A Deep Learning Approach to Predicting *Solutions* in Streaming Optimisation Domains

Mohamad Alissa
School of Computing, Edinburgh Napier University
M.Alissa@napier.ac.uk

Kevin Sim
School of Computing, Edinburgh Napier University
K.Sim@napier.ac.uk

Emma Hart
School of Computing, Edinburgh Napier University
E.Hart@napier.ac.uk

ABSTRACT

In the field of combinatorial optimisation, per-instance algorithm selection still remains a challenging problem, particularly with respect to streaming problems such as packing or scheduling. Typical approaches involve training a model to predict the best algorithm based on features extracted from the data, which is well known to be a difficult task and even more challenging with streaming data. We propose a radical approach that bypasses algorithm-selection altogether by training a Deep-Learning model using solutions obtained from a set of heuristic algorithms to directly predict a *solution* from the instance-data. To validate the concept, we conduct experiments using a packing problem in which items arrive in batches. Experiments conducted on six large datasets using batches of varying size show the model is able to accurately predict solutions, particularly with small batch sizes, and surprisingly in a small number of cases produces better solutions than any of the algorithms used to train the model.

CCS CONCEPTS

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

KEYWORDS

Algorithm Selection Problem, Deep Learning, Bin-packing

ACM Reference Format:
Mohamad Alissa, Kevin Sim, and Emma Hart. 2020. A Deep Learning Approach to Predicting *Solutions* in Streaming Optimisation Domains . In *Genetic and Evolutionary Computation Conference (GECCO '20), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377930.3390224

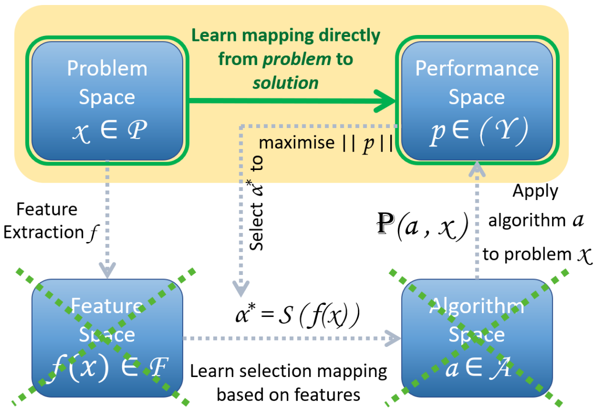
Learn mapping directly from *problem* to *solution*
Problem Space $x \in \mathcal{P}$
Performance Space $p \in (\mathcal{Y})$
maximise $|| p ||$
Select α^* to
Feature Extraction f
$\mathbf{P}(a, x)$
Apply algorithm a to problem x
Feature Space $f(x) \in \mathcal{F}$
$\alpha^* = S(f(x))$
Learn selection mapping based on features
Algorithm Space $a \in \mathcal{A}$

Figure 1: The traditional schematic of the Algorithm Selection Problem [22], with proposed modifications shown in green

1 INTRODUCTION

The per-instance algorithm selection problem (ASP) was first introduced by Rice [22] over forty years ago. Informally, it can be stated as follows: given a problem space $\mathcal{P}$ containing a potentially infinite sized set of instances for a given problem domain, a feature space $\mathcal{F}$ describing a set of characteristics extracted from $\mathcal{P}$, an algorithm space $\mathcal{A}$ containing the set of algorithms available to solve the problem instances, a performance space $\mathcal{Y}$ that maps each algorithm in $\mathcal{A}$ to a set of performance metrics, find a mapping that maps the problem characteristics in $\mathcal{F}$ to an algorithm space $\mathcal{A}$ that maximises the performance metrics in $\mathcal{Y}$.

For a representative set of problem instances, the algorithm selection process normally includes three main tasks: (1) extracting useful features; (2) determining/designing good algorithms(s); (3) selecting an appropriate mapping technique $\mathcal{S}(\mathcal{F}(\mathcal{X}))$. However, it is well known that obtaining useful features is not an easy task and often requires expert input which is not always available [10, 17, 20, 23]. This is even more challenging with problems that have a temporal nature, for instance job-shop scheduling [14, 30] and online bin-packing [13, 21], where tasks/items continually arrive to be assigned to a machine or packed in a particular bin in the same sequence that they arrive. In such cases, features should capture the temporal information present in the sequence, which cannot be achieved using classical statistical approaches for feature definition. Step (2) (determining/designing algorithms) is generally more straightforward, particularly given recent advances in the field of automated algorithm design [24], for instance using genetic-programming or grammatical-evolution to design new algorithms and heuristics. However, the final step of learning a mapping between the feature-space $\mathcal{F}$ and the algorithm-space $\mathcal{A}$ is also non-trivial, usually involving feature-selection and tuning of a classification model.

Although the difficulties associated with feature-selection can be partially mitigated by recent proposals to bypass the feature-extraction task by capturing the temporal information implicit in a sequence and using this as direct input to a Deep-Learning algorithm to learn a mapping from *instance to algorithm* [1], here we propose a more radical solution: train a Deep-Learning algorithm to *directly predict the solution to an instance from the instance-data itself.* The approach is illustrated in figure 1, which shows Rice's original diagram modified according to our proposal. Rather than constructing a feature-space from the instance-data and then mapping this to the algorithm-space, our approach learns a mapping directly from problem to solution for a given performance metric. The model is *trained* using solutions obtained by greedy selection of the best algorithm from the algorithm space $\mathcal{A}$ (which does not require the construction of the feature-space $\mathcal{F}$).

To learn this direct mapping, we make use of a deep-learning technique known as an Encoder-Decoder LSTM[1] which has received much attention in the natural-language domain in tasks such as text-translation which generate a translated sentence directly from an input consisting of a stream of words [5, 25]. To prove the concept is viable, we provide results on a simplified version of a streaming problem in the domain of bin-packing, in which items arrive in *batches.* Once a batch is packed, the next batch is treated as a new sub-problem. The approach represents a paradigm-shift in the way in which we approach algorithm-selection, in removing the need to select an algorithm altogether, and instead directly returning the best solution.

We investigate the following questions:

(1) To what extent can an Encoder-Decoder LSTM directly predict an optimised solution from a stream of data describing a problem-instance?
(2) To what extent are the results influenced by the length of the stream?
(3) To what extent is the accuracy of the model influenced by the diversity of algorithm-space (and therefore implicitly the solution-space) used to train the model?

Experimental results using our model are very encouraging. The model achieves an accuracy of 85% - 99% on unseen instances in term of predicting the *best* solution for small batch-size, although accuracy decreases as batch-size increases and/or the diversity of heuristics used to create solutions for training increases. It also returns a high percentage of *valid* solutions (92% - 99%) on the smaller batches. Finally, we show that in some cases it is also capable of providing better solutions than any of the algorithms on which it was trained: interestingly, there appears to be a trade-off between accuracy and "creativity", suggesting a ripe avenue for further work.

2 RELATED WORK

The vast majority of previous work in the ASP domain follows the cycle first outlined by Rice [22], shown in figure 1 and discussed in the previous section [11], in which an important first step is the extraction of a feature-set. However, as Rice himself, noted: "*The determination of the best (or even good) features is one of the most important, yet nebulous, aspects of the algorithm selection problem.*". Extracting good features that correlated to algorithm performance remains a complex and hard task [10, 17, 20, 23]. Specifically within combinatorial optimisation, significant attention has been made to defining useful feature-sets in popular domains such as TSP [20] (who define 287 features), other domains remain more elusive, although Kerschke *et al* note that there is significant potential to derive new features from recent work on fitness landscapes [18].

In addition to the issues associated with feature-extraction, a further challenge arises with respect to the need to deal with streaming data within combinatorial optimisation, particularly given the many practical applications that fall into this category (e.g. in scheduling and packing). A recent survey article describing the state-of-the-art in algorithm-selection [11] highlights both the necessity of developing ASP methods for learning in the context of streaming data and the difficulties associated with this. Specific challenges include the fact that the order of data points cannot be changed; the potentially large size of streams; the fact that data points can be evaluated once then are discarded; and that the underlying distribution of the data points in the stream can adapt over time. Only a few selection approaches have been proposed. This includes supervised-learning approaches [26, 27] and unsupervised-learning approaches such as stream-clustering which identify, track and update clusters over time [3, 7]. However, due to the huge space of parameter and algorithm combinations, clear guidelines on how to set and adjust them over time are lacking [2, 4, 16]. A recent paper addressed both the issue of feature-extraction and that of streaming data in proposing a deep-learning approach (RNN-LSTM) that learned from implicit temporal information present in the problem instances to predict the best performing heuristic [1]. This algorithm was "feature-free", i.e. it did not require the design and selection of features to describe an instance. The algorithm was shown to provide promising results in the domain of 1-d bin-packing.

The approach proposed in this paper goes beyond that of previous authors in further pursuing the goal of abandoning the need to derive features in describing a model that directly predicts a *solution* from the instance data rather than the algorithm that best solves it. It takes inspiration from the natural-language processing field in which an Encoder-Decoder LSTM model has been demonstrated to give state-of-the-art performance on tasks such as text translation [5, 25], image captioning [29], conversational modelling [28], learning to execute programs [32] and movement classification [9].

3 STREAMING DATA INSTANCES

We consider a streaming version of a 1-d bin-packing problem in which items arrive in fixed-size *batches.* Items have to be packed into a set of empty bins. We use a subset of the datasets that were recently described by [1] which have 120 items per instance that are initialised with item sizes drawn from two different distributions as shown in table 1. These instances are concatenated then split into batches with different window sizes. Note that these instances were specifically evolved by Alissa et al. [1] to maximise the difference between the fitness of the best-solving heuristic and the next best heuristic, therefore each subset of instances that are best-solved by a particular heuristic are likely to have similar characteristics. To generate solutions to these instances to use as training data for the model, we consider the 4 heuristics that were used by Alissa

[1]Long Short Term Memory

et al. [1] (BF, FF, NF, WF)[2] which pack one item per time-step in the order they arrive, plus 2 additional heuristics Best-Fit-Descending (BFD) and First-Fit-Descending (FFD) [15] that consider each batch in its entirety.

Table 1: Data set details. Bin Capacity is fixed at 150 [1]

DS	n_{items}	Lower - Upper Bounds	Distribution	total
DS1	120	[40-60]	Gaussian	4000
DS2	120	[20-100]	Uniform	4000

For the specific type of streaming problem considered, we split each instance into batches of size 6, 12 and 18 items per batch. Each batch is solved using each of *n* chosen heuristics (depending on the experiment). The solution with the best fitness according to the Falkenauer fitness [6] given in equation 1 is added to a training set.

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

For reference, table 2 shows the percentage of the solutions solved best by each heuristic under investigation on DS[1,2] with window sizes 6, 12 and 18. The table also highlights the *single-best-solver* (SBS), i.e the heuristic that 'wins' more solutions than any other heuristic in the set. BF and BFD are the SBS in DS1 and 2 respectively, regardless of batch size.

Table 2: Percentage of solutions solved best by each heuristic per dataset and per batch size. Values in bold show the single-best-solver in the related dataset

DS-#H-W.S.	H1	H2	H3	H4	H5	H6
	BF	FF	NF	WF	BFD	FFD
DS1-6H-6	**34.85%**	0.03%	0.14%	2.08%	62.89%	0.00%
DS1-6H-12	**39.77%**	0.33%	2.14%	7.75%	50.02%	0.00%
DS1-6H-18	**44.60%**	1.00%	3.28%	13.58%	37.54%	0.00%
DS2-6H-6	32.63%	1.21%	2.09%	1.08%	**62.98%**	0.02%
DS2-6H-12	12.95%	1.69%	2.87%	2.24%	**80.16%**	0.09%
DS2-6H-18	7.72%	1.91%	1.93%	2.89%	**85.48%**	0.07%

4 ENCODER-DECODER LSTM ARCHITECTURE FOR SOLUTION PREDICTION

This section describes the proposed model for providing a mapping from instance-data directly to a solution. Predicting a solution for a given problem instance is what is commonly known as a sequence-to-sequence problem (or seq2seq). These problems are challenging due to the fact that the number of items in the input and output sequences can be arbitrary and even from different spaces (e.g. in example statistical machine translation). However, Seq2Seq problems have been tackled in other domains using a deep-learning architecture known as an Encoder-Decoder LSTM [5, 33] that has proven very effective and achieved state-of-the-art performance.

[2]Best-Fit;First-Fit;Next-Fit;Worst-Fit [1]

An Encoder-Decoder LSTM architecture contains two models as shown in Fig 2. The *encoder* is used for mapping an input sequence into a vector of fixed dimensionality, and the *decoder* for outputting a predicted sequence based on the encoder output. The Encoder-Decoder LSTM uses a training technique known as *teacher forcing* [8]. This has been shown to train RNNs quickly and effectively where the decoder model receives the ground truth output $y_{(t)}$ as input at time t + 1 as shown in Fig 2. This approach was originally described and developed as an alternative technique to BackPropagation Through Time (BPTT) for training RNNs architecture that have lack hidden-to-hidden connections [8, 31]. A detailed explanation of the model is outside the scope of this paper. However, we provide a brief outline of how it works and how we have adapted for use as a solution predictor.

4.1 Training

Fig 2 provides a conceptual overview of how the Encoder-Decoder LSTM works. During training, the encoder produces a vector that represents the input sequence (window of items) to initialise the state of the decoder. Triggered by the encoder's state, the decoder produces each item's index in the output sequence in a one-shot manner (i.e. rather than recursively), as the entire solution (target sequence) is known during training. Training data is represented as a fixed-length list of item-sizes in a batch, e.g. [52, 57, 53, 53, 51, 55].

4.2 Testing

Similar to training, the encoder provides a fixed vector from the input. As shown in Fig 3, the main difference with testing phase is that the whole solution is not known beforehand, so the decoder will be called recursively for each item's index until the end of the solution. Using the encoder's state and the starting sequence value (SS), the decoder starts to predict the first item's index in the solution and feed the first predicted index with the first decoder unit's cell back to the next time step unit. This process will end when reaching the maximum size of solution. If the network predicts a solution before reaching the maximum size then it will predict the End Sequence (ES) value until reaching the max solution size. For a batch of size *n*, the network output is of the form shown below, which indicates which item in the batch is placed in which bin:

- Solution = [2, 6, **Sep**, 3, 4, **Sep**, 1, 5, **ES**]

The output is interpreted as follows: *place the 2nd and 6th items in bin 1, the 3rd and 4th items in bin 2, and the 1st and 5th items in bin 3.* A *separator* value is used to indicate breaks between bins. ES refers to the end of the sequence. Any values outside the range 1 to *n* (where *n* is the batch-size) could be chosen to represent the separator and ES. Here we select ES = 152 and Sep = 0; ES is a large value to make it out of the batch-size range, and the difference between the ES and Sep values is maximised to make it easier for the network to learn.

The process of prediction starts when the encoder initiates the decoder state with a vector representing the problem instance. Given the encoder state, alongside a trigger to indicate the start of a sequence (SS), the first decoder unit tries to predict the index of the first item in the solution, to provide the first decoder unit state. The first decoder unit state, along with the first predicted item in the solution then feed back into the second decoder unite to predict the second solution item, thus providing the second decoder unit state and so on. Note that once all items are packed, the network will predict ES until the end of the sequence is reached.

5 METHODOLOGY

We use the Keras functional API [3] implementation of an LSTM where the input is a list of item weights and the output is a solution. The input and output are both one-hot encoded [33]. As the predicted solutions vary in length due to the variable number of separator tokens used to delineate bins, we pad solutions to a fixed length by repeatedly appending the value End sequence ES = "152". Experiments are conducted on Google Colab[4] with GPU run-time used to execute the experiments. A preliminary empirical investigation was conducted to tune the Encoder-Decoder LSTM architecture and hyper-parameters using the ranges shown in table 3. The "Adam" optimiser [12] was used in all tuning experiments due to its reported accuracy, speed and low memory requirements. Note that further preliminary investigation considered various options for the representation of the output other than that described in the previous section. These included predicting the list of items per bin at each time step and predicting the size of the item to be packed per time step. However, based on these early experiments, it was clear that predicting the item's index per time step using separators to delineate bins provided the most promising results.

Each dataset was split into a training set (80%) and test set (20%). Training sets were created for batches of size (6,12,18). Each training set contains a list of input-output pairs in which the input is a list of item-sizes in the batch, and the output is a solution represented as a list of indices of items, with a separator delineating bins. For training, the solution associated with an input is the best solution found from greedily applying h heuristics, according to Falkenauer's fitness function. Experiments were conducted that varied both the size and composition of the set of H heuristics used to select a solution. All experiments were repeated five times. For each experiment, we save the model that provides the lower error from the training phase. We test the trained models on the test set, and additionally on two new random datasets RDS[1,2] each of which includes 800 instances created from the same properties of the original datasets, see Table 1.

To evaluate the results, we make use of the following metrics:

- **Accuracy:** The percentage of solutions that are exactly the same as the solutions in the test set.
- **Validity:** The percentage of solutions that are *valid* in the sense that (1) all items in the batch are represented exactly once in the solution and (2) no constraints regarding bin-capacity are broken.
- **Bilingual Evaluation Understudy (BLEU):** This metric is commonly used in the natural language processing field for evaluating the quality of translated text [19]. It measures the overlap between the predicted solution and the ground truth, returning a value between 0 and 1.
- **LSTM >BSS:** How many valid solutions are *better* than the Best Solver Solution (BSS) (in terms of the Falkenauer fitness metric). This metric provides insight into whether the model is 'creative' in the sense that it can produce novel, high-quality solutions not generated by the heuristics used in training.
- **LSTM <BSS:** How many valid solutions are worse than the Best Solver Solution (BSS) (in terms of the Falkenauer fitness metric).

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

6 RESULTS

This section describes the results to the questions posed in the introduction in turn.

6.1 Encoder-Decoder LSTM performance

The first experiment conducted uses the simplest possible training set in which only one heuristic is used to create solutions for a batch, i.e. the model only has to learn a mapping between instance and the solutions generated by a single specific heuristic. This is motivated by the idea that a given heuristic follows the same pattern in solving a set of instances, and therefore provides a simpler task for the model to learn. We use the BFD heuristic in these experiments as it wins the majority of instances in each dataset (table 2). Results are shown as experiment #1 in Tables 4 and 5.

The results show that Encoder-Decoder LSTM model can be used as a learner to predict an accurate solution for a given problem instance obtaining 98% and 88% accuracy for DS1 and DS2 respectively. One notices that the results from experiments on DS2 (Table 5) are lower than the ones on DS1 (Table 4). One reason might be that the DS2 instances are created using a wider range of item-size values [20, 100] compared to the DS1 instances which have item-sizes in the narrower range [40, 60]. This wider range makes the problem instances more difficult for the encoder to summarise and thus potentially the model requires more encoder layers.

99% and 95% of the predicted solutions are *valid* in DS1 and 2 respectively. On both datasets, the BLEU metric returns values in between the accuracy and validity values. Perhaps surprisingly, a very small number of 'creative' solutions are produced, i.e solutions which outperform the heuristic used to generate the training set; however, the models also produce some solutions that are lower in performance than the BSS.

6.2 Impact of Training with Diverse Heuristics

Experiments #2 to #5 in Tables 4 and 5 aim to study how the various metrics are influenced by the diversity of the heuristics used to create the training set, i.e. whether training on a dataset created using multiple diverse heuristics makes it harder or easier to learn an effective model. We consider 4 different heuristic sets:

- (BF, BFD): this set contains two heuristics that have very different characteristics (the former packing a single item in the order of appearance, the latter sorting the entire batch before packing).
- (BF, WF): these heuristics have similar characteristics, both packing a single item.
- (BF, FF, NF, WF): all the heuristics that pack a single item in order of appearance.
- (BF, FF, NF, WF, BFD, FFD): all heuristics.

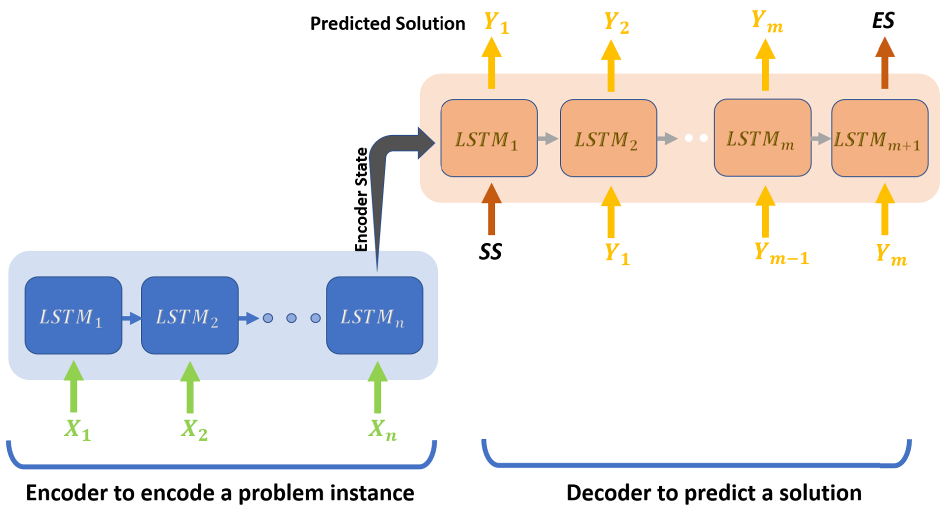

Figure 2: The training process of the Encoder-Decoder LSTM

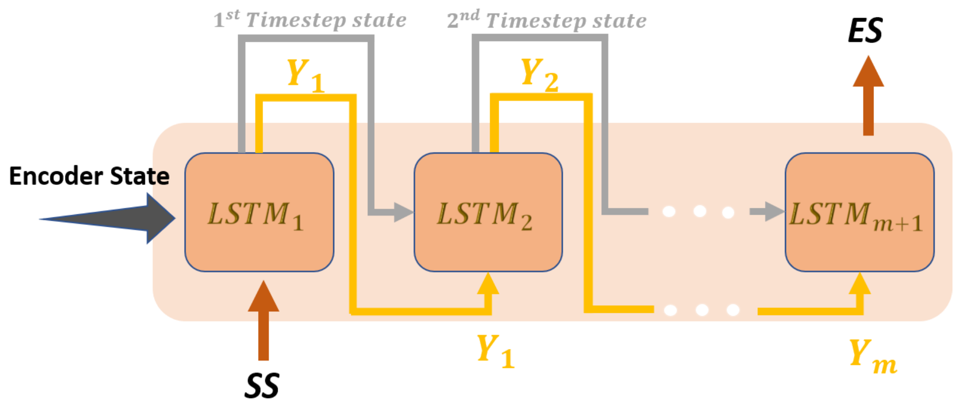

Figure 3: The testing process of the decoder

Table 3: Range of values that used in the Encoder-Decoder LSTM hyper-parameters tuning; the table also shows the final selected values

	#Epoch	Batch Size	#Layer	Memory Unites	Optimiser	LR	Loss Function
Range	[50,200]	[32, 2048]	[1_1 - 4_1, 2_2, 3_3]	[32, 2048]	adam	[0.0001, 0.001]	categ_crossentropy
Best	100	128	3_1 + Full-Connect Layer	1024	adam	0.001	categ_crossentropy

Note that as (BF, FF) produce very similar results we did not test this combination (with the same applying to (BFD, FFD)). Also, the combination (BF, NF) is not evaluated given that NF solves only a few of the instances.

We find that training with solutions created from diverse heuristics that have different characteristics in terms of the methods they use to create a solution reduces the ability of the learner to predict accurate and valid solutions (e.g. comparing experiment #2 to #1) — a learner with more memory units is likely required to cover this diversity. In contrast, solutions that are created using heuristics with similar characteristics (experiments #3,4) obtain similar results to experiment #1.

From the evaluation metrics perspective, we find that accuracy and validity appear correlated. The decrease in accuracy for experiment #2 (c.f. experiments #1,3,4) is accompanied by a small increase in the number of *new* solutions found that are better than any produced by the heuristics used to create the test data[5]. However, this

[5]Obviously a model that generates novel solutions that are better than those contained in the test set must have reduced accuracy as this metric measures the percentage of solutions that are identical to those in the test-set.

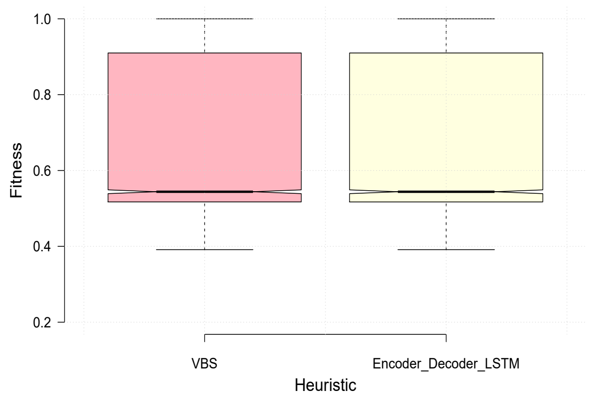

Figure 4: Evaluating Encoder-Decoder LSTM predictor VS VBS for DS1

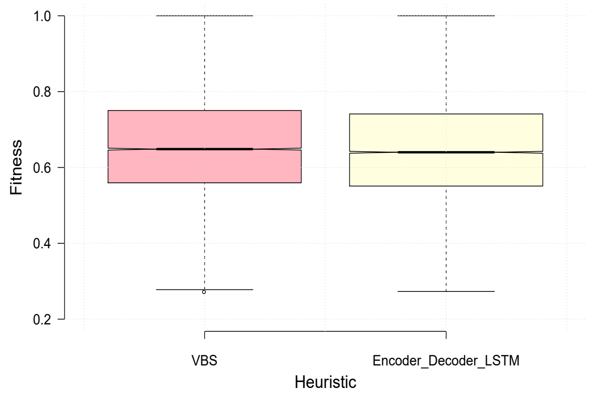

Figure 5: Evaluating Encoder-Decoder LSTM predictor VS VBS for DS2

also appears to be accompanied by a significant increase in the number of solutions obtained that are worse than the best solver used to create the test data.

Fig 4 and 5 show the performance distributions achieved by the VBS (i.e. the greedy selection of the solution created using the best heuristic for each batch) and by our Encoder-Decoder LSTM learner for all valid solutions on the test sets DS[1,2], using batch-sizes of 6 items and considering 4 heuristics (i.e. experiments #4). For these two experiments, our learner obtains very similar performance to the VBS. A paired t-test confirms that we cannot reject the null hypothesis that the VBS and Encoder-Decoder LSTM achieve the same results on DS1 (p-value = 0.11), while for DS2, the null hypothesis is rejected (p-value = 6.1 x 10^{-81}).

6.3 Effect of batch-size

Experiments #4, #6 and #7 in tables 4 and 5 aim to analyse the effect of the batch size on the solution prediction. We use batches of size 6, 12 and 18 items per instance[6]. These experiments are trained using solutions obtained from the set of 4 heuristics [1-4] given that this setting returned the best result in the previous experiments.

[6]These sizes are chosen as on average each bin can fit two or three items

The results show that the larger batch-size in both DS[1,2] reduces the ability of the learner to produce accurate and valid solutions: a more complex encoder(i.e. more layers) is required to address this. However, although the accuracy is decreased with the larger batches, we see a corresponding increase in the models' creativity, i.e. its ability to produce better solutions than the heuristics it was trained on. As in previous experiments, this is also accompanied by a significant increase in solutions which are worse than BSS however. Unlike in experiments #2 to #5, the Bleu metric has better values than accuracy and validity in experiments #6 and #7 which means although the solutions do not map exactly to those produced by the training heuristics, there is good overlap with the original solutions.

6.4 Generalisation to new instances

The final series of experiments investigates the ability of the trained models from the experiments in tables 4 and 5 to generalise to a new dataset of unseen instances. These instances are generated at random from a distribution with the same parameters as DS1, DS2 respectively. However, recall that the instances in DS1, DS2 were specifically evolved to maximise the difference between the fitness of the best-solving heuristic and the next best heuristic, so are likely to have particular characteristics that are not apparent when a random ordering of items is used to generate the instance.

Tables 6 and 7 show the results of testing the previously trained models on new unseen instances. Experiments #1 to #6 in these tables show that the models trained on the non-random datasets DS1 and DS2 have excellent ability to generalise over the new problem instances RDS1 and RDS2, obtaining results that only deviate by a maximum of 2% in terms of accuracy when compared to the previous results presented in tables 4 and 5. Experiments #7 and #8 using the larger batch-size show accuracies that are reduced by up to 7% in comparison to the non-random datasets, although we note that these models performed relatively poorly on the original datasets.

7 CONCLUSION

In order to tackle the Algorithm Selection Problem in the context of streaming data, we have proposed a radical solution which uses a deep-learning model to directly predict a solution from instance-data. This bypasses the typical processes of ASP first defined by Rice [22] in which features need to be derived from instances and mapped to an algorithm-space via a learner of some kind.

In order to show the potential benefits of pursuing this line of research more deeply, we have evaluated the approach on a simple type of streaming problem in the packing domain in which items arrive in batches, and each batch must be packed before moving to the next. We have provided the first description of a model that is capable of predicting a solution and then studied two questions in depth. The first concerns the influence of batch-size on performance, while the second aims to understand whether the diversity of the heuristics used to create the training data influences performance.

Experiments showed that the proposed approach is able to predict very accurate solutions with smaller batches, particularly using solutions produced by heuristics with similar characteristics. Furthermore, and perhaps surprisingly, all of the trained models were

Table 4: The mean and std results of repeated five times the approach on DS1 (120-N-40-60) using different combinations of the heuristics H[1-6]

#	Window size	#Heuristics	Heuristics	Acc	Valid	BLEU	LSTM >BSS	LSTM <BSS
1	6	1	H5	98.08% (+/- 0.24%)	99.08% (+/- 0.16%)	98.53% (+/- 0.18%)	1 (+/- 2)	149 (+/- 20)
2	6	2	H[1, 5]	91.41% (+/- 0.40%)	95.89% (+/- 0.56%)	92.77% (+/- 0.37%)	8 (+/- 3)	578 (+/- 77)
3	6	2	H[1, 4]	98.87% (+/- 0.75%)	99.23% (+/- 0.75%)	99.20% (+/- 0.55%)	3 (+/- 3)	37 (+/- 12)
4	6	4	H[1-4]	98.76% (+/- 0.94%)	99.26% (+/- 0.77%)	99.13% (+/- 0.64%)	3 (+/- 1)	60 (+/- 38)
5	6	6	H[1-6]	91.32% (+/- 0.79%)	95.85% (+/- 0.62%)	92.55% (+/- 0.67%)	9 (+/- 3)	587 (+/- 43)
6	12	4	H[1-4]	62.76% (+/- 1.11%)	81.50% (+/- 0.69%)	79.75% (+/- 0.66%)	63 (+/-18)	1341 (+/- 42)
7	18	4	H[1-4]	27.28% (+/- 1.13%)	47.59% (+/- 0.09%)	60.45% (+/- 0.55%)	50 (+/-6)	885 (+/- 53)

Table 5: The mean and std results of repeated five times the approach on DS2 (120-U-20-100) using different combinations of the heuristics H[1-6]

#	Window size	#Heuristics	Heuristics	Acc	Valid	BLEU	LSTM >BSS	LSTM <BSS
1	6	1	H5	88.42% (+/- 0.29%)	94.90% (+/- 0.34%)	91.95% (+/- 0.16%)	7 (+/- 4)	968 (+/- 51)
2	6	2	H[1, 5]	68.06% (+/- 0.41%)	87.42% (+/- 0.71%)	77.24% (+/- 0.37%)	32 (+/- 5)	2526 (+/-134)
3	6	2	H[1, 4]	85.18% (+/- 0.10%)	92.67% (+/- 0.21%)	89.63% (+/- 0.01%)	41 (+/-11)	1091 (+/- 39)
4	6	4	H[1-4]	84.87% (+/- 0.23%)	93.20% (+/- 0.84%)	89.39% (+/- 0.07%)	26 (+/- 5)	1211 (+/- 105)
5	6	6	H[1-6]	65.81% (+/- 0.48%)	86.60% (+/- 0.67%)	75.58% (+/- 0.35%)	28 (+/- 4)	2746 (+/- 63)
6	12	4	H[1-4]	27.14% (+/- 0.80%)	55.86% (+/- 3.01%)	59.91% (+/- 0.64%)	114 (+/- 17)	2120 (+/- 164)
7	18	4	H[1-4]	4.32% (+/- 0.59%)	20.42% (+/- 2.42%)	41.87% (+/- 0.97%)	32 (+/- 4)	733 (+/- 87)

able to produce a small number of novel solutions that had better performance according to the Falkenauer fitness metric than any of the heuristics used to train the system.

Clearly, the work presented here only represents the first steps into this new paradigm. Further work is required to modify the approach to deal with more general versions of streaming problems, and to evaluate its potential when trained on solutions generated using a greater variety of optimisation algorithms, including meta-heuristic as well as hyper-heuristic approaches. Feeding dynamic information about the current state of partial solutions into the networks is also likely to be of benefit. Despite this, we suggest that the method provides a promising new direction for research within ASP that is ripe for further exploration.

Table 6: The mean and std results of the trained models on DS1 (120-N-40-60-*Random*) using different combinations of the heuristics H[1-6]

#	Window size	#Heuristics	Heuristics	Acc	Valid	BLEU	LSTM >BSS	LSTM <BSS
1	6	1	H5	97.33% (+/- 0.25%)	98.69% (+/- 0.19%)	97.91% (+/- 0.17%)	1 (+/- 1)	199 (+/- 16)
2	6	2	H[1, 5]	89.08% (+/- 0.61%)	94.47% (+/- 0.82%)	90.78% (+/- 0.49%)	5 (+/- 4)	684 (+/- 83)
3	6	2	H[1, 4]	98.75% (+/- 0.74%)	99.21% (+/- 0.76%)	99.09% (+/- 0.54%)	9 (+/-3)	46 (+/- 5)
4	6	4	H[1-4]	98.74% (+/- 0.76%)	99.27% (+/- 0.69%)	99.10% (+/- 0.51%)	6 (+/- 3)	60 (+/- 21)
5	6	6	H[1-6]	88.98% (+/- 0.84%)	94.49% (+/- 0.62%)	90.56% (+/- 0.72%)	5 (+/- 3)	700 (+/- 79)
6	12	4	H[1-4]	61.16% (+/- 0.80%)	78.51% (+/- 0.65%)	79.26% (+/- 0.53%)	60 (+/- 7)	1253 (+/- 34)
7	18	4	H[1-4]	23.52% (+/- 1.39%)	42.05% (+/- 1.32%)	59.46% (+/- 0.84%)	52 (+/- 5)	811 (+/- 7)

Table 7: The mean and std results of the trained models on DS2 (120-U-20-100-*Random*) using different combinations of the heuristics H[1-6]

#	Window size	#Heuristics	Heuristics	Acc	Valid	BLEU	LSTM >BSS	LSTM <BSS
1	6	1	H5	89.24% (+/- 0.56%)	94.81% (+/- 0.57%)	92.70% (+/- 0.31%)	7 (+/- 2)	834 (+/- 51)
2	6	2	H[1, 5]	66.76% (+/- 0.44%)	85.77% (+/- 0.75%)	76.81% (+/- 0.39%)	30 (+/- 5)	2505 (+/-69)
3	6	2	H[1, 4]	84.80% (+/- 0.18%)	91.92% (+/- 0.40%)	89.74% (+/- 0.05%)	65 (+/- 9)	1029 (+/- 100)
4	6	4	H[1-4]	84.04% (+/- 0.56%)	91.64% (+/- 0.95%)	89.11% (+/- 0.37%)	33 (+/- 2)	1115 (+/- 66)
5	6	6	H[1-6]	63.69% (+/- 0.57%)	84.74% (+/- 0.57%)	74.57% (+/- 0.48%)	28 (+/- 6)	2811 (+/- 89)
6	12	4	H[1-4]	20.26% (+/- 1.41%)	47.75% (+/- 4.02%)	55.93% (+/- 0.88%)	113 (+/- 14)	2065 (+/- 200)
7	18	4	H[1-4]	2.58% (+/- 0.36%)	13.97% (+/- 2.08%)	39.69% (+/- 1.00%)	24 (+/- 4)	518 (+/- 81)

REFERENCES

[1] Mohamad Alissa, Kevin Sim, and Emma Hart. 2019. Algorithm selection using deep learning without feature extraction. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 198–206.

[2] Amineh Amini, Teh Ying Wah, and Hadi Saboohi. 2014. On density-based data streams clustering algorithms: A survey. *Journal of Computer Science and Technology* 29, 1 (2014), 116–141.

[3] Matthias Carnein, Dennis Assenmacher, and Heike Trautmann. 2017. An empirical comparison of stream clustering algorithms. In *Proceedings of the Computing Frontiers Conference*. ACM, 361–366.

[4] Matthias Carnein and Heike Trautmann. 2019. Optimizing data stream representation: An extensive survey on stream clustering algorithms. *Business & Information Systems Engineering* (2019), 1–21.

[5] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078* (2014).

[6] 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.

[7] Shufeng Gong, Yanfeng Zhang, and Ge Yu. 2017. Clustering stream data by exploring the evolution of density mountain. *Proceedings of the VLDB Endowment* 11, 4 (2017), 393–405.

[8] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press.

[9] Félix G Harvey and Christopher Pal. 2015. Semi-supervised learning with encoder-decoder recurrent neural networks: Experiments with motion capture sequences. *arXiv preprint arXiv:1511.06653* (2015).

[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] Pascal Kerschke and Heike Trautmann. 2019. Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning. *Evolutionary computation* 27, 1 (2019), 99–127.

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

[13] Chan C Lee and Der-Tsai Lee. 1985. A simple on-line bin-packing algorithm. *Journal of the ACM (JACM)* 32, 3 (1985), 562–572.

[14] Ming Liu, Yinfeng Xu, Chengbin Chu, and Feifeng Zheng. 2009. Online scheduling on two uniform machines to minimize the makespan. *Theoretical Computer Science* 410, 21-23 (2009), 2099–2109.

[15] 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.

[16] Stratos Mansalis, Eirini Ntoutsi, Nikos Pelekis, and Yannis Theodoridis. 2018. An evaluation of data stream clustering algorithms. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 11, 4 (2018), 167–187.

[17] 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.

[18] Gabriela Ochoa and Nadarajen Veerapen. 2018. Mapping the global structure of TSP fitness landscapes. *Journal of Heuristics* 24, 3 (2018), 265–294.

[19] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*. Association for Computational Linguistics, 311–318.

[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] Prakash Ramanan, Donna J Brown, Chung-Chieh Lee, and Der-Tsai Lee. 1989. On-line bin packing in linear time. *Journal of Algorithms* 10, 3 (1989), 305–326.

[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] 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

[24] Thomas Stützle and Manuel López-Ibáñez. 2019. Automated design of metaheuristic algorithms. In *Handbook of metaheuristics*. Springer, 541–579.

[25] I Sutskever, O Vinyals, and QV Le. 2014. Sequence to sequence learning with neural networks. *Advances in NIPS* (2014).

[26] Jan N van Rijn, Geoffrey Holmes, Bernhard Pfahringer, and Joaquin Vanschoren. 2014. Algorithm selection on data streams. In *International Conference on Discovery Science*. Springer, 325–336.

[27] Jan N van Rijn, Geoffrey Holmes, Bernhard Pfahringer, and Joaquin Vanschoren. 2018. The online performance estimation framework: heterogeneous ensemble learning for data streams. *Machine Learning* 107, 1 (2018), 149–176.

[28] Oriol Vinyals and Quoc Le. 2015. A neural conversational model. *arXiv preprint arXiv:1506.05869* (2015).

[29] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. 2015. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3156–3164.

[30] Gary R Weckman, Chandrasekhar V Ganduri, and David A Koonce. 2008. A neural network job-shop scheduler. *Journal of Intelligent Manufacturing* 19, 2 (2008), 191–201.

[31] Ronald J Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural computation* 1, 2 (1989), 270–280.

[32] Wojciech Zaremba and Ilya Sutskever. 2014. Learning to execute. *arXiv preprint arXiv:1410.4615* (2014).

[33] Aston Zhang, Zachary C Lipton, Mu Li, and Alexander J Smola. 2019. Dive into Deep Learning. *Unpublished draft. Retrieved* 3 (2019), 319.