Offline Learning for Selection Hyper-heuristics with Elman Networks

W. B. Yates and E. C. Keedwell

Computer Science, College of Engineering, Mathematics and Physical Sciences,
University of Exeter, Exeter, EX4 4QF, UK.
wy254@exeter.ac.uk E.C.Keedwell@exeter.ac.uk

Abstract. Offline selection hyper-heuristics are machine learning methods that are trained on heuristic selections to create an algorithm that is tuned for a particular problem domain. In this work, a simple selection hyper-heuristic is executed on a number of computationally hard benchmark optimisation problems, and the resulting sequences of low level heuristic selections and objective function values are used to construct an offline learning database. An Elman network is trained on sequences of heuristic selections chosen from the offline database and the network's ability to learn and generalise from these sequences is evaluated. The networks are trained using a leave-one-out cross validation methodology and the sequences of heuristic selections they produce are tested on benchmark problems drawn from the HyFlex set. The results demonstrate that the Elman network is capable of intra-domain learning and generalisation with 99% confidence and produces better results than the training sequences in many cases. When the network was trained using an inter-domain training set, the Elman network did not exhibit generalisation indicating that inter-domain generalisation is a harder problem and that strategies learned on one domain cannot necessarily be transferred to another.

Keywords: Hyper-heuristics, Elman networks, Offline learning.

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 [1]).

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 [2], [3], and [4]). In [2] *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 [3] *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 [4], *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 [5] and [6]) 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 [7]) 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 [8]) 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 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 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 4 presents the conclusions of this study.

2 Methodology

Section 2.1 contains a description of the HyFlex benchmark problems and the DBGen hyper-heuristic used to generate the offline learning database. In Section 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 2.3 details the architecture of the Elman network used in this study, while Section 2.4 describes the construction of the intra-domain and inter-domain training sets. Finally, in Section 2.5, the BLIND hyper-heuristic that is used to evaluate the sequences produced by the trained Elman networks is presented.

2.1 HyFlex and the Offline Learning Database

The Hyper-heuristics Flexible framework (or HyFlex[1], see [7]) is a set of benchmark problems that has been used in a number of studies. See for example [9], [10], [11], [12], [5], and [13]. 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

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

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
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 1 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 if
17.    if (new-obj ≥ cur-obj and r ≥ 0.5) then
18.        new-sol ← cur-sol;
19.        new-obj ← cur-obj;
20.    end if
21. end while
```

The random, unbiased, single selection hyper-heuristic DBGen used to generate the offline learning database is shown in listing 1. 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.

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.

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 [8] and [14]). 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

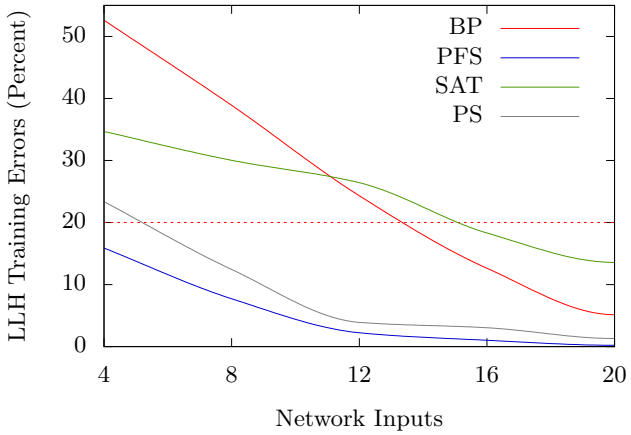
$$(\overbrace{0,0,1,0}^{\mathtt{R}}).$$

The number of selections to be used as an input is termed the memory length of a selection strategy (see [15]). 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 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 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.

Fig. 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.

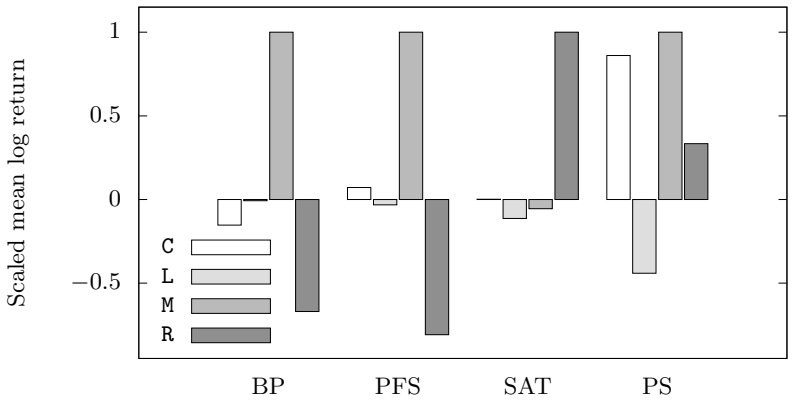


After training, given some initial input, an Elman network produces a sequence of outputs. The output sequence may converge to a single point, a limit cycle of repeating values, or produce a chaotic non-repeating sequence.

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 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.

Fig. 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 [16]) 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.

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.

3 Results

Section 3.1 presents the results of training the Elman networks with the intra-domain and inter-domain training sequences. In Section 3.2 the sequences that are generated by the trained networks are evaluated on the HyFlex problems using the BLIND hyper-heuristic.

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 [16]) using the parameters shown in table 1. The learning rate, momentum term, and the number of training epochs have not been optimised.

The results of network training are summarised in table 2 and figure 3. Table 2 shows the results of training the Elman network with the 40 intra-domain

Table 1: The Elman network structure and training parameters.

Input	Hidden	Out	Learn	Momentum	Epochs
16	16	4	0.1	0.25	1000

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 3a shows the percentage of low level heuristic errors dur-

Table 2: The averaged training results of the Elman network on the intra-domain training sets.

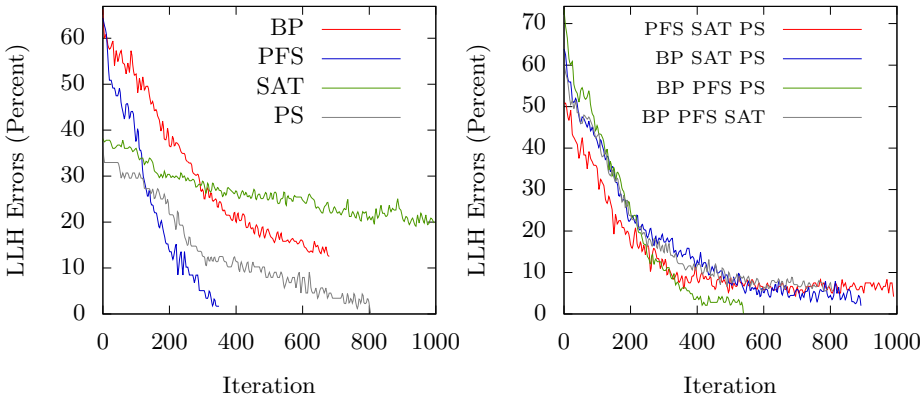
Dom.	Num.	Wrong (%)	Error	Epochs
BP	369.0	12.6407	4.2958	907.7
PFS	94.5	1.0260	1.0491	328.9
SAT	288.2	18.3158	4.1991	918.9
PS	121.5	3.0474	0.9290	947.3

ing intra-domain training for 4 representative problems (number 7, 19, 34, and 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 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

Fig. 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.



(a) Intra-domain training results.

(b) Inter-domain training results.

Similarly, table 3 and figure 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.

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 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 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.

Table 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

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

$$\overline{\alpha}_f(\text{BEST}) < \overline{\alpha}_f(\text{INTRA})$$

is accepted with 99% confidence. The results of the t-test are shown in table 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.

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

Table 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	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]

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.

References

[1] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. Woodward, *A Classification of Hyper-heuristic Approaches*. Springer US, 2010.

[2] P. Ross, S. Schulenburg, J. G. Marín-Bläzquez, and E. Hart, "Hyper-heuristics: Learning to combine simple heuristics in bin-packing problems," in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, GECCO

2002, (San Francisco, CA, USA), pp. 942–948, Morgan Kaufmann Publishers Inc., 2002.

[3] E. K. Burke, S. Petrovic, and R. Qu, "Case-based heuristic selection for timetabling problems," *Journal of Scheduling*, vol. 9, no. 2, pp. 115–132, 2006.

[4] H. Terashima-Marín, J. C. Ortiz-Bayliss, P. Ross, and M. Valenzuela-Rendón, "Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems," in *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO 2008, (New York, NY, USA), pp. 571–578, ACM, 2008.

[5] A. Kheiri and E. Keedwell, "A sequence-based selection hyper-heuristic utilising a hidden Markov model," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO 2015, pp. 417–424, ACM, 2015.

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

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

[8] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[9] J. D. Walker, G. Ochoa, M. Gendreau, and E. K. Burke, "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*, pp. 265–276, 2012.

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

[11] M. Mısır, K. Verbeeck, P. D. Causmaecker, and G. V. Berghe, "A new hyper-heuristic as a general problem solver: an implementation in HyFlex," *Journal of Scheduling*, vol. 16, no. 3, pp. 291–311, 2013.

[12] J. H. Drake, E. Özcan, and E. K. Burke, "A comparison of crossover control mechanisms within single-point selection hyper-heuristics using HyFlex," in *IEEE Congress on Evolutionary Computation (CEC)*, (Sendai, Japan), pp. 3397–3403, May 2015.

[13] P. Dempster and J. H. Drake, "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)* (H. J. Kim and W. Z. Geem, eds.), (Berlin, Heidelberg), pp. 83–94, Springer Berlin Heidelberg, 2016.

[14] J. L. Elman, "Distributed representations, simple recurrent networks, and grammatical structure," *Machine Learning*, vol. 7, pp. 195–224, 1991.

[15] R. Bai, E. K. Burke, M. Gendreau, G. Kendall, and B. McCollum, "Memory length in hyper-heuristics: An empirical study," in *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling*, pp. 173–178, 2007.

[16] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.