the concept of apprenticeship learning [14]. We employ a time-delay neural network as a generation hyper-heuristic, using apprenticeship learning to automatically construct selection hyper-heuristics that control a set of predefined low-level heuristics for the OVRP. Although Tyasnurita et al. [13] demonstrated the potential of generating selection hyper-heuristics in this manner, only the heuristic selection component was generated, based on observing the behaviour of a single expert approach. Here, we generate multiple components, including heuristic selection and move acceptance methods for each low-level heuristic, observing and learning from the behaviours of multiple existing expert approaches. We perform extensive experiments using a large benchmark of well-known OVRP instances with different characteristics, using two state-of-the-art selection hyper-heuristics as experts to train the proposed method. The empirical results show that the proposed approach is indeed capable of automatically constructing effective selection hyper-heuristics, observing and improving upon how experts perform heuristic selection and move acceptance. An additional contribution is the introduction of a new approach to run-time analysis of hyper-heuristics, using a method adapted from evolutionary activity plots.

Section 2 provides an overview of the relevant background literature, introducing selection and generation hyper-heuristics, data science techniques and hyper-heuristics, and time-delay neural networks. Section 3 introduces the open vehicle routing problem, before Section 4 provides the details of the proposed methodology. In Section 5 we present a set of computational experiments and results, including an analysis of the proposed approach. Finally, Section 6 concludes the work providing a summary and directions for future research.

2. Background

2.1. Selection and generation hyper-heuristics

Hyper-heuristics are a set of approaches that operate at a higher level of abstraction than traditional computational search techniques [10]. Rather than operating over a set of solutions directly, hyper-heuristics operate in a search space of heuristics or heuristic components. Burke et al. [9] identified two main classes of hyper-heuristics: selection hyper-heuristics and generation hyper-heuristics. Whilst selection hyper-heuristic choose between a set of existing low-level heuristics which to apply at a given point in the search, generation hyper-heuristics aim to generate new heuristics from existing heuristics or heuristic components. A high-level overview of these two paradigms is provided in Fig. 1.

Selection hyper-heuristics typically operate on a single solution, using a set of low-level heuristics from which a heuristic is repeatedly selected and applied, yielding a new solution each iteration. At each step a decision is made whether to accept the new solution (i.e. the move). This continues until a given termination criterion is satisfied. Selection hyper-heuristics have been used to tackle a wide variety of problems in the literature previously, including: examination timetabling [16], vehicle routing [17], nurse rostering [18], knapsack [19,20], software testing [21] and scheduling problems [22,23]. The first Cross-domain Heuristic Search Challenge (CHeSC 2011) was a competition designed to provide focus to the growing interest in selection hyper-heuristics, providing a standardised benchmark to allow different methods to be compared fairly. The winner of this international competition, AdapHH [24], was demonstrated to outperform 19 other entrants across six problem domains. AdapHH works on the basis that the nature of the low-level heuristics available are not known in advance. It attempts to identify which heuristics perform well together in an online manner. AdapHH was shown to demonstrate strong generalisation capabilities, adapting well to unseen problem domains and instances, and was long considered the state-of-the-art in selection hyper-heuristics. Multi-Stage Hyper-heuristic (MSHH) [25] is a more recent selection hyper-heuristic that was demonstrated to outperform AdapHH. MSHH utilises and invokes multiple interacting hyper-heuristics cyclically, controlling the transition between the hyper-heuristics available. In this paper, we use these two state-of-the-art hyper-heuristics as 'experts' for apprenticeship learning, with the hypothesis that learning behaviours from the combination of both of these experts will deliver improved performance.

There is also considerable research interest in generation hyper-heuristics, which aim to automate the process of designing heuristics, metaheuristics and hyper-heuristics. Creating heuristics is a challenging process, attempting to generate novel heuristics by searching the space of potential heuristic components. Typically, rule abstractions designed by humans representing move operators constitute the low-level heuristic search space that a selection hyper-heuristic works with. Several techniques in this area aim to construct heuristics which are competitive with human-designed heuristics. Evolutionary algorithms, such as Genetic Programming (GP), are often used to automatically generate heuristics [12]. Automated heuristic generation with GP has been explored within the context of a range of problems including to create strip packing heuristics [26], heuristics for project scheduling [27], and evolving reusable constructive heuristics for the multidimensional 0-1 knapsack problem [28]. Alternative approaches to GP for automated heuristic generation include Grammatical Evolution, previously used to generate heuristics for various problem domains including bin packing [29], vehicle routing [30], and boolean satisfiability [31]. More recently research focus has shifted towards generating selection hyper-heuristics directly, rather than just the low-level heuristics they operate with, often incorporating ideas and techniques from data science. Section 2.3 will discuss approaches that combine these concepts.

2.2. Information flow in selection hyper-heuristics

As shown in Fig. 1, the traditional hyper-heuristic framework consists of an upper layer at the hyper-heuristic level where the low-level heuristics are selected or generated, and a lower layer at the problem domain level where the low-level heuristics are applied to the problem domain. This basic framework was proposed by Cowling et al. [15], under the assumption that generality and re-usability of algorithmic components can be achieved by imposing a strict domain barrier concept between these two layers. This barrier restricts the information flow between layers, with only domain-independent information able to pass through the barrier, such as objective function values and low-level heuristic type details. Since it does not deal with the solution space directly, the data provided is minimal or highly abstract. Swan et al. [32] argued against this accepted notion of the domain barrier, presenting a new framework that demonstrates the usefulness of allowing some universal problem domain knowledge to pass through the domain barrier, without the loss of generality.

The availability of an arbitrarily rich set of features is one of the key ingredients to success when applying machine learning techniques.

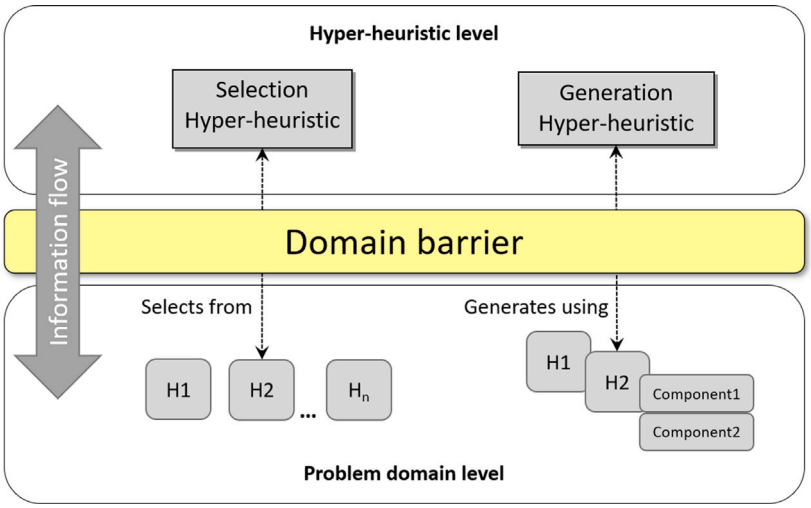


Fig. 1. High-level overview of the two main classes of hyper-heuristic based on the original framework of Cowling et al. [15].

Furthermore, several studies have shown that allowing some lower-level information to pass through the domain barrier can significantly improve the performance of hyper-heuristics [20,33,34]. This includes our previous study [13], which exploited not only domain-independent knowledge such as objective function values, but also general domain-level knowledge in the form of the distance between solutions. The empirical results indicated that allowing a richer information environment can bring advantages to the learning process. We follow this approach in our work here, allowing distance metrics to pass through the domain barrier to be used by the components at the higher hyper-heuristic level.

2.3. Data science techniques and hyper-heuristics

A close link exists between data analytics and optimisation algorithms. Combining data science and optimisation approaches can offer significant opportunities for improved performance in terms of the quality of solutions found. Early work tended to focus on data mining approaches. Thabtah and Cowling [35] presented associative classification algorithms to predict the best heuristic for a selection hyper-heuristic to choose. To identify an effective classification technique, the authors compared six algorithms, consisting of three associative (CBA, MCAR, MMAC) and three traditional classifiers (C4.5, RIPPER, PART), in terms of their ability to extract useful rules for heuristic selection. The experimental results showed that in general the associative classifiers outperformed the traditional methods. Li et al. [36] used artificial neural networks and boolean logistic regression to support graph-based hyper-heuristics that construct solutions for exam timetabling problems. Tapia-Avitia et al. [37] also deployed artificial neural networks, attempting to learn patterns in sequences of heuristics that lead to strong performance in continuous optimisation problems.

A number of recent studies related to the area of data science and hyper-heuristics have deployed tensor-based hyper-heuristics as an advanced machine learning technique. Tensor analysis has been applied to a range of problems, including the six CHeSC 2011 problem domains [38] and nurse rostering [18]. The empirical results indicate that significant overall performance improvement is possible using this approach. Another related technique is Case-Based Reasoning (CBR), previously used as a hyper-heuristic for course timetabling problems [39]. This work stores previously encountered similar problems in a source case, then adapts to fit a new problem based on these cases. The source is then updated with the newly solved problems. There is also work contributing to the hybridisation of Reinforcement Learning (RL) with meta- or hyper-heuristics [40]. These RL approaches learn to generate good heuristics by gathering experience whilst solving problems. The authors illustrated that using learning automata to search the policy space in RL can boost the performance of meta/hyper-heuristics. More recently, Q-learning has been used as an RL algorithm, where each pair of state and action is given a total cumulative reward value (Q-value) to guide the selection of appropriate components in a hyper-heuristic [41,42].

Another data science technique which has been shown to generate selection hyper-heuristics effectively is apprenticeship learning. Previous work has adopted apprenticeship learning to solve online bin packing problems by using k-means clustering for policy generation [43], and for vehicle routing by a decision tree constructed from a C4.5 classifier [44]. Our previous study [45] also investigated the vehicle routing problem, with the learning component based on a multilayer perceptron classifier taken from the field of artificial neural networks. The computational experiments demonstrated the success of the proposed approach, outperforming other previously suggested methods, including C4.5 apprenticeship learning. Moreover, neural networks were used by Smith-Miles [46] to model the relationships in the meta-data between features extracted from problem instances and algorithm performance. The models produced consist of rules which are relevant to assisting with automated algorithm selection.

In addition to hyper-heuristics, data science and machine learning methods can be used to enhance the performance of other optimisation techniques [11]. In particular, data mining approaches have been used to extract patterns from a population of solutions, which are then used to guide the search process of evolutionary algorithms. Early work by Le and Ong [47] used frequent pattern mining to identify 'building blocks' within genetic algorithms, in order to provide insight into the search process over time. Chia et al. [48] used frequent itemset mining to inform the mutation operator within a memetic algorithm, identifying promising sub-regions of the domain of each variable in real-valued optimisation problems. More recently, Zhou et al. [49] presented a frequent pattern-based search (FPBS) method, which also used data mining to guide the generation of new solutions in the evolutionary process of a memetic algorithm. A similar approach was proposed by Zhou et al. [50], which used frequent itemset data mining to drive the recombination process in a memetic algorithm for the α-separator problem.

2.4. Time delay neural networks

Time delay neural networks (TDNN) are an artificial neural network architecture often used for data classification [51,52]. TDNNs are similar in structure to Multi-Layer Perceptrons (MLP), consisting of three

layers: input, hidden, and output, and all connections are feedforwards. A TDNN is a multi-layer feedforward network whose hidden and output neurons are replicated across time [53]. Because of the similarities between the structures of MLPs and TDNNs, backpropagation can be used to train a TDNN. Backpropagation is measured as the gradient descent of the mean-squared error of the difference between the desired output and the observed network output. The derivative of the error is propagated back through the network, and the weights adjusted accordingly. This operation is repeated several times for all training data until the network yields the desired output [54]. The only aspect that distinguishes training a TDNN is that the connection weights are updated according to the average rather than the independent error signal [55]. Although TDNNs were initially designed for speech recognition tasks, they are capable of strong performance when dealing with any sequential data (e.g. time series), by supplementing the input with time-delayed copies of previous inputs. Previous work in the literature covers a wide range of applications, such as traffic flow prediction [56] and image sequence analysis [57].

Compared with models based on MLPs, TDNN model prediction can be more accurate and robust [58,59]. The complexity of implementing a TDNN is moderate compared to Recurrent Neural Networks (RNN), and does not require extensive memory storage requirements [60]. In TDNNs, the inputs to any neuron/node i can include the outputs of earlier nodes, both for the current time step t and some defined number d of previous time steps $(t-1, t-2, \ldots t-d)$. This is done using tap-delay lines [61] which affect the input–output mapping. Previous work by Peddinti et al. [62] indicated that TDNNs can effectively model long-term temporal dependencies via this mechanism, particularly when addressing large data scenarios. As the weights in TDNNs are vectors, selecting the length of the weight vector is a crucial decision. If the length is too long, the model cannot respond to the input changes in a timely manner. While if it is too short, the model cannot represent the large-time-delay property of the system. The optimal weight length depends on the application and the computational budget available [63].

Based on the properties above, and our own previous success generating the heuristic selection component of a selection hyper-heuristic with TDNNs [13], in this paper we use TDNNs to model the heuristic selection and move acceptance behaviour of two expert selection hyper-heuristics, generating complete selection hyper-heuristics for the OVRP via apprenticeship learning.

3. Open Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is a field of study with a broad definition and a large number of variants [64–67]. The VRP literature has increased significantly over time since it was first proposed by Dantzig and Ramser in the late 1950s [68]. For the interested reader, a comprehensive review of the VRP and its variants can be found in Eksioglu et al. [69]. A solution to the VRP aims to provide an optimal distribution of deliveries from a central depot to some customers with known demands, subject to a particular set of constraints. The typical main objective in a VRP is to minimise the cost of vehicle routes, such as route length, total driving costs or driving time, while the general constraints of VRP relate to the vehicle capacities or type, and number of customers to be served. In a standard VRP, the solution approach needs to identify a sequence of deliveries to customers for a fleet of fixed capacity vehicles based at a single depot, where each vehicle must leave from and return to the same depot.

The Open Vehicle Routing Problem (OVRP) was introduced by Schrage [70], who described certain characteristics of some real-life VRPs. In the OVRP variant, vehicles are not required to return to the depot at the end of a route, so a route ends after servicing the last customer. OVRPs often relate to cases where private vehicles are used, since these vehicles tend to have a higher cost when forced to return to the depot. Another issue is that the vehicles available may be unable to satisfy the customer demand when the cost of returning to the depot is included. Bodin et al. [71] presented an OVRP case study within the FedEx Express company. Following this, interest in the OVRP was limited for a number of years until the work of Sariklis and Powell in 2000 [72], who presented a sequential two-phase method called Cluster-First Route-Second (CFRS), using a minimum spanning tree with penalties procedure. Brandão [73] presented improved results on the same benchmarks, using a Tabu Search based approach which utilised insertion and swap operators between two routes in a solution. Fleszar et al. [7] implemented a variable neighbourhood search algorithm which explored different neighbourhoods, allowing segments to be exchanged between routes and subroutes within a single route to be reversed. Li et al. [3] deployed a record-to-record travel algorithm (ORTR), providing a comparison to a number of existing approaches, in addition to introducing a new benchmark set of larger OVRP instances.

Following the introduction this new benchmark, interest in applying metaheuristics to the OVRP increased considerably. Repoussis et al. [8] introduced a hybrid evolution strategy (HES), integrating a memory-based trajectory local search algorithm within an evolutionary algorithm framework, achieving some new best solutions. Subramanian et al. [74] hybridised Iterated Local Search and Set Partitioning (ILS-RVND-SP), showing strong performance over benchmarks from a variety of VRP variants. The guided local search method (GELS) of Hosseinabadi et al. [75] was demonstrated to offer very strong results, matching or outperforming HES on all but two of the large instances tested. Ozcetin et al. [76] considered a real-world distribution problem, combining truck loading with vehicle routing. An exact approach was used to plan the deliveries of fully-loaded trucks, whilst a hybrid genetic algorithm was used to solve the open vehicle routing problem for partial loads.

Although many previous studies have tended to solve relatively small problem instances, recent research trends have shifted towards larger real-world problems [77]. In this study, we consider three sizes of OVRP instances taken from different sources (Standard (S) [73], Large (L) [3], and Very Large (VL) [78]), as shown in Table 1. For the instances with the same number of customers, number of vehicles, and vehicle capacity between instances, the service time and maximum route time differs.

The objective of the OVRP is to minimise the total travel $cost$, as defined for a solution S in Eq. (1):

$$cost(S) = c \times v + d \tag{1}$$

This objective function includes the cost of the number of vehicles used (v) and the distance travelled (d). We assume that any benefit of reducing the distance travelled is outweighed by the operating cost of an extra vehicle [3]. A weighting coefficient c (set to 1000 in our work) is used to emphasise the importance of reducing the number of vehicles, which is much more costly than the overall distance travelled. The primary objective is to minimise the number of vehicles v required, completing all deliveries within the least total distance travelled possible d.

4. Proposed approach

Motivated by the work described in Section 2 above, the work in this paper further investigates the potential of exploiting neural network architectures for automated selection hyper-heuristic design. The proposed approach is based on the concept of Learning from Demonstration (LfD). LfD is a technique that derives policies from gathering demonstrators examples and mapping their actions [79]. By labelling each recorded movement, a library of samples which can be used for learning from is built. Our approach is also inspired by the concept of apprenticeship learning (AL), developed by Abbeel and Ng [14]. AL is a branch of Machine Learning within which an ‘apprentice’ learning agent can observe another ‘expert’ agent’s behaviour, and has been successfully applied in the fields of robotics and control, including

Table 1
OVRP benchmark instances used to test our approach.

Instance ID	Customers	Vehicles	Vehicle capacity
S1	50	5	160
S2	75	10	140
S3	100	8	200
S4	150	12	200
S5	199	16	200
S6	50	5	160
S7	75	10	140
S8	100	8	200
S9	150	12	200
S10	199	16	200
S11	120	7	200
S12	100	10	200
S13	120	7	200
S14	100	10	200
L1	200	5	900
L2	240	9	550
L3	280	7	900
L4	320	10	700
L5	360	8	900
L6	400	9	900
L7	440	10	900
L8	480	10	1000
VL1	560	10	1200
VL2	600	15	900
VL3	640	10	1400
VL4	720	10	1500
VL5	760	21	900
VL6	800	11	1700
VL7	840	21	900
VL8	880	10	1800
VL9	960	10	2000
VL10	1040	10	2100
VL11	1120	10	2300
VL12	1200	10	2500

for flight simulation programs [80], and robot control [81]. In such approaches, supervised learning is used to find a parameterised policy from the examples provided by the experts. Previous works in AL have shown the effectiveness of using expert demonstrators [44,80,81].

With this previous work in mind, we explore the promising concept of LfD/AL for automated design of computational search algorithms. Although this study does not follow the structure of these approaches directly, the idea of student and teacher learning is represented by the generated selection hyper-heuristic and the expert hyper-heuristic(s) respectively. The proposed approach aims to generate new selection hyper-heuristics which are able to offer improved performance compared to the existing expert approaches used as demonstrators.

Fig. 2 illustrates the general process flow of the proposed approach. The overall framework consists of three successive phases, following a train and test structure. The first phase constructs datasets to use for model training. These datasets consist of the relevant actions and solution features obtained from the expert hyper-heuristics while solving a set of selected problem instances. In the second phase, the datasets are fed into a Time Delay Neural Network (TDNN) to create classifiers for heuristic selection and move acceptance to form a new selection hyper-heuristic. Finally, the third phase applies and tests the generated selection hyper-heuristic to unseen problem instances. These three phases are explained in detail in the subsequent subsections.

4.1. Phase I – Dataset construction

In this study, we use two state-of-the-art expert hyper-heuristics, Adaptive Hyper-heuristic (AdapHH) [24] and Multi-Stage Hyper-heuristic (MSHH) [25]. We will observe their search behaviour as they select a low-level heuristic and make move acceptance decisions, recording them as actions. The expectation is that by collecting this data and learning from the combination of these two expert approaches, the generated hyper-heuristics can deliver improved performance.

The two actions of an expert recorded correspond to the two main components of selection hyper-heuristics: heuristic selection and move acceptance. The input and output features of these actions are as follows:

Input features: There are two main input features to these actions, consisting of the change in the objective value ($objDiff$) and the *distance* between the solutions before and after the application of a heuristic. The distance metric used is the adjacency-based distance as defined by Kubiak [82], which counts the number of mismatched positions in a route. This metric was chosen as solutions to the VRP are represented as a permutation of customers, so the position of a customer within a route is directly related to the quality of the solution. Assuming a simple example of a single vehicle visiting three customers $\{c_1, c_2, c_3\}$, $\langle c_1, c_2, c_3\rangle$ represents a route with a cost of 10 where the vehicle starts from the location of the first customer, then goes to the second customer and finally visits the third one. The vehicle then stays there. Applying a swap operator randomly, a new route $\langle c_1, c_3, c_2\rangle$ is generated, with a cost of 6. In this case, the difference in cost would be 4 and the distance between solutions would be 2.

Output features: In addition, there is an output feature for each action. For heuristic selection, the output is simply the low-level heuristic (LLH) selected. For the move acceptance criteria, as both expert hyper-heuristics always accept improving solutions, identifying a policy to handle non-improving (i.e. either equal or worsening) solutions is important in order to avoid becoming trapped in local optima. Hence, the output feature recorded for move acceptance is whether a non-improving solution was accepted (Equal Accepted, Worsening Accepted).

We execute each of the experts on a set of selected instances from a benchmark problem, collecting data to train effective heuristic selection and move acceptance strategies with. A separate dataset is maintained for each action, with data only collected if the expert accepts a solution. While a single dataset is constructed for heuristic selection, for move acceptance a dataset is constructed for each separate LLH. This will enable the methods trained on the dataset to learn appropriate acceptance decisions following the application of a particular heuristic. Eight LLHs are employed in this study, the details of each will be explained subsequently below. Consequently, there are eight datasets collected for move acceptance criteria, with input saved every eight consecutive steps (t). An overview of the data collection process is provided in Fig. 3.

We use the HyFlex implementation of the VRP for our experiments [83], which contains ten LLHs (LLH0, ..., LLH9). However, since our hyper-heuristics operate under a single-point based search framework, we exclude the crossover operators LLH5 and LLH6. The remaining eight LLHs can be grouped based on their type, consisting of local search (LS), which conduct iterative search within the neighbourhood to improve the incumbent solution, mutation (MU) which perturb a solution randomly, and ruin-recreate (RR) which partially destroy then repair a given solution. To ensure consistency, we use the same LLH indexing as our previous work [45], more details on the implementation of each LLH can be found in Walker [84]. The mutation and local search LLHs in our study are:

- **LLH0** (MU0): *two-opt* mutation swaps two adjacent customers within a single route.
- **LLH1** (MU1): *or-opt* mutation moves two adjacent customers to a different location within a single route.
- **LLH4** (LS0): *shift* local search moves a customer from one route to another provided that the new position yields an improvement.
- **LLH7** (MU2): *shift* mutation moves a single customer from one route to another.
- **LLH8** (LS1): *two-opt* local search takes the end sections of two routes and swaps them to create two new routes.

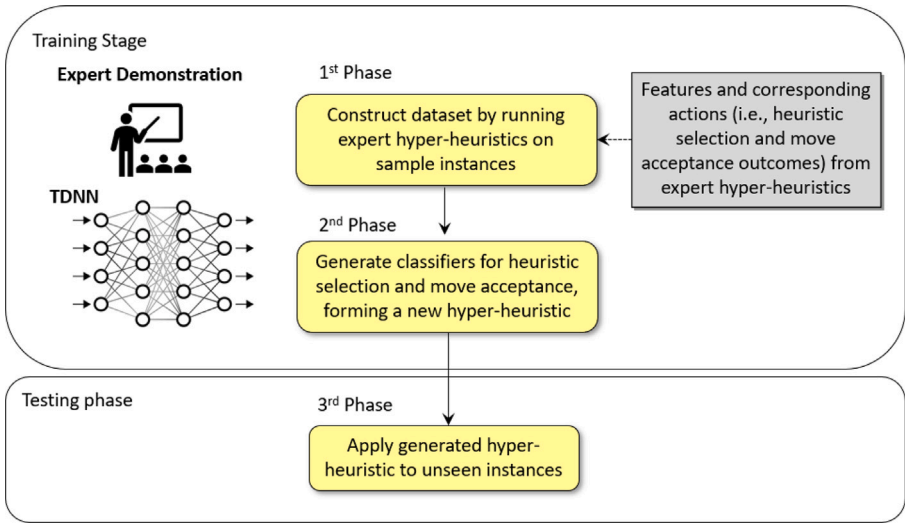


Fig. 2. Proposed apprenticeship learning framework to generate selection hyper-heuristics with Time Delay Neural Networks.

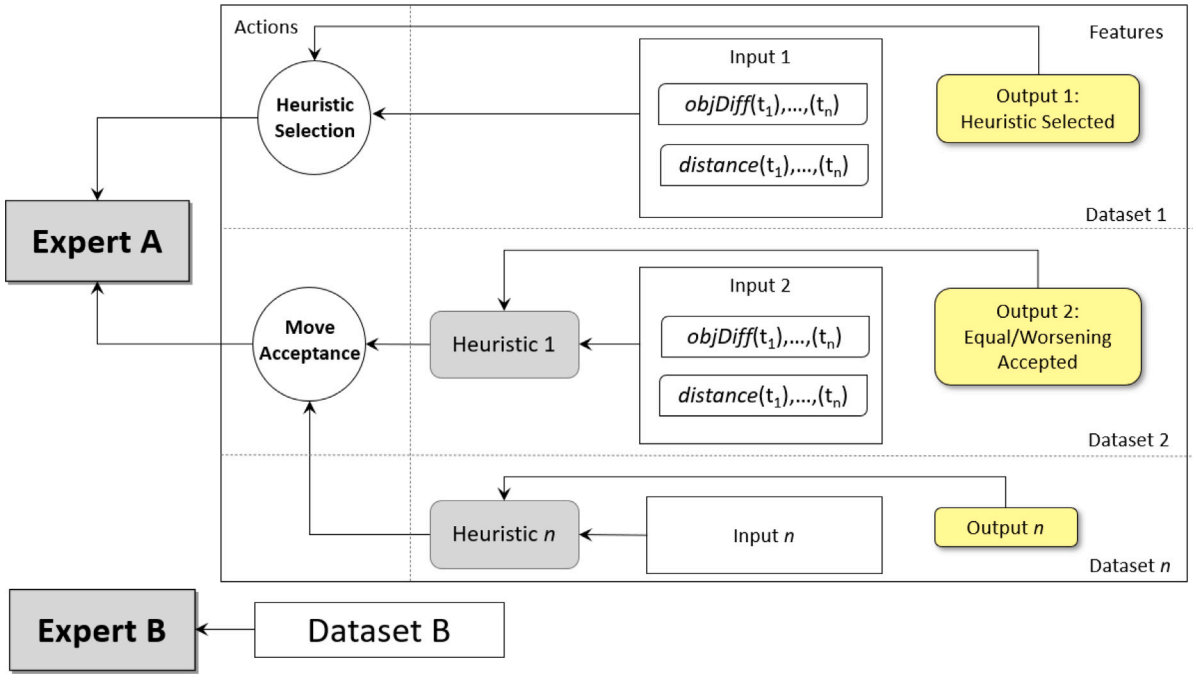


Fig. 3. Data collection from an expert hyper-heuristic while solving a problem instance.

- **LLH9** (LS2): *GENI* local search takes a customer from one route and places into another route, in between the two customers of that route which are closest to it.

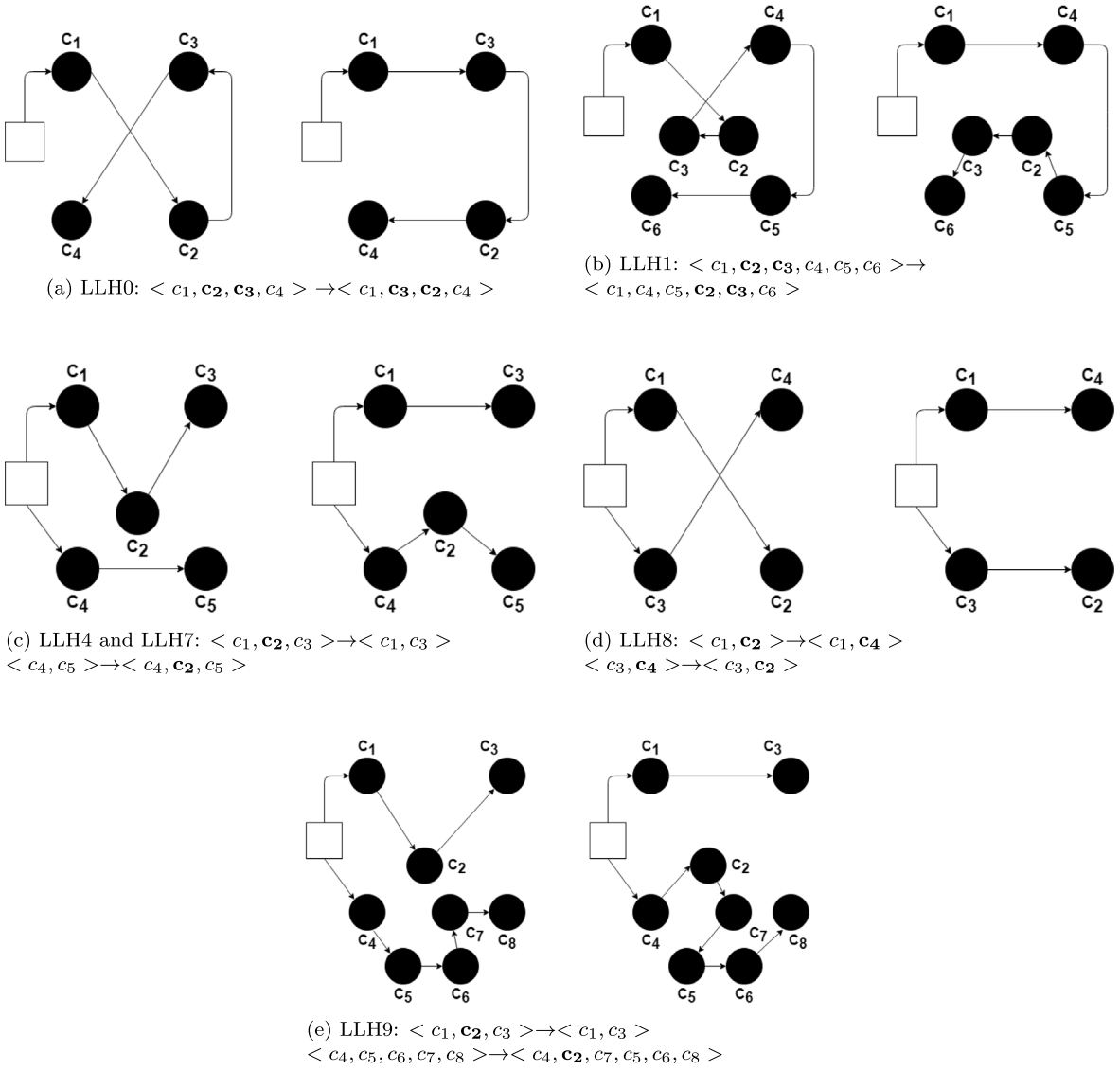
Examples of the neighbourhood moves that are made by each of the mutational and local search low-level heuristics are provided in Fig. 4. We also include the following ruin-recreate LLHs:

- **LLH2** (RR0): *location-based radial* ruin-recreate operator selects a number of customers to remove from the solution based on their proximity to a given location.
- **LLH3** (RR1): *time-based radial* ruin-recreate operator selects a number of customers to remove from the solution based on the proximity of their arrival time to a given time.

There are two parameters which control the behaviour of low-level heuristics: Intensity of Mutation (IoM) for mutational and ruin-recreate heuristics, and Depth of Search (DoS) for local search heuristics. The values for both parameters influence the number of move steps and are in the range of [0,1]. A higher DoS setting will lead to a local search heuristic searching a larger number of neighbouring solutions for an improving move. Higher values of IoM denote that more variables in the solution are modified during mutation. In the case of ruin-recreate heuristics, higher values of IoM indicate that more parts of a solution will be destroyed and rebuilt. This study uses fixed values of 0.8 and 0.4 for DoS and IoM respectively, based on the parameter tuning experiments by Gümüş et al. [85] for the vehicle routing problem domain in HyFlex.

4.2. Phase II – Generating classifiers

The current state of the search is a critical factor when deciding how to proceed in the search process using a selection hyper-heuristic. The objective value measured at a step t depends directly on the previous objective values at steps $t-1$, $t-2$, etc. A previous study by



(a) LLH0: $< c_1, \mathbf{c_2}, \mathbf{c_3}, c_4 > \rightarrow < c_1, \mathbf{c_3}, \mathbf{c_2}, c_4 >$

(b) LLH1: $< c_1, \mathbf{c_2}, \mathbf{c_3}, c_4, c_5, c_6 > \rightarrow$
$< c_1, c_4, c_5, \mathbf{c_2}, \mathbf{c_3}, c_6 >$

(c) LLH4 and LLH7: $< c_1, \mathbf{c_2}, c_3 > \rightarrow < c_1, c_3 >$
$< c_4, c_5 > \rightarrow < c_4, \mathbf{c_2}, c_5 >$

(d) LLH8: $< c_1, \mathbf{c_2} > \rightarrow < c_1, \mathbf{c_4} >$
$< c_3, \mathbf{c_4} > \rightarrow < c_3, \mathbf{c_2} >$

(e) LLH9: $< c_1, \mathbf{c_2}, c_3 > \rightarrow < c_1, c_3 >$
$< c_4, c_5, c_6, c_7, c_8 > \rightarrow < c_4, \mathbf{c_2}, c_7, c_5, c_6, c_8 >$

Fig. 4. Illustration of OVRP perturbative low-level heuristics.

Peddinti et al. [62] showed the effectiveness of TDNNs to model long-term temporal dependencies, when dealing with large data scenarios. Their results are in line with our need to learn from several previous time steps in the search process (long dependencies) to solve large-scale OVRPs. Hence, in the second phase, we generate classifiers from each dataset using TDNNs. The first and second phases complement each other, constituting the whole learning process, including data collection and training. The output from the second phase is a model representing a selection hyper-heuristic, which generalises what has been learned from the datasets, combining the heuristic selection and move acceptance classifiers together.

A TDNN model is determined by three essential factors, consisting of the network architecture, the input and activation functions of the unit, and the weight of each input connection [86]. As the first two elements are fixed, the behaviour of the TDNN is defined by the values of the weights of each connection. During training, these weights are initially set to random values, before the training set instances are repeatedly exposed to the network. Then the network output is compared with the desired output for the particular instance at hand. A minor adjustment is made to all weights in the network in the direction that would bring the observed network output values closer to the desired output.

A TDNN has certain parameters that can influence performance, in particular, the number of hidden nodes, learning rate, and momentum. We used the same parameter settings as presented in our previous study [13], which were tuned using the Taguchi design of experiments method: 24 hidden nodes, 0.07 learning rate, and 0.9 momentum.

4.3. Phase III – Applying the generated hyper-heuristic

In the final phase, we apply the generated TDNN classifiers to unseen instances from the OVRP benchmarks, independent of the instances used by the expert hyper-heuristics in the first phase. The pseudo-code of the final phase is illustrated in Algorithm 1. Firstly, a starting solution to the current problem S is initialised (Line 1). Then a number of steps are iteratively completed until a maximum time limit is reached. A heuristic h is selected and applied to the current solution based on the TDNN classifier (Lines 4–5). In the case that an improving solution is found following the application of a heuristic chosen by the TDNN classifier, it will always be accepted (Line 8). If there is no improvement found, the decision whether to accept or reject the new solution is made by the move acceptance method based on the

TDNN classifier (Line 13). The procedure will stop at the maximum time allowed and return the best solution found (Line 16).

Algorithm 1 Application of the generated TDNN hyper-heuristic

```
1: initialise feasible starting solution S
2: S_best ← S
3: while maximum time allowed is not exceeded do
4:     select a heuristic h using TDNN Heuristic Selection
5:     S' ← apply heuristic h to solution S
6:     δ = cost(S) − cost(S')
7:     if δ > 0 then
8:         S ← S'        ▷ Improving solution is accepted automatically
9:         if cost(S) < cost(S_best) then
10:            S_best ← S        ▷ Update best solution found if necessary
11:        end if
12:    else
13:        TDNN Move Acceptance criteria is applied        ▷ if S' is accepted, S ← S'
14:    end if
15: end while
16: return S_best
```

5. Computational experiments

All experiments are performed on an Intel(R) Core(TM) i7 (3.40 GHz) with 16 GB RAM. For each of the selection hyper-heuristics generated, 30 trials on each instance are executed, with each trial terminating after 600 nominal seconds based on the benchmark time limit for these instances used in previous work in the literature [13,87]. Firstly we compare the apprenticeship learning hyper-heuristic trained using both experts to selection hyper-heuristics trained using only the datasets yielded from an individual expert. Additional performance comparison to the performance of each individual expert executed independently is also provided. This comparison is done based on models trained and tested on the same type of instance (either S, L or VL), before cross-class training and testing experiments are performed (i.e. training on one type of instance, then testing on another) to assess the generalisation capability of the proposed method. Then we analyse the prediction accuracy and average accuracy per class for the TDNN apprenticeship learning hyper-heuristic trained using both experts, providing the receiver operating characteristic (ROC) curves for each low-level heuristic. Finally, we assess the performance of individual low-level heuristics and different low-level heuristic types within the generated method, comparing the selection trends to those that can be observed from the two expert hyper-heuristics.

5.1. Comparison of TDNN selection hyper-heuristics with different training sets

In this section, we first compare the performance of the proposed TDNN apprenticeship learning hyper-heuristic trained using both experts (TDNN-ALHH-Both), TDNN-ALHH based on a single expert (TDNN-ALHH-AdapHH and TDNN-ALHH-MSHH), and the individual expert hyper-heuristics (AdapHH and MSHH) applied separately. These results were obtained by training on a sample set of a single size of OVRP instances (either S, L or VL), then tested on unseen instances in the same group of problem size. The training set used three sample instances arbitrarily chosen with differing numbers of customers and vehicle capacities. The results of testing over different sets of instances (Standard, Large, Very Large, All) are summarised in Fig. 5. The average rank out of the five methods for each set of instances is provided, with a lower rank indicating better performance. For each individual instance, the average rank based on performance over 30 runs is used for this comparison.

Fig. 5 shows that the proposed method trained by observing both expert hyper-heuristics (TDNN-ALHH-Both) clearly outperforms the other methods. The average rank (out of 5) for this method across the entire benchmark is 1.62. Based on a Tukey test (within a 95% confidence interval), there is a statistically significant difference between the performance of TDNN-ALHH-Both and three of the four methods tested. However, the difference in performance between TDNN-ALHH-Both and the second best method (TDNN-ALHH-MSHH, average rank 2.12) is not statistically significantly different. For both of the individual expert methods tested (MSHH and AdapHH), there is a statistically significant difference between the performance of the apprentice trained by observing that method (TDNN-ALHH-MSHH and TDNN-ALHH-AdapHH respectively), with the apprentice outperforming the expert in both cases. The relative performance of AdapHH is particularly striking (average rank 4.50, no best performance), as this has historically been a state-of-the-art method, winning the CHeSC 2011 competition for selection hyper-heuristics. As AdapHH contains a number of online learning mechanisms, it is likely that it has to spend a lot of computational time learning effective strategies for heuristic search that are subsequently exploited by the apprentice hyper-heuristics from the beginning of the search.

In order to analyse the generalisation capability of the proposed apprenticeship leaning approach, we will also test various combinations of different size problem instances for training and testing. We test five different combinations. The first three are trained on small sized instances, then tested on Standard, Large and Very Large instances respectively. This will provide insight into whether training on smaller instances is sufficient for high performance in larger scale problem instances. The fourth combination trains on Large instances, and is tested on Very Large instances, to see if increasing the size of the training instances also increases performance. The final combination consists of a training set that includes both Standard and Large instances, and is then tested on the Very Large instances.

A performance comparison for using different size instances of OVRP in training and testing, based on Wilcoxon signed-rank tests, are given in Table 2. The values in the table represent the number of instances that a particular difference in performance is observed, where '>' indicates that *Algorithm 1* performs better than *Algorithm 2* (with a statistically significant difference in the performance within a confidence interval of 95% over 30 runs), and '<' indicates vice versa. Non-statistically significant differences in performance are denoted by '≈'. The total number of instances for each training-test set combination differs depending on the test set used, there are 14 Standard instances, 8 Large instances, and 12 Very Large instances.

Table 2 highlights the ability of the proposed method to generalise well, since statistically significantly better performance is observed in more than half of the instances tested. This table shows that the apprentices are able to consistently outperform the experts whose behaviour they observed. Again, this is likely due to the lack of overhead required to learn which heuristics are effective, and which to exclude, demonstrating the advantage of learning these characteristics in an offline manner. The subsequent subsections provide additional analysis on the behaviour and insight into the results obtained by the best approach observed in this section (TDNN-ALHH-Both).

5.2. An analysis of the classifiers generated

Data classification tasks in the field of machine learning are divided into binary, multi-class, multi-labelled and hierarchical [88]. For each of these classification sub-fields, the performance measure can be different. A classifier's evaluation is generally based on prediction accuracy, which is the percentage of correct predictions divided by the total number of predictions [86]. There are several methods for calculating a classifier's accuracy, in this work we use cross-validation. The data set is divided into equal-sized subsets, which are used for both training and testing. For each subset, the classifier is trained on the union of the remaining subsets, then tested on the original subset. The average of the error rate of each subset is therefore an estimate of the error

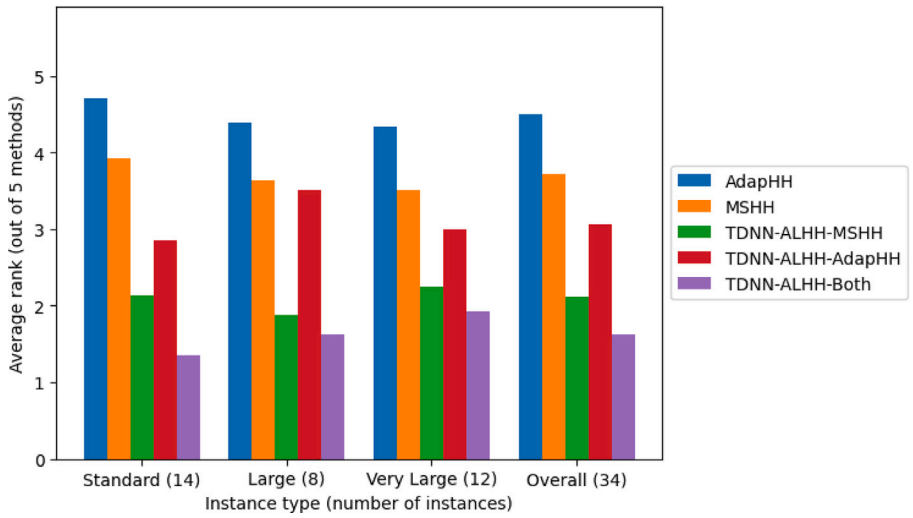


Fig. 5. Average rank over different instance sets of TDNN-ALHH employing both experts (TDNN-ALHH-Both) to TDNN-ALHH employing a single expert (TDNN-ALHH-AdapHH and TDNN-ALHH-MSHH) and the expert itself (AdapHH and MSHH).

Table 2
Performance comparison for different combinations of training and test set instance types.

Algorithm 1	Algorithm 2	>	<	≈
Standard (train) to standard (test)				
TDNN-ALHH-Both	TDNN-ALHH-AdapHH	10	1	3
TDNN-ALHH-Both	TDNN-ALHH-MSHH	10	1	3
TDNN-ALHH-Both	AdapHH	11	1	2
TDNN-ALHH-Both	MSHH	10	1	3
TDNN-ALHH-AdapHH	AdapHH	11	0	3
TDNN-ALHH-MSHH	MSHH	10	0	4
Standard (train) to large (test)				
TDNN-ALHH-Both	TDNN-ALHH-AdapHH	5	0	3
TDNN-ALHH-Both	TDNN-ALHH-MSHH	5	0	3
TDNN-ALHH-Both	AdapHH	6	1	1
TDNN-ALHH-Both	MSHH	6	0	2
TDNN-ALHH-AdapHH	AdapHH	5	2	1
TDNN-ALHH-MSHH	MSHH	5	1	2
Standard (train) to very large (test)				
TDNN-ALHH-Both	TDNN-ALHH-AdapHH	5	1	6
TDNN-ALHH-Both	TDNN-ALHH-MSHH	5	0	7
TDNN-ALHH-Both	AdapHH	5	1	6
TDNN-ALHH-Both	MSHH	5	0	7
TDNN-ALHH-AdapHH	AdapHH	6	1	5
TDNN-ALHH-MSHH	MSHH	6	0	6
Large (train) to very large (test)				
TDNN-ALHH-Both	TDNN-ALHH-AdapHH	7	3	2
TDNN-ALHH-Both	TDNN-ALHH-MSHH	7	1	4
TDNN-ALHH-Both	AdapHH	7	0	5
TDNN-ALHH-Both	MSHH	7	0	5
TDNN-ALHH-AdapHH	AdapHH	7	1	4
TDNN-ALHH-MSHH	MSHH	7	2	3
Standard + Large (train) to very large (test)				
TDNN-ALHH-Both	TDNN-ALHH-AdapHH	7	1	4
TDNN-ALHH-Both	TDNN-ALHH-MSHH	7	0	5
TDNN-ALHH-Both	AdapHH	9	2	1
TDNN-ALHH-Both	MSHH	9	1	2
TDNN-ALHH-AdapHH	AdapHH	8	2	2
TDNN-ALHH-MSHH	MSHH	8	1	3

rate of the classifier. Here, we will use cross-validation with ten folds to evaluate the accuracy of the model trained using the datasets from both expert hyper-heuristics (TDNN-ALHH-Both).

The heuristic selection classifier component of TDNN-ALHH-Both has an exact match ratio (correctly classified instances rate) of 56.7%. This performance is stronger than previous studies, which obtained an accuracy of 51.2% [13] and 50.3% [45] respectively for the heuristic selection classifier. Although a classification accuracy of around 50% may seem unimpressive at first glance, it is imperative to note that we are working with multi-class classification. A previous study demonstrated that it is more challenging to reach an accuracy of 50% for a multi-class dataset, than to achieve an accuracy of 75% for a binary dataset [89].

Table 3
The per-class effectiveness of the heuristic selection classifier component of TDNN-ALHH-Both.

Class	Exact match ratio (%)	AUC
LLH0	98.25	0.781
LLH1	99.46	0.768
LLH2	91.58	0.864
LLH3	99.18	0.651
LLH4	75.99	0.743
LLH7	77.43	0.956
LLH8	64.32	0.805
LLH9	93.78	0.782

In addition to accuracy, multi-class domains typically also use average-accuracy to assess the quality of the overall classification [88]. This is due to the fact that considering overall accuracy does not separate between the number of correctly classified examples of different classes, especially in the context of imbalanced classification problems. Table 3 shows the details on the exact match ratio for every class (low-level heuristic) available for selection. The rate of correctly classified instances for each class ranges between 64.32% and 99.46%. Based on average accuracy, the classification effectiveness of the heuristic selection classifier is 87.5%. For move acceptance, the trained model has fixed strategies based on the behaviour of the two expert approaches for some of the low-level heuristics: equal moves are always accepted for LLH0, LLH4, LLH8, and LLH9 and worsening moves are always accepted for LLH7. For the remaining low-level heuristics, the accuracies of the move acceptance classifiers for LLH1, LLH2, and LLH3 are 85.2%, 89.1% and 87% respectively.

To further illustrate the predictive ability of classifiers for heuristic selection, we also provide Receiver Operating Characteristics (ROC) analysis with Area Under the Curve (AUC) score in Table 3. AUC is also referred to as Balanced Accuracy [88]. This metric is often used in imbalanced data classification [90], which is suitable for our datasets. In imbalanced domains, it is necessary to combine the individual measures from a confusion matrix to take into account the class distribution. A well-known approach to unify these measures is to use the ROC curve,

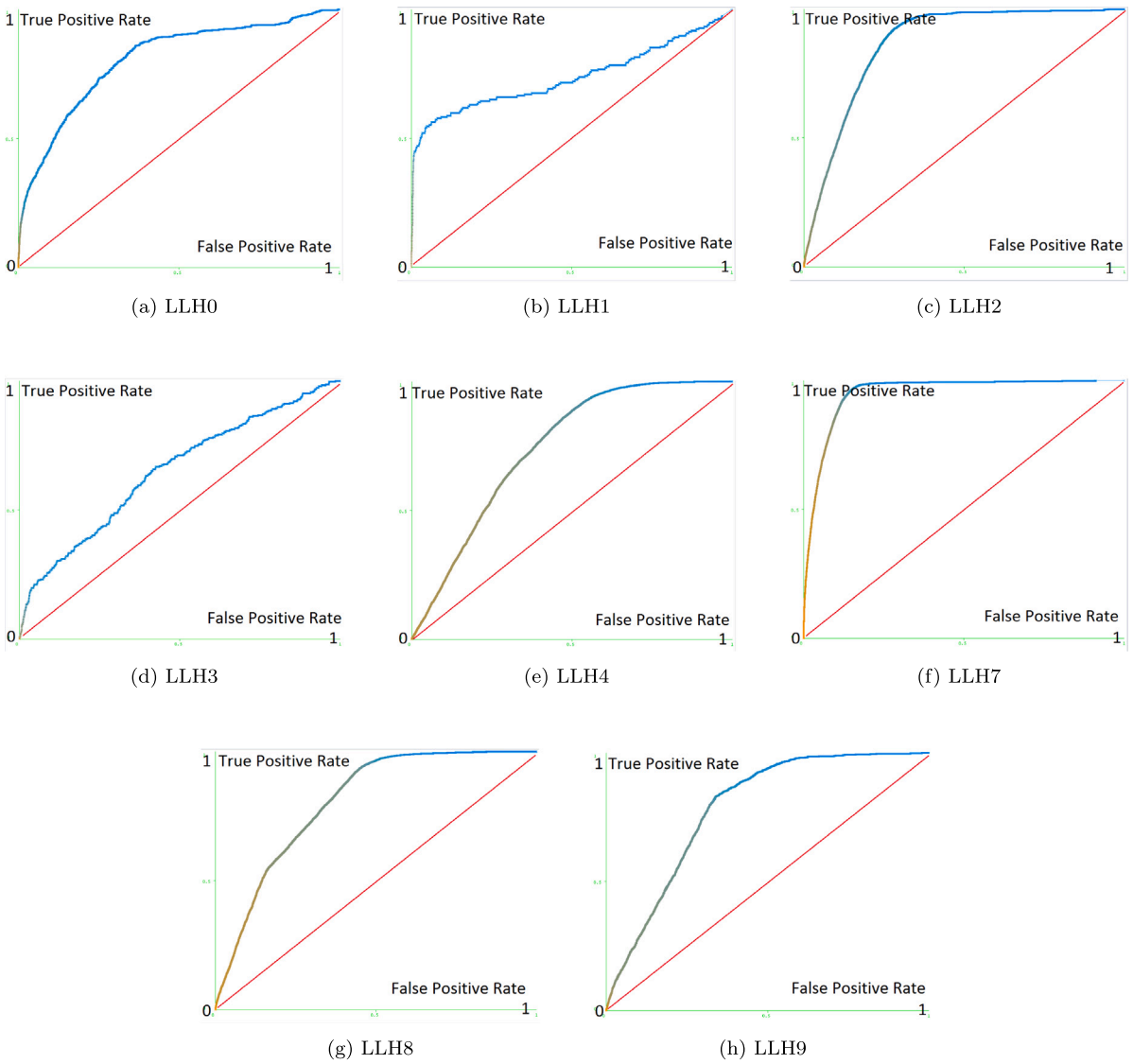


Fig. 6. Receiver Operating Characteristics (ROC) curves for different low-level heuristics, visualising the predictive ability of the heuristic selection classifier component of TDNN-ALHH-Both.

shown for each low-level heuristic in Fig. 6. The ROC curve is created by plotting the True Positive Rate (TPR) on the y-axis against the False Positive Rate (FPR) on the x-axis. The TPR of a classifier is the number of positive instances correctly classified (true positive) compared to the total positive instances (true positive and false negative). While the FPR is estimated as a number of negative instances falsely classified (false positive) compared to the total negative instances (false positive and true negative). A good classifier is reflected by a ROC curve which lies in the upper left hand corner of the square. Therefore, a perfect classifier is indicated by a ROC curve goes straight up the y-axis (following the left-hand side of the square) then along the x-axis (across the top boundary of the square). On the other hand, a classification rule no better than chance is portrayed by a ROC curve which follows the diagonal (from the lower left corner to the top right corner) of the square, as illustrated by a red diagonal line in Fig. 6.

The AUC is simply the area under the ROC curve. For AUC, the higher the score (approaching 1), the better the performance of a classifier, while a score of 0.5 is comparable to random choice. It can be seen from Table 3 that all of the classes have a score of more than 0.5 (between 0.651 and 0.956), demonstrating that the classifiers have learnt some information about the class. Based on the criteria above for identifying a good classifier, the best ROC curve is for LLH7, followed by LLH2 and LLH8, respectively. The following sections will analyse the utilisation and activity of these low-level heuristics in more detail.

5.3. Analysis of low-level heuristic utilisation

This section provides some analysis on the success of individual low-level heuristics during the search process. For this purpose, we trained the hyper-heuristic on a set of arbitrarily chosen instances representing different problem sizes (Standard, Large and Very Large). The instances selected are S3, S7, S11, L1, L5, VL2, VL6 and VL10. We use the same instances for both experts. We test the generated hyper-heuristic on three sets of unseen instances consisting of a single problem size, independent of the training instances chosen above. The behaviour for the Large and Very Large instances is almost identical, so for brevity we present the results from the Standard and Very Large instances. Fig. 7 illustrates the average utilisation rate (ratio) for each low-level heuristic by the apprentice trained using both experts (TDNN-ALHH-Both) and the original expert hyper-heuristics AdapHH and MSHH. The *utilisation rate* is calculated as the number of improvements that a low-level heuristic yields over the best solution found, divided by the total number of improvements. In order to capture the typical behaviour of

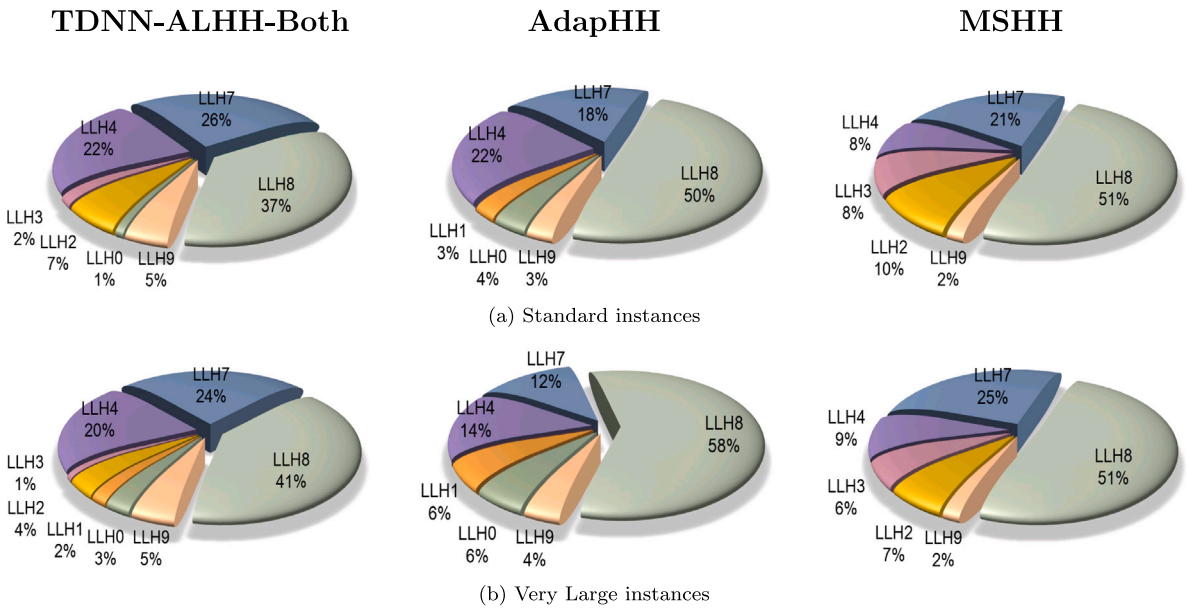


Fig. 7. The mean low-level heuristic utilisation rate averaged over ten trials per problem instance for (left to right) the apprentice (TDNN-ALHH-Both), AdapHH, and MSHH.

the generated hyper-heuristics, each instance is run ten times, with the average value used in the results.

The experimental results indicate that learning appropriate behaviours for different sizes of problem instance has been successful. Comparing the generated hyper-heuristic to the experts it has learned from in Fig. 7, the similarity between these charts shows that the proposed algorithm benefits from using some of the same effective low-level heuristics as the experts (LLH4, LLH7, and LLH8 tend to dominate). The highest ratios tend to be hill-climber operators, followed by mutational operators. From these charts, we note it becomes clear that not all LLHs demonstrate an improvement in solution quality during the search process in all hyper-heuristics. TDNN-ALHH-Both tends to follow the behaviour of MSHH more closely, with improvements yielded from using LLH2 and LLH3, even if this is only a small proportion of the time. AdapHH did not utilise these two low-level heuristics to improve solutions. This behaviour is perhaps to be expected, since MSHH is a state-of-the-art approach, previously demonstrated to outperform AdapHH across six problem domains [25]. Despite this, AdapHH still seems to provide some contribution to the behaviour of the generated hyper-heuristics. TDNN-ALHH-Both follows the behaviour of this expert closely for some LLHs, utilising LLH4 and LLH7 at approximately the same frequency as AdapHH.

From Fig. 7(a), we can see that LLH1 is not present at all for the apprentice (TDNN-ALHH-Both) on the Standard instances. From a machine learning perspective this is not too surprising, MSHH does not utilise LLH1, while AdapHH very rarely utilises this heuristic. Comparing Fig. 7(a) (Standard) and Fig. 7(b) (Very Large), TDNN-ALHH-Both includes LLH1 for Very Large instances. It appears that pattern is recognised from AdapHH, which utilises this low-level heuristic at a higher rate for Very Large instances compared to the Standard instances. This condition also helps TDNN-ALHH-Both to perform well, because it increases the variation in the LLHs used, even though the larger instances are considerably more challenging. From the hill-climber operators, the generated hyper-heuristic and both experts all utilise LLH8 most often. Where the generated hyper-heuristic differs from both experts is the proportion of times LLH8 yields an improvement, and to a lesser extent LLH9. These heuristics are utilised at a noticeably different rate than by both of the expert approaches, although LLH8 is still utilised significantly more often than any other low-level heuristic.

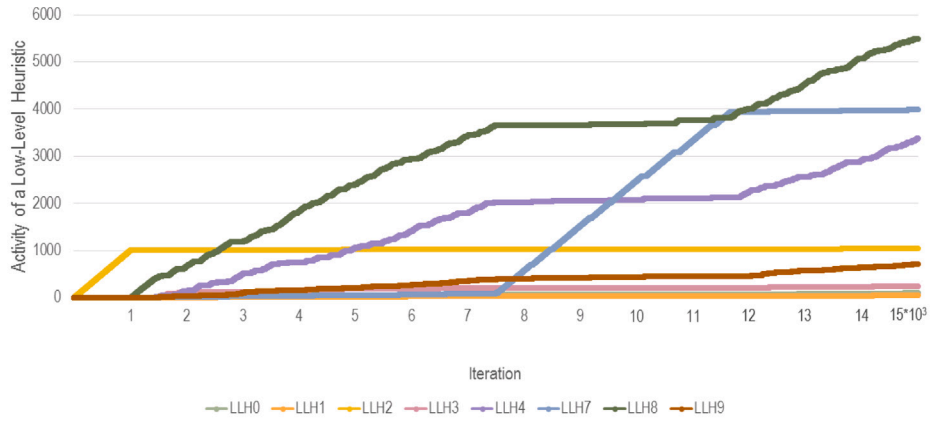
We can see that machine learning is clearly working, finding hidden patterns in these choices as the algorithm operates, based on the merged data from two experts. The ability to capture these hidden patterns, in the form of transitions between different low-level heuristics, leads to a more effective heuristic search method. Without knowing where the data originates, the apprenticeship learning approach is able to identify the best low-level heuristic to apply. The expert methods used here have been previously deployed to solve other variants of VRP. The performance of AdapHH when solving VRP with time windows including a combination of elements from Personnel Rostering have been reported in the literature [17]. While the performance and behaviour of MSHH when solving VRP instances from CHeSC 2011 has been published by Kheiri et al. [25]. Both of these previous works reported similar results regarding heuristic performance, with LLH8 the most favoured low-level heuristic.

Table 4
Percentage utilisation (%) comparison between types of LLH.

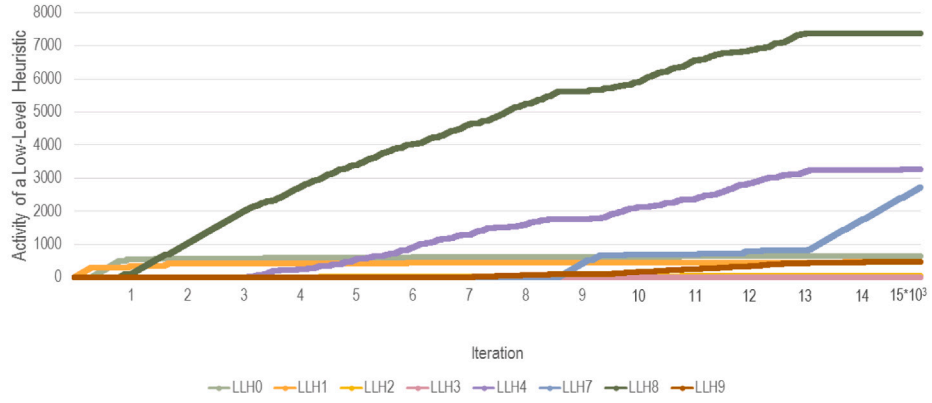
Algorithm	Local search	Mutation	Ruin-recreate
Standard			
TDNN-ALHH-Both	64%	27%	9%
AdapHH	75%	25%	0%
MSHH	61%	21%	18%
Very Large			
TDNN-ALHH-Both	66%	29%	5%
AdapHH	76%	24%	0%
MSHH	62%	25%	13%

Table 4 presents the percentage utilisation between different types of low-level heuristic for the apprentice hyper-heuristic trained using both experts (TDNN-ALHH-Both), and for both of the experts executed independently.

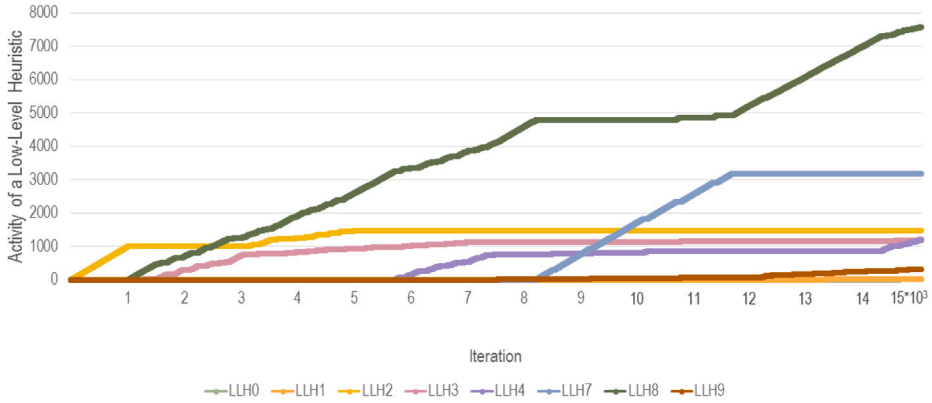
Table 4 shows a consistency between all three approaches, with the majority of improving moves resulting from the application of hill-climber (local search) heuristics, rather than mutational and ruin-recreate. It is perhaps unsurprising that local search operators are responsible for the vast majority of improving moves, as by their very nature they are guaranteed to produce a solution that is at least as good as the current solution. Interestingly, the mutation operators yield many more improving solutions than the ruin-recreate operators. This is perhaps unexpected, since the ruin-recreate operators incorporate a significant amount of domain knowledge, and could be considered as 'intelligent' heuristics.



(a) TDNN-ALHH-Both



(b) TDNN-ALHH-AdapHH



(c) TDNN-ALHH-MSHH

Fig. 8. A sample activity plot of low level heuristics from generated selection hyper-heuristics over time (iterations in thousands) while solving instance S11, where the TDNN was trained using the data from (a) both experts, (b) AdapHH and (c) MSHH.

5.4. An analysis of low-level heuristic selection

Fig. 8 provides low-level heuristic *selection activity* plots for the generated hyper-heuristics trained using different expert datasets, to illustrate the usage of different low-level heuristics over time in multiple runs. This plot is inspired by evolutionary activity plots [91] of memes in memetic algorithms, which count the occurrence of each meme considering all solutions in a given iteration (generation) of a run. In our case, selection hyper-heuristics are based on single-point based search, so we can use a similar count: the number of times a low-level heuristic is selected at the given iteration considering multiple runs. This generates monotonically increasing functions with respect to the number of iterations. The gradient of the plot indicates whether or not low-level heuristic is favoured consistently across multiple runs. The more frequently a low-level heuristic is selected and invoked, the steeper the slope is. The aim is to identify which low-level heuristics are favoured at different stages of the search process. This analysis can provide deeper insight into the behaviour of the generated hyper-heuristics.

The low-level heuristic activity plots in Fig. 8 record the frequency of selection of each low-level heuristic up until that point for every iteration on Standard instance S11, chosen arbitrarily for this analysis. Unlike Fig. 7, which identifies the overall percentage of global best solutions generated by a low-level heuristic, the activity plots show how often a particular low-level heuristic is selected at every decision point over time. We limit each run to 15,000 iterations, the scale of the x-axis values of the plots are in thousands.

It can be seen for TDNN-ALHH-Both that early in the search, LLH2 is frequently selected, then after some time it begins to choose LLH4, LLH8, and LLH9, followed by LLH3 and LLH7. A strategy of applying ruin-recreate in the early stages, followed by hill-climbers later on seems to be preferred. LLH7 tends to contribute more in the middle of the search process, diversifying the search, before hill-climber heuristics (LLH8, LLH4 and LLH9) are applied towards the end of the search process. This is supported by the highest AUC score (showing effective classifier performance) of LLH7 (followed by LLH8 as the third best) previously shown in Table 3 and Fig. 6. Looking at Fig. 8, it becomes clear that TDNN-ALHH-Both has its own policy that differs slightly from the expert demonstrators. Nevertheless, it seems to incorporate patterns of behaviour from both experts. TDNN-ALHH-Both mimics the behaviour of MSHH with its use of LLH8 and LLH7, and shows similarity to AdapHH by selecting LLH4 in the earlier stages of the search. It is worth mentioning that the activity plots of the low-level heuristics in Fig. 8 are aligned with Fig. 7 in terms of the low-level heuristics that are not used. The TDNN trained using both experts did not use LLH1, similar to MSHH which did not use LLH1 and LLH0, while AdapHH did not use LLH2 and LLH3.

6. Conclusion

This work examines the interplay between data science and optimisation, combining machine learning and hyper-heuristics. In this study, a generation hyper-heuristic is introduced, which takes advantage of a data science technique: apprenticeship learning using Time Delay Neural Networks (TDNN). TDNNs were deployed as a machine learning algorithm to generate components of a selection hyper-heuristic, learning from the demonstrations of two state-of-the-art 'expert' hyper-heuristics solving selected instances. We have presented an approach that exhibits learning from multiple experts, using the Open Vehicle Routing Problem (OVRP) as a case study, solving problem instances of various sizes. The proposed approach is used to generate the primary components of a hyper-heuristic: heuristic selection and move acceptance. The experimental results on a benchmark of OVRP instances show that observing and learning from multiple experts as they solve the training instances is a better strategy than using a single expert. In addition, improved performance when compared to the performance of each individual expert applied independently is demonstrated. The generated selection hyper-heuristic scales and generalises well, with improved performance observed even as the size of the OVRP instances increase. Unlike most previous hyper-heuristic studies, the higher level machine learning approach we use to generate selection hyper-heuristics has additional information available to it, in the form of the distance between solutions. Combining this information from the search behaviour of two expert demonstrators was able to yield improved performance over training using the information from a single demonstrator, and over directly applying either of the experts individually. The experimental analysis highlights the relationship between classification performance and the behaviour of the generated hyper-heuristic. The results indicate that improving the classification performance can in return enhance the heuristic optimisation process. Future work will consider combinations of larger sets of expert systems from which behaviour is learned, applied to a wider range of problem domains and instances. In particular, we will focus on generalisation from a cross-domain perspective, investigating the differences in performance based on different combinations of intra- and inter-domain learning [92].

CRediT authorship contribution statement

Raras Tyasnurita: Writing – original draft, Visualization, Validation, Software, Project administration, Investigation, Formal analysis, Data curation, Conceptualization. **Ender Özcan:** Writing – review & editing, Writing – original draft, Validation, Supervision, Project administration, Methodology, Investigation, Funding acquisition, Conceptualization. **John H. Drake:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Methodology, Investigation. **Shahriar Asta:** Writing – original draft, Visualization, Validation, Supervision, Software, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

[1] K. Braekers, K. Ramaekers, I. Van Nieuwenhuyse, The vehicle routing problem: State of the art classification and review, Comput. Ind. Eng. 99 (2016) 300–313.

[2] M. Hosseininia, F. Dadgostari, Hamiltonian paths and cycles, in: Graph Theory for Operations Research and Management: Applications in Industrial Engineering: Applications in Industrial Engineering, Vol. 96, IGI Global, 2012.

[3] F. Li, B. Golden, E. Wasil, The open vehicle routing problem: Algorithms, large-scale test problems, and computational results, Comput. Oper. Res. 34 (10) (2007) 2918–2930.

[4] A.N. Letchford, J. Lysgaard, R.W. Eglese, A branch-and-cut algorithm for the capacitated open vehicle routing problem, J. Oper. Res. Soc. 58 (12) (2007) 1642–1651.

[5] D. Pecin, A. Pessoa, M. Poggi, E. Uchoa, Improved branch-cut-and-price for capacitated vehicle routing, Math. Program. Comput. 9 (1) (2017) 61–100.

[6] G. Reinelt, The Traveling Salesman: Computational Solutions for TSP Applications, Springer-Verlag, 1994.

[7] K. Fleszar, I.H. Osman, K.S. Hindi, A variable neighbourhood search algorithm for the open vehicle routing problem, European J. Oper. Res. 195 (3) (2009) 803–809.

[8] P.P. Repoussis, C.D. Tarantilis, O. Bräysy, G. Ioannou, A hybrid evolution strategy for the open vehicle routing problem, Comput. Oper. Res. 37 (3) (2010) 443–455.

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

[10] J.H. Drake, A. Kheiri, E. Özcan, E.K. Burke, Recent advances in selection hyper-heuristics, European J. Oper. Res. 285 (2) (2020) 405–428.

[11] H. Song, I. Triguero, E. Özcan, A review on the self and dual interactions between machine learning and optimisation, Prog. Artif. Intell. 8 (2) (2019) 143–165.

[12] E.K. Burke, M.R. Hyde, G. Kendall, G. Ochoa, E. Ozcan, J.R. Woodward, Exploring hyper-heuristic methodologies with genetic programming, in: Computational Intelligence, Springer, 2009, pp. 177–201.

[13] R. Tyasnurita, E. Özcan, R. John, Learning heuristic selection using a time delay neural network for open vehicle routing, in: Evolutionary Computation (CEC), 2017 IEEE Congress on, IEEE, 2017, pp. 1474–1481.

[14] P. Abbeel, A.Y. Ng, Apprenticeship learning via inverse reinforcement learning, in: Proceedings of the Twenty-First International Conference on Machine Learning, ACM, 2004, p. 1.

[15] P. Cowling, G. Kendall, E. Soubeiga, A hyperheuristic approach to scheduling a sales summit, in: Practice and Theory of Automated Timetabling III, in: Lecture Notes in Computer Science, vol. 2079, Springer Berlin Heidelberg, 2001, pp. 176–190, http://dx.doi.org/10.1007/3-540-44629-X_11.

[16] E. Özcan, M. Misir, G. Ochoa, E.K. Burke, A reinforcement learning: great-deluge hyper-heuristic for examination timetabling, in: Modeling, Analysis, and Applications in Metaheuristic Computing: Advancements and Trends, IGI Global, 2012, pp. 34–55.

[17] M. Mısır, P. Smet, G.V. Berghe, An analysis of generalised heuristics for vehicle routing and personnel rostering problems, J. Oper. Res. Soc. 66 (5) (2015) 858–870.

[18] S. Asta, E. Özcan, T. Curtois, A tensor based hyper-heuristic for nurse rostering, Knowl.-Based Syst. 98 (2016) 185–199.

[19] J.H. Drake, E. Özcan, E.K. Burke, Modified choice function heuristic selection for the multidimensional knapsack problem, in: Genetic and Evolutionary Computing, Springer, 2015, pp. 225–234.

[20] J.H. Drake, E. Özcan, E.K. Burke, A case study of controlling crossover in a selection hyper-heuristic framework using the multidimensional knapsack problem, Evol. Comput. 24 (1) (2016) 113–141.

[21] G. Guizzo, M. Bazargani, M. Paixao, J.H. Drake, A hyper-heuristic for multi-objective integration and test ordering in google guava, in: International Symposium on Search Based Software Engineering, Springer, 2017, pp. 168–174.

[22] S.M. Pour, J.H. Drake, E.K. Burke, A choice function hyper-heuristic framework for the allocation of maintenance tasks in Danish railways, Comput. Oper. Res. 93 (2018) 15–26.
[23] S. Mahmud, A. Abbasi, R.K. Chakrabortty, M.J. Ryan, A self-adaptive hyper-heuristic based multi-objective optimisation approach for integrated supply chain scheduling problems, Knowl.-Based Syst. 251 (2022) 109190.
[24] M. Mısır, K. Verbeeck, P. De Causmaecker, G.V. Berghe, An intelligent hyper-heuristic framework for chesc 2011, in: Learning and Intelligent Optimization, Springer, 2012, pp. 461–466.
[25] A. Kheiri, E. Özcan, An iterated multi-stage selection hyper-heuristic, European J. Oper. Res. 250 (1) (2016) 77–90.
[26] E.K. Burke, M. Hyde, G. Kendall, J. Woodward, A genetic programming hyper-heuristic approach for evolving 2-D strip packing heuristics, IEEE Trans. Evol. Comput. 14 (6) (2010) 942–958.
[27] L. Zhu, J. Lin, Y.-Y. Li, Z.-J. Wang, A decomposition-based multi-objective genetic programming hyper-heuristic approach for the multi-skill resource constrained project scheduling problem, Knowl.-Based Syst. 225 (2021) 107099.
[28] J.H. Drake, M. Hyde, K. Ibrahim, E. Ozcan, A genetic programming hyper-heuristic for the multidimensional knapsack problem, Kybernetes 43 (9/10) (2014) 1500–1511.
[29] E.K. Burke, M.R. Hyde, G. Kendall, Grammatical evolution of local search heuristics, IEEE Trans. Evol. Comput. 16 (3) (2012) 406–417.
[30] J.H. Drake, N. Kililis, E. Özcan, Generation of VNS components with grammatical evolution for vehicle routing, in: European Conference on Genetic Programming, Springer, 2013, pp. 25–36.
[31] G. Mweshi, N. Pillay, An improved grammatical evolution approach for generating perturbative heuristics to solve combinatorial optimization problems, Expert Syst. Appl. 165 (2021) 113853.
[32] J. Swan, P. De Causmaecker, S. Martin, E. Özcan, A re-characterization of hyper-heuristics, in: L. Amodeo, E.-G. Talbi, F. Yalaoui (Eds.), Recent Developments in Metaheuristics, Springer International Publishing, Cham, 2018, pp. 75–89.
[33] E. Özcan, A.J. Parkes, Policy matrix evolution for generation of heuristics, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, ACM, 2011.
[34] S. Asta, E. Ozcan, A tensor analysis improved genetic algorithm for online bin packing, in: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, ACM, 2015, pp. 799–806.
[35] F. Thabtah, P. Cowling, Mining the data from a hyperheuristic approach using associative classification, Expert Syst. Appl. 34 (2) (2008) 1093–1101.
[36] J. Li, E.K. Burke, R. Qu, Integrating neural networks and logistic regression to underpin hyper-heuristic search, Knowl.-Based Syst. 24 (2) (2011) 322–330.
[37] J.M. Tapia-Avitia, J.M. Cruz-Duarte, I. Amaya, J.C. Ortiz-Bayliss, H. Terashima-Marin, N. Pillay, A primary study on hyper-heuristics powered by artificial neural networks for customising population-based metaheuristics in continuous optimisation problems, in: 2022 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2022, pp. 1–8.
[38] S. Asta, E. Özcan, A tensor-based selection hyper-heuristic for cross-domain heuristic search, Inform. Sci. 299 (2015) 412–432.
[39] E.K. Burke, B.L. MacCarthy, S. Petrovic, R. Qu, Knowledge discovery in a hyper-heuristic for course timetabling using case-based reasoning, in: International Conference on the Practice and Theory of Automated Timetabling, Springer, 2002, pp. 276–287.
[40] W. Li, E. Özcan, R. John, A learning automata based multiobjective hyper-heuristic, IEEE Trans. Evol. Comput. (2017).
[41] J. Lin, Y.-Y. Li, H.-B. Song, Semiconductor final testing scheduling using Q-learning based hyper-heuristic, Expert Syst. Appl. 187 (2022) 115978.
[42] Z.-Q. Zhang, F.-C. Wu, B. Qian, R. Hu, L. Wang, H.-P. Jin, A Q-learning-based hyper-heuristic evolutionary algorithm for the distributed flexible job-shop scheduling problem with crane transportation, Expert Syst. Appl. 234 (2023) 121050.
[43] S. Asta, E. Özcan, A.J. Parkes, A.Ş. Etaner-Uyar, Generalizing hyper-heuristics via apprenticeship learning, in: European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, 2013, pp. 169–178.
[44] S. Asta, E. Özcan, An apprenticeship learning hyper-heuristic for vehicle routing in HyFlex, in: Evolving and Autonomous Learning Systems (EALS), 2014 IEEE Symposium on, IEEE, 2014, pp. 65–72.
[45] R. Tyasnurita, E. Ozcan, S. Asta, R. John, Improving performance of a hyper-heuristic using a multilayer perceptron for vehicle routing, in: The 15th UK Workshop on Computational Intelligence, 2015.
[46] K.A. Smith-Miles, Towards insightful algorithm selection for optimisation using meta-learning concepts, in: Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on, IEEE, 2008, pp. 4118–4124.
[47] M.N. Le, Y.S. Ong, A frequent pattern mining algorithm for understanding genetic algorithms, in: Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence: 4th International Conference on Intelligent Computing, ICIC 2008 Shanghai, China, September 15-18, 2008 Proceedings 4, Springer, 2008, pp. 131–139.
[48] J.Y. Chia, C.K. Goh, K.C. Tan, V.A. Shim, Memetic informed evolutionary optimization via data mining, Memet. Comput. 3 (2011) 73–87.
[49] Y. Zhou, J.-K. Hao, B. Duval, Frequent pattern-based search: A case study on the quadratic assignment problem, IEEE Trans. Syst. Man Cybern.: Syst. 52 (3) (2020) 1503–1515.
[50] Y. Zhou, X. Zhang, N. Geng, Z. Jiang, S. Wang, M. Zhou, Frequent itemset-driven search for finding minimal node separators and its application to air transportation network analysis, IEEE Trans. Intell. Transp. Syst. 24 (8) (2023) 8348–8360.
[51] K.J. Lang, G.E. Hinton, A Time-Delay Neural Network Architecture for Speech Recognition, Carnegie Mellon University, Computer Science Department, 1988.
[52] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, K.J. Lang, Phoneme recognition using time-delay neural networks, IEEE Trans. Acoust. Speech Signal Process. 37 (3) (1989) 328–339.
[53] S. Haykin, Neural Networks: A Comprehensive Foundation, Macmillan College Publishing Company, New York, 1994.
[54] Y. Yamashita, Time delay neural networks for the classification of flow regimes, Comput. Chem. Eng. 21 (1997) S367–S371.
[55] E.A. Wan, Temporal backpropagation for FIR neural networks, in: Neural Networks, 1990., 1990 IJCNN International Joint Conference on, IEEE, 1990, pp. 575–580.
[56] B. Abdulhai, H. Porwal, W. Recker, Short Term Freeway Traffic Flow Prediction Using Genetically-Optimized Time-Delay-Based Neural Networks, California PATH (Partners for Advanced Transportation Technology) working paper, 1999.
[57] C. Wohler, J.K. Anlauf, An adaptable time-delay neural-network algorithm for image sequence analysis, IEEE Trans. Neural Netw. 10 (6) (1999) 1531–1536.
[58] D. Shi, H. Zhang, L. Yang, Time-delay neural network for the prediction of carbonation tower's temperature, IEEE Trans. Instrum. Meas. 52 (4) (2003) 1125–1128.
[59] J. Zhu, J. Zurcher, M. Rao, M.Q. Meng, An on-line wastewater quality predication system based on a time-delay neural network, Eng. Appl. Artif. Intell. 11 (6) (1998) 747–758.
[60] E.W. Saad, D.V. Prokhorov, D.C. Wunsch, Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks, IEEE Trans. Neural Netw. 9 (6) (1998) 1456–1470.
[61] D.S. Clouse, C.L. Giles, B.G. Horne, G.W. Cottrell, Time-delay neural networks: Representation and induction of finite-state machines, IEEE Trans. Neural Netw. 8 (5) (1997) 1065–1070.
[62] V. Peddinti, D. Povey, S. Khudanpur, A time delay neural network architecture for efficient modeling of long temporal contexts, in: Sixteenth Annual Conference of the International Speech Communication Association, 2015.
[63] C. Wöhler, J.K. Anlauf, A time delay neural network algorithm for estimating image-pattern shape and motion, Image Vis. Comput. 17 (3) (1999) 281–294.
[64] C. Liu, G. Kou, X. Zhou, Y. Peng, H. Sheng, F.E. Alsaadi, Time-dependent vehicle routing problem with time windows of city logistics with a congestion avoidance approach, Knowl.-Based Syst. 188 (2020) 104813.
[65] Y. Niu, D. Kong, R. Wen, Z. Cao, J. Xiao, An improved learnable evolution model for solving multi-objective vehicle routing problem with stochastic demand, Knowl.-Based Syst. 230 (2021) 107378.
[66] Y. Wang, Q. Li, X. Guan, J. Fan, M. Xu, H. Wang, Collaborative multi-depot pickup and delivery vehicle routing problem with split loads and time windows, Knowl.-Based Syst. 231 (2021) 107412.
[67] M. Song, L. Cheng, An augmented Lagrangian relaxation method for the mean-standard deviation based vehicle routing problem, Knowl.-Based Syst. 247 (2022) 108736.
[68] G.B. Dantzig, J.H. Ramser, The truck dispatching problem, Manage. Sci. 6 (1) (1959) 80–91.
[69] B. Eksioglu, A.V. Vural, A. Reisman, The vehicle routing problem: A taxonomic review, Comput. Ind. Eng. 57 (4) (2009) 1472–1483.
[70] L. Schrage, Formulation and structure of more complex/realistic routing and scheduling problems, Networks 11 (2) (1981) 229–232.
[71] L. Bodin, Routing and scheduling of vehicles and crews, the state of the art, Comput. Oper. Res. 10 (2) (1983) 63–211.
[72] D. Sariklis, S. Powell, A heuristic method for the open vehicle routing problem, J. Oper. Res. Soc. 51 (5) (2000) 564–573.
[73] J. Brandão, A tabu search algorithm for the open vehicle routing problem, European J. Oper. Res. 157 (3) (2004) 552–564.
[74] A. Subramanian, E. Uchoa, L.S. Ochi, A hybrid algorithm for a class of vehicle routing problems, Comput. Oper. Res. 40 (10) (2013) 2519–2531.
[75] A.A.R. Hosseinabadi, J. Vahidi, V.E. Balas, S.S. Mirkamali, OVRP_GELS: solving open vehicle routing problem using the gravitational emulation local search algorithm, Neural Comput. Appl. 29 (10) (2018) 955–968.
[76] E. Ozcetin, G. Ozturk, Z.K. Ozturk, R. Kasimbeyli, N. Kasimbeyli, A decision support system for consolidated distribution of a ceramic sanitary ware company, Expert Syst. Appl. 213 (2023) 118785.
[77] M. Gendreau, C.D. Tarantilis, Solving Large-Scale Vehicle Routing Problems with Time Windows: The State-Of-The-Art, Cirrelt Montreal, 2010.
[78] F. Li, B. Golden, E. Wasil, Very large-scale vehicle routing: new test problems, algorithms, and results, Comput. Oper. Res. 32 (5) (2005) 1165–1179.
[79] B.D. Argall, S. Chernova, M. Veloso, B. Browning, A survey of robot learning from demonstration, Robot. Auton. Syst. 57 (5) (2009) 469–483.

[80] C. Sammut, Automatically constructing control systems by observing human behaviour, in: Proc. of the Internat. Workshop on Inductive Logic Programming, 1992.
[81] R. Amit, M. Matari, Learning movement sequences from demonstration, in: Development and Learning, 2002. Proceedings. The 2nd International Conference on, IEEE, 2002, pp. 203–208.
[82] M. Kubiak, Distance measures and fitness-distance analysis for the capacitated vehicle routing problem, in: Metaheuristics, Springer, 2007, pp. 345–364.
[83] G. Ochoa, M. Hyde, T. Curtois, J.A. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A.J. Parkes, S. Petrovic, et al., Hyflex: A benchmark framework for cross-domain heuristic search, in: European Conference on Evolutionary Computation in Combinatorial Optimization, Springer, 2012, pp. 136–147.
[84] J.D. Walker, Design of Vehicle Routing Problem Domains for a Hyper-Heuristic Framework (Ph.D. thesis), University of Nottingham, 2015.
[85] D.B. Gümüş, E. Ozcan, J. Atkin, An investigation of tuning a memetic algorithm for cross-domain search, in: Evolutionary Computation (CEC), 2016 IEEE Congress on, IEEE, 2016, pp. 135–142.
[86] S.B. Kotsiantis, I. Zaharakis, P. Pintelas, Supervised machine learning: A review of classification techniques, 2007.
[87] S. Asta, E. Özcan, A.J. Parkes, Batched mode hyper-heuristics, in: International Conference on Learning and Intelligent Optimization, Springer, 2013, pp. 404–409.
[88] M. Sokolova, G. Lapalme, A systematic analysis of performance measures for classification tasks, Inf. Process. Manage. 45 (4) (2009) 427–437.
[89] A. Statnikov, C.F. Aliferis, I. Tsamardinos, D. Hardin, S. Levy, A comprehensive evaluation of multicategory classification methods for microarray gene expression cancer diagnosis, Bioinformatics 21 (5) (2004) 631–643.
[90] V. López, A. Fernández, S. García, V. Palade, F. Herrera, An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics, Inform. Sci. 250 (2013) 113–141.
[91] M.A. Bedau, C.T. Brown, Visualizing evolutionary activity of genotypes, Artif. Life 5 (1) (1999) 17–35.
[92] M. Mısır, Matrix factorization based benchmark set analysis: A case study on hyflex, in: Asia-Pacific Conference on Simulated Evolution and Learning, Springer, 2017, pp. 184–195.