Algorithm Selection Using Deep Learning Without Feature Extraction

Mohamad Alissa
Department of Computer Science,
Edinburgh Napier University
M.Alissa@napier.ac.uk

Kevin Sim
Department of Computer Science,
Edinburgh Napier University
K.Sim@napier.ac.uk

Emma Hart
Department of Computer Science,
Edinburgh Napier University
E.Hart@napier.ac.uk

ABSTRACT

We propose a novel technique for algorithm-selection which adopts a deep-learning approach, specifically a Recurrent-Neural Network with Long-Short-Term-Memory (RNN-LSTM). In contrast to the majority of work in algorithm-selection, the approach does *not need any features* to be extracted from the data but instead relies on the temporal data sequence as input. A large case-study in the domain of 1-d bin packing is undertaken in which instances can be solved by one of four heuristics. We first evolve a large set of new problem instances that each have a clear "best solver" in terms of the heuristics considered. An RNN-LSTM is trained directly using the sequence data describing each instance to predict the best-performing heuristic. Experiments conducted on small and large problem instances with item sizes generated from two different probability distributions are shown to achieve between 7% to 11% improvement over the single best solver (SBS) (i.e. the single heuristic that achieves the best performance over the instance set) and 0% to 2% lower than the virtual best solver (VBS), i.e the perfect mapping.

CCS CONCEPTS

• **Theory of computation → Packing and covering problems**;
• **Computing methodologies → Machine learning**;

KEYWORDS

Deep Learning, Recurrent Neural Network, Algorithm Selection.

ACM Reference Format:
Mohamad Alissa, Kevin Sim, and Emma Hart. 2019. Algorithm Selection Using Deep Learning Without Feature Extraction. In *Genetic and Evolutionary Computation Conference (GECCO '19), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3321707.3321845

1 INTRODUCTION

Exploiting the complementary performance of different algorithms on sets of diverse problem instances drives the goal of algorithm selection. Determining the best performing algorithm for an unseen instance has been shown to be a complex problem that has attracted much interest from researchers over the decades. Originally formulated by Rice [22] the per-instance algorithm selection problem (ASP) can be defined as:

"Given a set I of instances of a problem P, a set $A = A_1, ...A_n$ of algorithms for P and a metric $m : A \times I \rightarrow R$ that measures the performance of any algorithm $A_j \in A$ on instance set I, construct a selector S that maps any problem instance $i \in I$ to an algorithm $S(i) \in A$ such that the overall performance of S on I is optimal according to metric m."[12]

A common approach to tackling the ASP is to treat it as a classification problem where each instance is described in terms of a vector of features and an instance's class indicates the best performing algorithm. Although there have been a number of successful studies using this method, the task of identifying features that correlate to algorithm performance is far from trivial, is time consuming and not always intuitive.

In this paper we investigate the ASP without the need to define features. Our novel approach relies solely on temporal patterns explicitly encoded in the instances used. We train a Long Short Term Memory Deep Learning model [8] to predict which of 4 simple approximation algorithms will perform best on a subset of unseen instances in the 1D-Bin Packing domain (BPP). Input to the model is simply the ordered list of item-sizes that need to be packed, and does not include any features derived *a-priori* from the data sets.

We test our approach on 4 large data sets, totalling 16,000 instances, specifically evolved using an evolutionary algorithm for the purpose of this study. Instances are evolved that elicit diverse behaviour (in terms of performance) from four simple approximation algorithms, resulting in a set of instances that are disparate in the performance space. Experimental results using our model are extremely encouraging, achieving an accuracy of between 82.38% and 95.75% on unseen instances. It should be noted that although conducted on the 1D-BPP our method can easily be extended to any discrete combinatorial problem where the sequence of items presented to the solver is fixed. Examples include packing, scheduling and routing in industrial settings such as resource allocation.

The main contributions are summarised as follows:

- The creation of new, large data sets of instances for the 1-D BPP domain via an evolutionary approach in which each instance is solved best by *exactly one* of four heuristic; these data sets are available for other researchers working in the field of ASP to compare approaches[1]
- The development of a novel ASP approach using deep learning that is trained using only sequences of instance data and does not require any features to be extracted from that data

[1] https://github.com/Kevin-Sim/BPP

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6111-8/19/07…$15.00
https://doi.org/10.1145/3321707.3321845

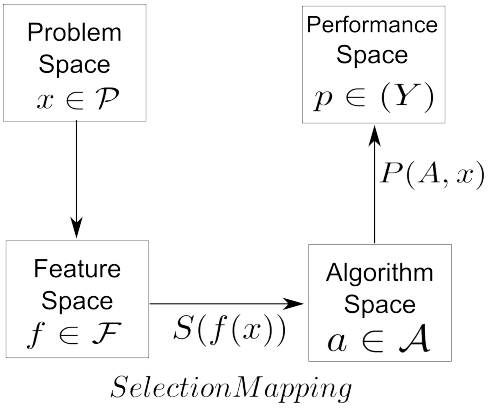


Figure 1: Schematic of the Algorithm Selection Problem [22]

2 RELATED WORK

A schematic of the ASP is shown in Figure 1 [22]. In Rice's definition:

- The problem space $\mathcal{P}$ represents a potentially infinite sized set of instances for the problem domain
- The feature space $\mathcal{F}$ describes a set of characteristics derived using feature extraction from $\mathcal{P}$
- The algorithm space $\mathcal{A}$ is the set of algorithms available for the problem domain
- The performance space $\mathcal{Y}$ maps each algorithm to a set of performance metrics [28].

The objective is to identify a mapping between $\mathcal{P}$ and $\mathcal{A}$ that maximises $\mathcal{Y}$. For a finite set of problem instances I, a fixed set of heuristics H and a single performance metric m, the Virtual Best Solver (VBS) is defined as a perfect mapping between I and H. The Single best solver (SBS) is the heuristic $\in H$ that achieves the best performance over I. A comprehensive review of different approaches towards algorithm-selection can be found in a number of survey papers in this active area of research [12, 14, 30]. Some of the most relevant approaches are described here.

One of the most common approaches to ASP is to identify features using expert knowledge, and then train machine-learning methods to predict the best performing algorithm(s) for an instance from its feature-profile [12]. However, identifying features that are significant in determining performance is complex, usually requires hand-crafting [10, 19, 20, 29], and often is not intuitive. Often the approach must also be combined with a *feature-reduction* method to simplify learning, e.g Principal Component Analysis (PCA) [17, 28], and understand the correlations between features and algorithm performance. An approach that tries to avoid having to hand-craft good features was described in [25] who evolve the *parameters* of a set of generic features of 1-d bin-packing problems that best improve the performance of KNN classifier. A different type of approach was proposed by Ross et al. [23] that could be used for ASP with constructive approaches to solution generation; rather than deriving features from the original description of an instance, features were derived from the current instance state each time a heuristic was applied. A learning classifier system XCS was used to map a set of problem-states to specific heuristics. An ASP approach that does not rely on explicit feature identification was proposed by Sim et al. [26]. Here, a system continuously generates novel heuristics which are maintained in an ensemble, and sample multiple problem instances from the environment. Heuristics that "win" an instance (perform best) are maintained. This was shown to rapidly produce solutions and generalise over the problem space, but required a greedy method of actually selecting between generated heuristics.

The majority of previous research just described uses classical machine-learning algorithms such as K-Nearest Neighbour (KNN), Wilson's Learning Classifier System [32] (XCS) and Neural Networks (NN). Recently, deep-learning algorithms have gained some traction in the ASP field due their ability to learn from extremely large data sets in reasonable time. Mao et al. [18] proposed a heuristics performance predictor using deep neural network trained on a large set of instances of 1-d bin-packing problem using 16 features as input. Their prediction system has achieved up to 72% validation accuracy to select the best performing heuristic that can generate a better quality bin packing solution. Although concerned with learning an optimisation method rather than algorithm-selection, Hu et al. [9] used a deep reinforcement learning (a Pointer Network), with 3D BPP to optimize the sequence of items to be packed into the bin by choosing the sequence, orientation and empty maximal space to pack cuboid shaped items. They claimed that their proposed method has obtained about 5% improvement than well-designed heuristic.

The approach proposed in this paper differs substantially from the previous work just described in that it abandons the need to derive features from a data set, circumventing the associated issues. Furthermore, as far as we are aware, it provides the first example of using a recurrent-neural network with long-short-term memory as an algorithm-selection technique for data which has temporal characteristics. Although such networks have demonstrated "ground-breaking performance on tasks as varied as image captioning, language translation and handwriting recognition" [16], their features have not been exploited within the ASP domain.

3 1-D BPP: HEURISTICS AND PROBLEM INSTANCES

The objective of the one dimensional bin packing problem (BPP) is to find a packing which minimises the number of containers, b, of fixed capacity c required to accommodate a set of n items with weights $\omega_j : j \in \{1 \dots n\}$ falling in the range $1 \leq \omega_j \leq c, \omega_j \in \mathbb{Z}$ whilst enforcing the constraint that the sum of weights in any bin does not exceed the bin capacity c. The lower and upper bounds on b, (b_l and b_u) respectively, are given by Equation 1. Any heuristic that does not return empty bins will produce, for a given problem instance, p, a solution using b_p bins where $b_l \leq b_p \leq b_u$.

In this study we are concerned with predicting the best heuristic for a problem instance based only on temporal information encoded in the sequence of items that are presented to the solver. By using a Recurrent Neural Network (RNN), we circumvent the requirement to first identify features that are typically used to facilitate mapping algorithm performance to problem instances. To allow us to achieve this goal a set of heuristics with complimentary performances on large instance sets is required.

It should be made clear that we address a specific variant of the offline BPP. All items to be packed are known before packing starts

but the order that items are presented to the packing heuristics is fixed and cannot change. There have been numerous studies over the decades that have investigated the performance of simple approximation algorithms for this variant of the BPP[2, 11]. These algorithms consider each instance as a fixed order sequence of items, only making a decision as to which bin to place each item into. We select 4 simple approximation algorithms from the literature specifically designed for this variation of the BPP [6].

- **First Fit (FF):** Places each item into the first feasible bin that will accommodate it.
- **Best Fit (BF):** Places each item into the feasible bin that minimises the residual space.
- **Worst Fit (WF):** Places each item into the feasible bin with the most available space.
- **Next Fit (NF):** Places each item into the current bin.

For all the algorithms listed, if no feasible bin is available to accommodate the next item then it is placed into a newly opened bin. NF is different to the other 3 algorithms in that it only ever considers the most recently opened bin. If an item cannot fit in the current bin that bin is closed and removed from the problem. The performance of an algorithm α on instance $\mathcal{I}$ is denoted by $A(\mathcal{I})$. $OPT(\mathcal{I})$ is the optimal solution for that instance. The worst-case performance ratio (WCPR) of an algorithm is defined as the smallest real number $r(A) > 1$ such that $\frac{A(\mathcal{I})}{OPT(\mathcal{I})} \leq r(A)$ for all possible instances. The WCPR of NF is known to be 2 [2] and it was recently concluded after many theoretical studies that the WCPR of FF and BF is $\frac{17}{10}$[3].

$$b_l = \left\lceil \sum_{j=1}^{n} \omega_j \div c \right\rceil, \; b_u = n \tag{1}$$

There are many sets of benchmark problem instances available in the literature. However, as noted in [30], while benchmark problems are suitable for comparing different algorithms performances, they are often unsuitable for investigating the ASP due to being similar in structure and often tailored towards the abilities of specific solvers. For illustration, consider the following: taking 5 well researched data sets totalling 1370 instances from the literature [5, 24] we find that if the problems are treated in the order that they are published, FF is the dominant heuristic solving 955 instances better than any of the other heuristics. BF wins on 414 instances and WF wins on 1. The success of FF is due to the fact that many of the published benchmarks are ordered using the best known solution and hence FF simply recreates that solution. If the item orders are randomised (we do this once only for interest) then BF, as expected, becomes the Single Best Solver winning on 926 instances. FF wins on 381, WF 61 and NF 2. Clearly using these relatively small sets of benchmark problems would lead to dramatically unbalanced data sets and hinder an investigation into the ASP.

In order to address this problem and create balanced data sets for our investigation, we use an Evolutionary Algorithm (EA) to generate 4 data sets, each with 4000 instances and using the bounds and distributions shown in Table 1. Each set of 4000 problem instances comprises of 4 sub-classes of 1000 instances that are uniquely solved best (according to Equation 2) by one of the four algorithms under investigation. Algorithm 1 describes how each sub-class of instances is generated. For each instance a population of candidate sequences is initialised with n_{items} and weights randomly drawn from the desired distribution. The EA then attempts to evolve the order of items such that the performance of the target algorithm α_{target} exceeds the performance of the other algorithms $\in \mathcal{A}$

Table 1: Data set Generation Parameters. Bin Capacity is fixed at 150

Data set	n_{items}	LowerBound - UpperBound	Distribution $\mathcal{D}$
DS1	120	[40-60]	Gaussian
DS2	120	[20-100]	Uniform
DS3	250	[40-60]	Gaussian
DS4	250	[20-100]	Uniform

Unlike in [1], who used a binary representation and evolved the weights of items, we use an integer representation: a chromosome is simply the sequence of n_{items} defining that instance. Item weights are randomly initialised at the start, then do not change. Only the order of items is evolved, preserving the desired weight distribution

Algorithm 1 EA Pseudo Code

Require: $\mathcal{A}$ = :The set of algorithms
Require: α_{target} = :The target algorithm
Require: $\mathcal{D}_{target}$ = :The target distribution
Require: $\mathcal{I} = \emptyset$:The set of instances being evolved
Require: $n_{instances}$ = : The number of instances to generate
Require: n_{items} = : The number of items in each instance
1: **repeat**
2: $population \leftarrow$ initialise($popSize, n_{items}, \mathcal{D}_{target}$)
3: **for** $i \leftarrow 1, maxIter$ **do**
4: $parent_1 \leftarrow$ select($population$)
5: $parent_2 \leftarrow$ select($population$)
6: $child \leftarrow$ crossover($parent_1, parent_2$)
7: $child \leftarrow$ mutate($child$)
8: $child_{fitness} \leftarrow$ evaluate($child$)
9: $population \leftarrow$ replace($child$)
10: **end for**
11: **if** $evaluate(population_{best}) > 0$ **then**
12: $\mathcal{I} \leftarrow \mathcal{I} + population_{best}$
13: **end if**
14: **until** $|\mathcal{I}| = n_{instances}$
15: **return** $\mathcal{I}$

In the pseudo-code shown, the target algorithm α_{target} is the algorithm $\alpha \in \mathcal{A}$ that we are attempting to evolve instances for, such that the solution fitness (defined by Equation 2) using α is better than the solutions produced by all the other algorithms $\in \mathcal{A}$. $\mathcal{D}_{target}$ is the distribution from Table 1 that is used to initialise the *population* (line 2). When using a Gaussian distribution, item weights are generated using a Gaussian distribution with standard deviation $(ub - lb) \div 2$ and mean $lb + (ub - lb) \div 2$. Items generated with weight $w < lb$ or $w > ub$ are discarded. The lower and upper bounds on item weights are given in Table 1. For each data set we run Algorithm 1 once for each $\alpha_{target} \in \mathcal{A}$ using $n_{instances=}$ 1000

resulting in 4 balanced data sets of 4000 instances where for each data set 1000 instances are solved best by each $\alpha \in \mathcal{A}$

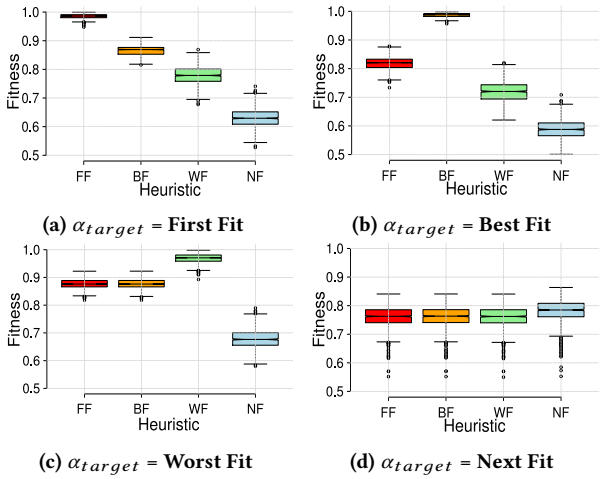
$$Fitness = \sum_{j=1}^{n} (\frac{fill_j}{c})^k \div n \quad (2)$$

Falkenauer's performance metric defined by Equation 2 [5] is used to gauge the quality of a solution produced by an individual algorithm $\alpha \in \mathcal{A}$. In this study k is fixed at 2. C is the bin capacity which is fixed at 150, $fill_j$ is the sum of the item sizes in bin_j and n is the number of bins used. The evaluation function used in Algorithm 1 assigns a score to each algorithm using Equation 2 and then orders the algorithms from best to last $\alpha_{1..4}$. The fitness of a chromosome is then $evaluation = fit(\alpha_{target}) - fit(\alpha_2)$. i.e. The fitness of a chromosome turns positive when the target algorithm has the best fitness on that instance. The population size is fixed at 500. Tournament selection is used with a tournament size of 2. Crossover is single point crossover applied with 99% probability and mutation (applied with 2% probability) simply swaps the order of two items in the chromosome. The child produced replaces the worst member of the population if it has higher fitness. The algorithm is run for $maxIter = 1X10^6$ iterations. At the end of that time the population best is kept only if the target algorithm is the best performing algorithm on that instance. Experiments were conducted on a High Performance Cluster with 23 nodes each comprising 2 × Xeon E5-2640v3 2.60GHz 8Core /16 Thread CPUs with 64gb 2133Mhz Memory per node. Code was implemented in Java.

Each of the Figures 2 a-c shows the performance of all algorithms on the subset of 1000 instances from DS2 identified in the caption by the target class. In all cases the target algorithm significantly outperforms the other algorithms. Figure 2d shows that for NF the difference in performance between the target class and the other algorithm appears less pronounced although it is still highly significant (comparing BF and NF using a paired Wilcoxon Signed-Rank Test gives a $p - value$ of $2.2e^{-16}$). NF is known to have the worst WCPR of all the algorithms investigated and the task of evolving these instances required many more attempts than for the other algorithms, taking around 3 times the computational effort to find 1000 instances for each data set. Plots for the other data sets show similar patterns but are omitted due to space limitations.

Figure 3 shows the performance of all algorithms on the full set of 4000 instances in DS2. Best Fit is the Single Best Solver across all 4000 instances (this is the same for the other data sets). The plot labeled VBS shows the Virtual Best Solver i.e. the performance of the Oracle that greedily selects the best algorithm for each instance. The objective for the ASP is to achieve the same performance as the VBS. To further illustrate the diversity in performance on the evolved instances Figure 4 shows the performance of each algorithm on DS2 reduced from 4 dimensional performance space to 2 dimensions using PCA.

Clearly we have succeeded in generating a set of instances that cover the performance space from the perspective of the algorithms and metric investigated. Figure 5 emphasises this further showing a Principal Component Analysis plot of the performance of each algorithm on the full set of 16,000 instances reduced from 4 to 2 dimensions. BF and FF are the closest in terms of their eigenvectors but are clearly still disparate. NF is isolated as the worst performing



(a) α_{target} = First Fit (b) α_{target} = Best Fit

(c) α_{target} = Worst Fit (d) α_{target} = Next Fit

Figure 2: Each plot shows how the 4 heuristics perform on the sub-class of 1000 instances evolved for each target heuristic (DS2 120 items U[20, 100])

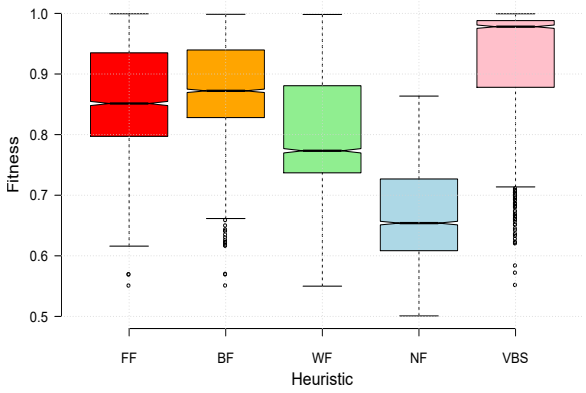


Figure 3: DS2 Performance

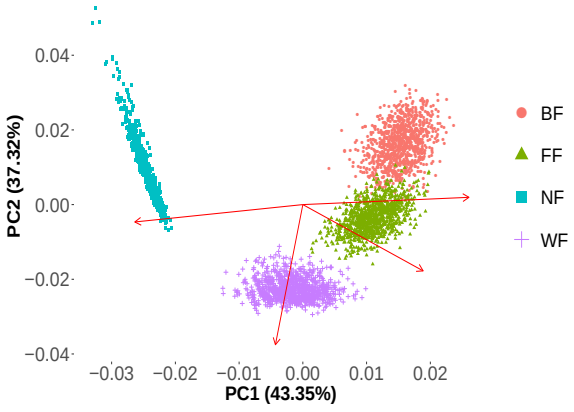


Figure 4: PCA plot of the performance of all heuristics on all 4,000 problem instance from DS2

heuristic — as expected due to its poor WCPR. If we shuffle the orders of instances to destroy the evolved sequences and replot with PCA (Figure 6) it it clear that there is now no identifying structure apparent in the plot that a selection method might exploit. On the randomised instances, BF wins for 10875 instances, FF 4586, WF 539 and NF fails to win on any instance.

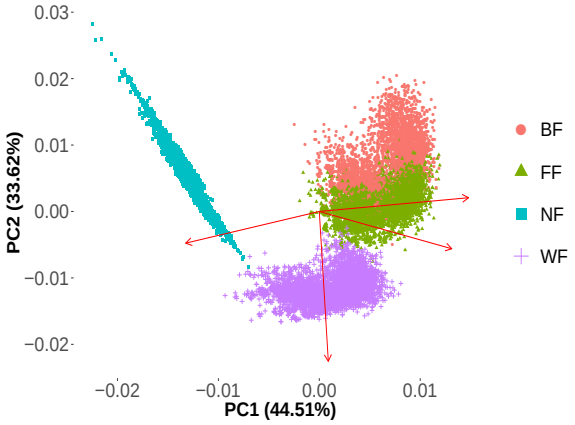


Figure 5: PCA plot of the performance of all heuristics on all 16,000 evolved problem instances

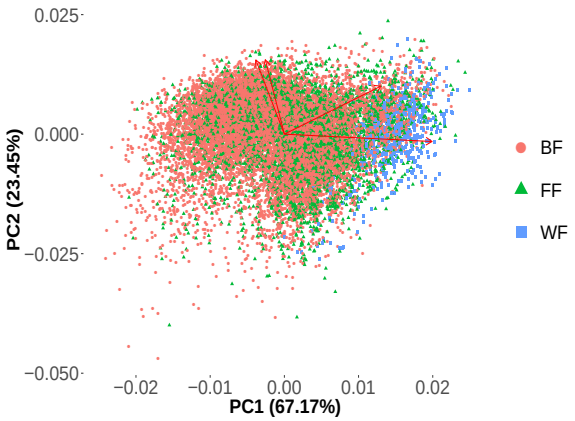


Figure 6: PCA plot of the performance of all heuristics on all 16,000 instances after the ordering of items is randomised

4 A DEEP LEARNING MODEL FOR ALGORITHM SELECTION

Conventional Machine Learning techniques used for the ASP consider vectors of features with little consideration to the order that features are presented. Candidate features typically describe spatial or statistical characteristics, identified for the problem domain while temporal information can be overlooked (although it could be argued that some temporal information is implicit in those approaches that dynamically calculate problem state and use this to select heuristics [23, 25]). We argue that for the algorithms investigated, that are restricted to packing items in the order that they are presented, it is crucial to consider information explicit in the ordering as the dominant factor affecting the performance of different solvers. Deep Learning has achieved ground breaking results in applications where the input is formatted as time-series data or in domains where sequences have specific orderings but without any explicit notion of time [16]. Examples including video and image recognition, natural language processing, music generation and speech recognition [7, 21, 27]

The benefit of algorithm selection relies on complementary performances between algorithms over large sets of problem instances for a given metric. Section 3 highlights that benchmark instances, or randomly generated instances, are largely unbalanced in terms of the relative performance they elicit in different algorithms. In order to facilitate our investigation of algorithm selection using ordered data, we have generated balanced data sets that maximise the complementary performance of a fixed set of algorithms. We have shown that the order items are presented is crucial with respect to the performance of different solvers. Here we use DL in an attempt to map information encoded in these sequences that identifies the best performing algorithm.

Out of curiosity, and without expectation, we conducted exploratory experiments with conventional ML techniques using only the instance data as input — that is, the ordered list of item weights was presented as input. We conducted a 10 fold cross validation on DS4 using both the Multi Layer Perceptron (MLP) and the Random Forest (RF) classifiers available in Weka [4] without altering any of the default parameters. The RF achieved 67.55 % accuracy with a Kappa statistic of 57.3% (explained in the next section). MLP was equally successful, achieving 67.08% accuracy with Kappa = 56.1%. Although better than expected, we hypothesise that a DL approach that has been designed to work with sequential data should provide more rigorous results.

We select a recurrent neural-network (RNN) as a machine learning method designed to learn from sequences of time series data. RNNs are one of the two most common architectures described under the umbrella term deep-learning (DL). They differ from Feedforward Neural Networks due to the presence of cyclic connections from each layers' output to the next layers input, with feedback loops returning to the previous layer (Fig 7-a) [16, 31]. This structure prevents traditional back-propagation being applied since there is not an end point where the back-propagation can stop. Instead, Back Propagation Through Time (BPTT) is applied: the RNN structure is unfolded to several neural networks with certain time steps and then the traditional back-propagation is applied to each one of them (Fig 7-b) [31]. RNNs are specifically designed to learn from time-series data where temporal information explicit in the order of sequences is used to identify relationships between the data and the expected outputs from the network.

In this study we use a specialised RNN known as a Long Short-Term Memory (LSTM) [8] that has been shown to be highly efficient and extremely effective in learning long-term dependencies from sequences of ordered data. LSTMs incorporate additional gates between neurons that can retain, retrieve and forget information over long periods of time [7]. In the LSTM network, the classic neurons in the hidden layer are replaced by memory blocks. Fig

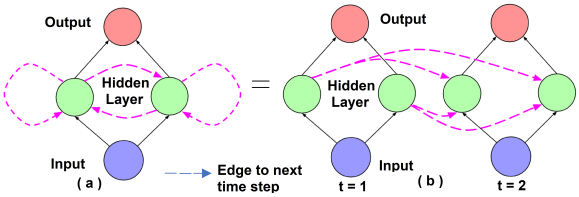


Figure 7: (a) a Simple RNN and (b) an example of unfolded RNN with two time steps [16]

8 shows that the block's input comes from the network through the input node and the only outputs from the block to the rest of the network emanate from the output gate multiplication. The input and output gates multiply the input and output of the cell respectively, while the forget gate multiplies the cell's previous state. A more comprehensive description of the rapidly expanding field of DL, which has many competing, but no prevalent architectures, is outwith the scope of this study.

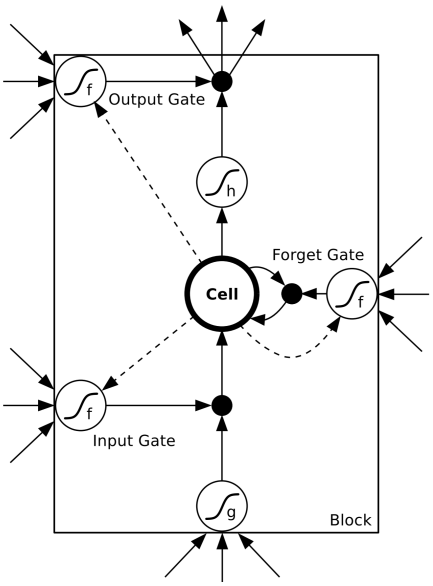


Figure 8: LSTM memory block with one cell where the small black circles are multiplications and the weighted connections from the cell to the gates are shown with dashed lines [7]

5 EXPERIMENTS AND RESULTS

We use the Keras library[2] implementation of an LSTM in "sequence-to-one mode" where input is an ordered list of item weights and output is a "one-hot" encoding using 4 bits to identify the best heuristic (1000 = BF, 0100 = FF, 0010 = NF, 0001 = WF). Experiments are conducted on Google Colab[3] with Tensor Processing Unite (TPU) run-time used to execute the experiments. A preliminary empirical investigation was conducted to tune the LSTM architecture and hyper-parameters. The "Adam" optimiser [13] was selected due to its reported accuracy, speed and low memory requirements. We undertook preliminary investigations using between one and four LSTM layers. Batch sizes of between 8 and 128 were used, with learning iterations varied between 100 and 700. Based on our findings we selected an LSTM with 2 layers, each layer composed of 32 neurons. We applied a batch-size of 32 using the "Adam" Optimiser with a learning rate of 0.001 and categorical cross-entropy used as the loss function (since we have more than two classes). We used 300 learning iterations for DS1 and DS2 and 700 learning iterations for DS3, DS4 and DS5 since the longer instances were found to require more learning iterations.

[2]https://github.com/fchollet/keras
[3]https://colab.research.google.com/notebooks/welcome.ipynb

Table 2: Mean Kappa values from experiments on DS1 to DS5 from the validation set

Data set	Mean Kappa	std	Avg Learning Iteration
DS1	71.50%	(+/- 6.33%)	251
DS2	86.54%	(+/- 2.31%)	102
DS3	74.58%	(+/- 4.77%)	451
DS4	93.79%	(+/- 2.94%)	352
DS5	76.08%	(+/- 2.63%)	337

We conduct independent experiments using DS1-4 to train and test our LSTM models. A further experiment is carried out using a data set composed of a mixture of instances taken from all 4 data sets (identified as DS5). We do this to investigate whether our LSTM model can generalise across a mixed set of instances of different lengths with item weights drawn from different probability distributions and bounds. DS5 contains 4000 instances with 1000 instances selected from each data set. For each data set 250 instances were selected at random for each class (FF, BF, WF and NF), resulting in a balanced data set containing equal numbers of instances from each class and each distribution.

Each data set (DS1-5) was split into training (80%) and test (20%) sets while maintaining the balance in size of each target class (each test set has 800 instances where 200 are solved best by each algorithm). The LSTM was trained using 10-fold cross validation using 10% of the training instances to verify each fold. The best model from each experiment was then tested on the corresponding test set comprising 800 unseen instances.

Table 2 shows results achieved on the validation sets used during training. Results on DS2 and DS4 are better than on the other data sets suggesting that correlations are easier to find in the data sets comprised of item weights generated from a wider range of values following a uniform distribution. It is well known that problems with an average weight of $\frac{C}{3}$ are more difficult to solve [5] and it is interesting that problems with those characteristics are more difficult to classify. Kappa provides a measure of statistical agreement between the predicted class and the actual class that takes into consideration the probability that the model classifies correctly by chance [15]. Kappa values >75% are considered to be a strong indicator that the classifier's performance is excellent; between 40% to 75% is considered as fair to good; lower than 40% is understood to be weak.

For each data set we select the best model (the model with the highest Kappa statistic from the 10-fold validation) and use it to

predict the target class for the corresponding unseen test set. Table 3 shows results achieved on each test set. We report Kappa, accuracy and precision as indicators of the LSTM's predictive ability. The p-values reported compare the range of fitness values achieved by the predicted algorithm in a pairwise fashion to the fitness achieved by the SBS and the VBS. Falkenauer's performance metric (equation 2) is used to evaluate solution quality and returns a value between 0 and 1. However, 1 is only achievable if all bins are filled to capacity and for individual instances the fitness achieved by the VBS can be as low as 0.5 (as is shown in Figure 3 for DS2). We therefore use a Wilcoxon signed-rank test to evaluate significance in a pairwise fashion for all comparisons of performance.

Table 3: Kappa, classification accuracy and average precision values of the best model in each experiment from the test set

Metric	DS1	DS2	DS3	DS4	DS5
Kappa	76.5%	88.67%	77.83%	94.33%	77.83%
Accuracy	82.38%	91.5%	83.38%	95.75%	83.38%
Average precision	93%	97%	93%	99%	92%
P value with SBS	10^{-48}	10^{-78}	10^{-43}	10^{-80}	10^{-41}
P value with VBS	10^{-7}	0.044	$2x10^{-6}$	0.31	$3x10^{-6}$

We observe that the experiments on instances from DS2 and DS4 obtain better results than the experiments on instances from DS1 and DS3. It might be that the instances that are evolved with items in the range [40-60] have fewer distinct item sizes than instances generated with items in the range [20-100], regardless of the length of the instances. It is interesting to note that for both distributions the results on the longer instances exceed those reported on the data sets with smaller numbers of items. We conjecture that the longer instances provide the LSTM with more information, hence increasing the ability to determine patterns in the item sequences. Training the LSTM on the longer instances requires significantly more learning iterations before the models converge, but the trained models from the experiments on DS3 and 4 with 250 items are more accurate than the models trained on shorter instances with only 120 items and the same distribution (i.e. results on DS3 are more accurate than DS1 and those for DS4 are more accurate than for DS2). The results of the experiment conducted on the combined DS5 set shows intermediate results with accuracy between that achieved on the experiments on instances with uniform distribution and those with Gaussian distribution. This model is able to generalise over instances sampled from all of the problem lengths and the different weight distributions investigated without any apparent loss of precision.

Table 4 presents confusion matrices extracted from the experiments conducted on the test sets from DS4 and DS5. On DS4, 766 out of 800 test instances correctly classified (96%). In most other cases (only 2 matrices are shown due to space limitations) the models were most frequently confused when attempting to classify the sequences identified as being solved best by FF and BF. It is interesting to note that these two algorithms are the closest in terms of performance as is shown by the proximity of their eigenvectors in Fig 5. Similarly, instances labeled as NF appear to be the easiest to identify and correspondingly are the most isolated in the performance space. It appears that the patterns shown by conducting a PCA analysis of the performance space are correlated with the ability of LSTM to identify the best performing algorithm from the raw instance sequences.

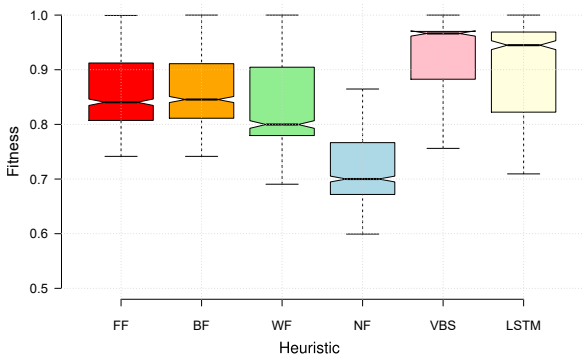


Figure 9: Evaluating LSTM predictor VS Traditional heuristics and VBS for DS1

Table 4: The Confusion Matrix of the best model in experiment on DS4 and DS5 from the testset

	DS4				DS5			
Heuristic	BF	FF	NF	WF	BF	FF	NF	WF
BF	194	4	0	2	160	36	0	4
FF	6	185	1	8	48	133	0	19
NF	0	0	196	4	0	0	198	2
WF	0	8	1	191	2	13	9	176

Figures 9 to 13 show the performance distributions achieved by each of the target algorithms investigated, by the VBS and by our LSTM selector, for each of the test sets DS[1-5]. For all experiments the LSTM significantly improves on the SBS (BF is the SBS for all data sets) and is close to the VBS (the perfect mapping) in some cases. On DS4 a pairwise comparison of the performance of the VBS and our LSTM returns a p-value of 0.31 indicating that there is no significant difference between the two performance vectors. Comparing the median performance of our LSTM predictor and the SBS we achieve between 7% to 11% improvement which is between 0% to 2% less than the VBS.

We have used Falkenauer's performance metric (Equation 2) throughout this paper to assess the quality of individual solutions as it provides a higher precision measure of performance than simply using the number of bins. However, the overall objective of the BPP is to minimise the number of bins used to pack a set of items. Ultimately, the number of containers defines the cost of any real-world solution. Table 5 shows the number of bins required to pack all 800 test instances for each DS[1-5] and contrasts this against the lowest possible number of bins used by the VBS and the number of bins needed using the algorithms predicted by our LSTM selector. The LSTM uses between 1.2% and 2.3% fewer bins than the SBS and between 0.2% and 1.4% more than the VBS. On DS4 we use over 1000 bins fewer than the SBS and only 238 (0.2%) more than the VBS which uses 83,823 bins.

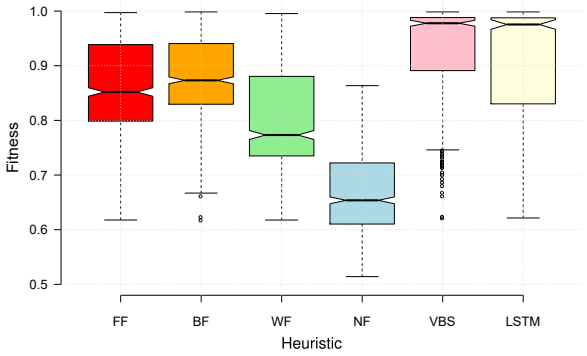


Figure 10: Evaluating LSTM predictor VS Traditional heuristics and VBS for DS2

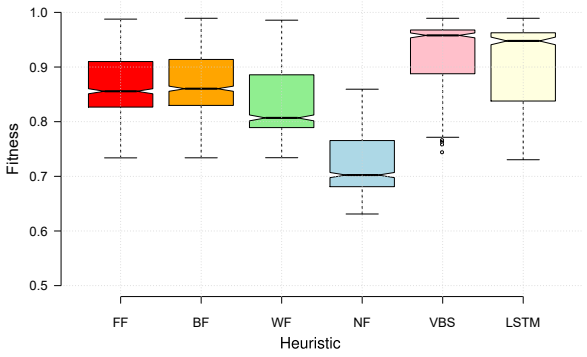


Figure 11: Evaluating LSTM predictor VS Traditional heuristics and VBS for DS3

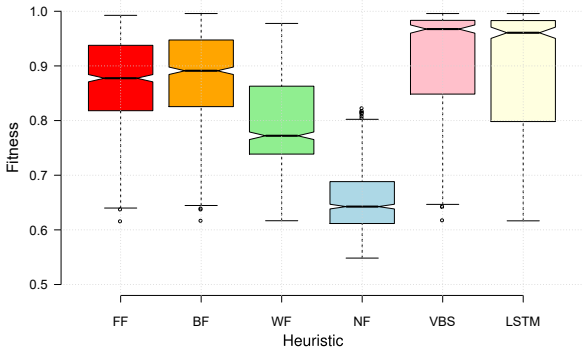


Figure 12: Evaluating LSTM predictor VS Traditional heuristics and VBS for DS4

Table 5: Total Bins required to pack instances in the test set

	FF	BF	WF	NF	VBS	LSTM
DS1	34815	34722	35357	38357	33439	33908
DS2	41494	41062	43031	47961	39929	40133
DS3	71839	71741	73312	79498	69638	70359
DS4	85677	85166	89435	100439	83823	84061
DS5	58536	58242	60345	66672	56772	57525

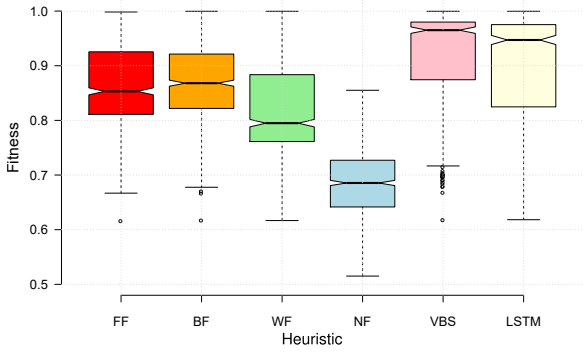


Figure 13: Evaluating LSTM predictor VS Traditional heuristics and VBS for DS5

6 CONCLUSION

We have described an approach to algorithm selection for combinatorial problems that exhibit an ordering with respect to the elements of the problem and how they should be dealt with. The approach does *not* require the design and selection of features to describe an instance. A deep-neural network (RNN-LSTSM) was trained using the sequence of items representing an instance directly as input to predict the best algorithm to solve the instance. The accuracy of the model ranges from 82-96%, depending on the data set used. In terms of performance, we show between 7% to 11% improvement over the single best solver (SBS) when considering the data set a whole, and 0% to 2% lower than the virtual best solver (VBS). As far we aware, this is the first time that such an approach has been used, and represents a significant step forward in algorithm selection, where the difficulties associated with defining suitable features and selecting from large sets of potential features are well understood.

The method was thoroughly evaluated on 4 different large data sets, exhibiting different numbers of items and different distributions of item-sizes. To facilitate training, we evolved a new, large database of instances, in which each instance has a distinct best-solver. This provides a balanced data set to use in training. The database is available as a resource to other researchers; moreover, the method used to evolve these instances could be generalised to evolve new instances for other classes of temporal problem, for example job-shop scheduling.

Future work will focus on extending the approach to larger and more complex sets of algorithms and to applying the method to other domains that have a temporal nature such as flow-shop/job-shop scheduling. We also intend to investigate if the method can be adapted to online problems where continuous streams of items are presented. By using a moving window, only examining the next n items to be packed, our method may be able to adapt to a continuously changing environment. Ultimately, the goal is to extract knowledge from the trained models in order to gain new insight into the correlation between orderings and predicted results.

REFERENCES

[1] Ivan Amaya, José Carlos Ortiz-Bayliss, Santiago Enrique Conant-Pablos, Hugo Terashima-Marín, and Carlos A Coello Coello. 2018. Tailoring Instances of the 1D Bin Packing Problem for Assessing Strengths and Weaknesses of Its Solvers. In *International Conference on Parallel Problem Solving from Nature*. Springer,

373–384.

[2] Maxence Delorme, Manuel Iori, and Silvano Martello. 2016. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research* 255, 1 (2016), 1 – 20. https://doi.org/10.1016/j.ejor.2016.04.030

[3] György Dósa and Jiří Sgall. 2014. Optimal Analysis of Best Fit Bin Packing. In *Automata, Languages, and Programming*, Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 429–441.

[4] Frank Eibe, A. Hall Mark, and H. Witten Ian. 2016. *The WEKA Workbench. Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques* (fourth edition ed.). Morgan Kaufmann.

[5] Emanuel Falkenauer and Alain Delchambre. 1992. A genetic algorithm for bin packing and line balancing. In *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*. IEEE, 1186–1192.

[6] M. R. Garey and D. S. Johnson. 1981. *Approximation Algorithms for Bin Packing Problems: A Survey*. Springer Vienna, Vienna, 147–172. https://doi.org/10.1007/978-3-7091-2748-3_8

[7] Alex Graves. 2012. Supervised sequence labelling. In *Supervised sequence labelling with recurrent neural networks*. Springer, 5–13.

[8] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Comput.* 9, 8 (Nov. 1997), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

[9] Haoyuan Hu, Xiaodong Zhang, Xiaowei Yan, Longfei Wang, and Yinghui Xu. 2017. Solving a new 3d bin packing problem with deep reinforcement learning method. *arXiv preprint arXiv:1708.05930* (2017).

[10] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. 2014. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence* 206 (2014), 79 – 111.

[11] D. Johnson, A. Demers, J. Ullman, M. Garey, and R. Graham. 1974. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. *SIAM J. Comput.* 3, 4 (1974), 299–325.

[12] Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. 2018. Automated Algorithm Selection: Survey and Perspectives. *Evolutionary computation* (2018), 1–47.

[13] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[14] Lars Kotthoff. 2016. *Algorithm Selection for Combinatorial Search Problems: A Survey*. Springer International Publishing, Cham, 149–190. https://doi.org/10.1007/978-3-319-50137-6_7

[15] J Richard Landis and Gary G Koch. 1977. The measurement of observer agreement for categorical data. *biometrics* (1977), 159–174.

[16] Zachary C Lipton, John Berkowitz, and Charles Elkan. 2015. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019* (2015).

[17] Eunice López-Camacho, Hugo Terashima-Marín, Gabriela Ochoa, and Santiago Enrique Conant-Pablos. 2013. Understanding the structure of bin packing problems through principal component analysis. *International Journal of Production Economics* 145, 2 (2013), 488–499.

[18] Feng Mao, Edgar Blanco, Mingang Fu, Rohit Jain, Anurag Gupta, Sebastien Mancel, Rong Yuan, Stephen Guo, Sai Kumar, and Yayang Tian. 2017. Small Boxes Big Data: A Deep Learning Approach to Optimize Variable Sized Bin Packing. In *Third IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2017, Redwood City, CA, USA, April 6-9, 2017*. 80–89.

[19] Eugene Nudelman, Kevin Leyton-Brown, Holger H. Hoos, Alex Devkar, and Yoav Shoham. 2004. Understanding Random SAT: Beyond the Clauses-to-Variables Ratio. In *Principles and Practice of Constraint Programming – CP 2004*, Mark Wallace (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 438–452.

[20] J. Pihera and N. Musliu. 2014. Application of Machine Learning to Algorithm Selection for TSP. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*. 47–54. https://doi.org/10.1109/ICTAI.2014.18

[21] Samira Pouyanfar, Saad Sadiq, Yilin Yan, Haiman Tian, Yudong Tao, Maria Presa Reyes, Mei-Ling Shyu, Shu-Ching Chen, and S. S. Iyengar. 2018. A Survey on Deep Learning: Algorithms, Techniques, and Applications. *ACM Comput. Surv.* 51, 5, Article 92 (Sept. 2018), 36 pages. https://doi.org/10.1145/3234150

[22] John R Rice. 1976. The Algorithm Selection Problem. In *Advances in Computers*, Morris Rubinoff and Marshall C. Yovits (Eds.). Vol. 15. Elsevier, 65 – 118.

[23] Peter Ross, Sonia Schulenburg, Javier G Marín-Bläzquez, and Emma Hart. 2002. Hyper-heuristics: learning to combine simple heuristics in bin-packing problems. In *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 942–948.

[24] Armin Scholl, Robert Klein, and Christian Jurgens. 1997. Bison: A fast hybrid procedure for exactly solving the one-dimensional bin packing problem. *Computers & Operations Research* 24, 7 (1997), 627 – 645. https://doi.org/10.1016/S0305-0548(96)00082-2

[25] Kevin Sim, Emma Hart, and Ben Paechter. 2012. A Hyper-heuristic Classifier for One Dimensional Bin Packing Problems: Improving Classification Accuracy by Attribute Evolution. In *Proceedings of the 12th International Conference on Parallel Problem Solving from Nature - Volume Part II (PPSN'12)*. Springer-Verlag, Berlin, Heidelberg, 348–357. https://doi.org/10.1007/978-3-642-32964-7_35

[26] Kevin Sim, Emma Hart, and Ben Paechter. 2015. A lifelong learning hyper-heuristic method for bin packing. *Evolutionary computation* 23, 1 (2015), 37–67.

[27] Sandro Skansi. 2018. *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer.

[28] Kate Smith-Miles, Davaatseren Baatar, Brendan Wreford, and Rhyd Lewis. 2014. Towards objective measures of algorithm performance across instance space. *Computers & Operations Research* 45 (2014), 12 – 24. https://doi.org/10.1016/j.cor.2013.11.015

[29] Kate Smith-Miles and Jano van Hemert. 2011. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence* 61, 2 (01 Feb 2011), 87–104. https://doi.org/10.1007/s10472-011-9230-5

[30] Kate A. Smith-Miles. 2009. Cross-disciplinary Perspectives on Meta-learning for Algorithm Selection. *ACM Comput. Surv.* 41, 1, Article 6 (2009), 25 pages. https://doi.org/10.1145/1456650.1456656

[31] Haohan Wang and Bhiksha Raj. 2017. On the origin of deep learning. *arXiv preprint arXiv:1702.07800* (2017).

[32] Stewart W. Wilson. 1995. Classifier Fitness Based on Accuracy. *Evol. Comput.* 3, 2 (June 1995), 149–175. https://doi.org/10.1162/evco.1995.3.2.149