Using Deep Learning to Select Optimal 2-D Bin Packing Heuristics

Team Boo!

Akaash Sanyal
Johns Hopkins University
asanyal5@jhu.edu

Teya Bergamaschi
Johns Hopkins University
tbergam1@jhu.edu

Lori Baxter
Johns Hopkins University
lbaxter6@jhu.edu

May 1, 2020

Abstract

Our project explores a deep learning approach to optimizing the Bin Packing Problem (BPP), specifically focusing on the 2-D, offline, single fixed bin size variation of the problem. We have created a modified approach to this NP-hard problem that selects the optimal heuristic from a set of efficient bin packing heuristics. Using a pseudo-randomly generated data set with 75/25 training/testing split, we explored 3 bin algorithms and 3 packing algorithms. In order to increase efficiency, we chose to extract a choice number of relevant features from our data based on the properties of the boxes. Ultimately, we built an algorithm that had a custom testing accuracy of 81.73% with a top 3 testing accuracy of 83.61%. These results lend well to future analyses, including a multi-label approach or other variations of the BPP.

1 Introduction

The Bin Packing Problem (BPP) is a classic NP-hard problem in combinatorial optimization and can take many forms. Common BPPs can be one, two, or three dimensional and can involve different packing goals including fitting a finite number of items into one minimally sized bin (often with one dimension of freedom and one or two fixed dimensions) or fitting a number of items into a minimum number of larger bins [1]. For this project, we focused on an offline variety of BPP that involves fitting an arrangement of rigid rectangular items inside larger rectangular bins with a single fixed capacity while minimizing the number of bins used. Specifically, we considered the instance when you had a single fixed bin size and a known set of boxes whose sizes were less than the size of the bin and which you could order in any fashion before beginning to pack them. Additionally, we allowed for free rotation of the boxes during packing.

This case of regular rectangular items is one of the most common variations of the BPP and has many practical applications in common domains, rendering it one of the most studied and commonly explored optimization problems [2]. For example, domains such as logistics transportation, scheduling, manufacturing, and network optimization all utilize forms of Bin Packing [1].

Given the NP-hard nature of the problem, finding an exact solution to the BPP for larger data sets is very difficult; as such, several efforts to develop approximation algorithms have been made, leading to the creation of packing heuristics. None of these heuristics are universally better than the others and perform best on certain kinds of instances. Therefore, selecting the optimal heuristic used to solve a BPP instance is a challenging problem in its own right. A deep learning solution to this selection allows for a more optimal and efficient solution to many BPP instances than selecting a single heuristic for all BPP instances or trying many heuristics on a single BPP instance to find the most optimal solution [3]. Deep learning poses its own series of obstacles such as training and testing large sets of data, but ultimately has the potential to yield more reliable and accurate results [1]. Our deep learning approach necessitates a large data set, a certain degree of feature engineering, and creation of a multilayer neural network, but should improve the selection of a heuristic when tackling the BPP.

2 Related Work

To build our research base for our project, we surveyed three papers each with a unique approach to Bin Packing. Each paper explores different heuristic approaches to BPP, a variety of computational methodologies, and a wide range of solutions.

In their paper, *Small Boxes Big Data: A Deep Learning Approach to Optimize Variable Sized Bin Packing*, Mao et al. explore a similar problem to our own project: a deep learning solution to BPP. Their paper reviewed approximate solutions to the 1-D variable sized BPP. Using bin assignment and bin allocation optimization heuristics, Mao et al. built a static optimization heuristic that works similarly to a dynamic one by employing deep learning methods. Their deep learning approach overcomes drawbacks found in machine learning (such as the curse of dimensionality) by applying large training data sets, auto feature selection, and fast and accurate labeling. Their model used large raw data sets and fast label procedure to create a predictive system that achieved up to 89% training accuracy and 72% validation accuracy [1].

This paper has multiple applications to our own work. Mao et al. set the foundation for our deep learning work by walking through the structure of an effective BPP neural network [1]. We have expanded many of the ideas and methods from their paper to apply to our 2-D deep learning BPP project and have used many of their results to guide our own efforts.

Hu et al. also approach Bin Packing using deep learning in their paper *Solving a New 3D Bin Packing Problem with Deep Reinforcement Learning Method*. Their problem is structured around the real life applications of a variation of BPP in which a single, variable sized bin is constructed around a optimum packing arrangement. They approach this "New 3D BPP" with the goal of minimizing the cost of the total size of a bin, with its cost proportional to the bin's surface area. A multitude of different obstacles arise from this variation of BPP, including properly sequencing the items being packed and properly orienting items within the bin spaces. Using DRL techniques such as Pointer Networks, Hu et al. build a DRL-based method that significantly outperforms a heuristic algorithm [3]. Though our BPP problem was completely different than the one explored in *Solving a New 3D BPP*, the DRL approaches provided an interesting exploration of different applications of deep learning and the focus on packing order informed some of our early exploration into heuristics to use for our model.

A Thousand Ways to Pack the Bin- A Practical Approach to Two-Dimensional Rectangle Bin Packing by Jylänki adds to our wealth of potential heuristics by surveying several commonly used algorithms for solving the 2-D BPP. Though primarily a summary of these approaches, Jylänki's findings offer insight into both the methodology behind common BPP solutions and the comparative effectiveness of these approaches [2]. By walking through numerous ways to solve the BPP, we were able to make an informed decision in choosing our own heuristics and methods of solving a 2-D BPP.

Other related work for our project includes *Optimal Rectangle Packing: An Absolute Placement Approach* by Huang and Korf, which considers a BPP variation that finds all enclosing rectangles of minimum area that contain a given set of rectangles without overlap [4].

3 Methodology

3.1 Data Sets

Our data set was an entirely pseudo-randomly generated data set consisting of 40,000 instances, with a 75/25 training/testing split. Each instance consisted of a fixed rectangular 2D bin size between 1 by 1 and 40 by 40 and between 2 and 1000 boxes whose sizes were pseudo-randomly generated such that they were equal to or smaller than the bin size but at least size 1 by 1. All bin and box lengths were integers.

3.2 Prediction Model

Our prediction model is given a set of inputs X and labels Y and estimates a prediction function $\hat{Y} = F(X)$. The distance between these predictions and true labels is measured by a loss function, $L(Y, \hat{Y})$, which is minimized to improve the function.

3.2.1 Feature Extraction

Since each instance is variable in size (number of boxes) and loading a large number of instances requires a lot of memory, we chose to extract relevant features to feed into our model to increase efficiency, rather than have our model run on raw data. We extracted features for each instance based on properties of boxes, properties of bins, and the interaction of the two. These features are then treated as the aforementioned inputs, X. In total we extracted around 60 features, which are listed below in Table 3.2.1.

Feature Type	Features
Item Features	Number of items
	Item area*
	Perimeter*
	Length of short side*
	Length of long side*
	Length of a side*
	Ratio of short to long side*
Bin Features	Bin area
	Bin perimeter
	Length of short side
	Length of long side
	Ratio of short to long side
Cross Features	Item to bin area ratio*
	Item to bin perimeter ratio*
	Item to bin length of short side ratio*
	Item to bin length of long side ratio*

Table 3.2.1: Features

* Six individual features were extracted from each of these: the sum, minimum, maximum, average, standard deviation, and variance.

3.2.2 Heuristic Choices

We considered three bin algorithms and three packing algorithms. The three bin algorithms we considered were found in *A comparison of next-fit, first-fit, and best-fit* and are as follows [5]:

1. Bin First Fit: Pack the box into the first bin it fits into, leaving all bins open.
2. Bin Next Fit: Perform Pack the box into the first bin it fits into, closing every bin encountered that the box cannot fit in to.
3. Bin Best Fit: Search all open bins and pack the box into the bin that gives best fit.

The three packing algorithms we considered were found in *A Thousand Ways to Pack the Bin- A Practical Approach to Two-Dimensional Rectangle Bin Packing* and are as follows [2]:

1. Maximal Rectangles Best Short Side Fit (BSSF): All empty areas in a bin are divided into maximally sized rectangles. The next box is packed into a rectangle such that the lesser difference between box and rectangle side length is minimized.
2. Maximal Rectangles Best Area Fit (BAF): All empty areas in a bin are divided into maximally sized rectangles. The rectangle with the smallest area is then packed. If there is a tie in area, the BSSF rule is used to break it.
3. Maximal Rectangles Best Long Side Fit (BLSF): All empty areas in a bin are divided into maximally sized rectangles. The next box is packed into a rectangle such that the greater difference between box and rectangle side length is minimized.

We considered a total of nine heuristics through the combination of the three bin algorithms and three packing algorithms, found in Table 3.2.2.

Bin Algo\Packing Algo	BSSF	BAF	BLSF
Bin Next Fit	0	1	2
Bin First Fit	3	4	5
Bin Best Fit	6	7	8

Table 3.2.2: Heuristic Choices

Throughout all of our heuristics, we allowed the rotation of boxes and ordered the boxes based on the ratio of the short side to the long side prior to running.

3.2.3 Label Space

In order to generate the labels for an instance, we pack it with all of our selected heuristics to make a list of the number of bins needed for each. As our aim is to identify the best

heuristic in terms of lowest number of bins, we choose the index of the heuristic that had the lowest number of bins. However, we often found that several heuristics produced the relatively best packing; since our prediction model was based on the assumption that labels are mutually exclusive, we would randomly select one of them to be considered the best heuristic by our model. This best heuristic was one-hot encoded as the label and these labels serve as our Y, as mentioned in section 3.2. In our final model, we elected to make this choice between the best heuristics randomly instead of in some order (such as lowest index or alphabetically) to limit any potential bias this would create. Originally, we attempted to pick the lowest indexed best heuristic, but this biased the model towards the lower indexed heuristics and deteriorated performance.

3.2.4 Neural Network

In our L layer neural network, each layer $l \in \{0, 1, \ldots, L-1\}$ is described by a $m \times n$ weight matrix W_l and an activation function $f_{a(l)}$. The weight matrix transforms n inputs to m outputs (where n is the number of nodes in layer l and m is the number of nodes in layer $l+1$) and the activation function transforms these m outputs to a single scalar value. As a result, the entire network can be described as follows:

$$Y^{(0)} = X = [x_1, x_2, \ldots, x_n],\ Y^{(l)} = [Y_1^{(l)}, Y_2^{(l)}, \ldots, Y_m^{(l)}],\ Y_i^{(l)} = f_{a(l)}(W_l \cdot Y^{(l-1)})$$

Note the special case for the first input layer when $l = 0$. The final layer uses an activation $f_a(x) = softmax(x) = \frac{exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}$. The loss function is the categorical cross entropy, defined by

$$L(Y, \hat{Y}) = \sum_{k=1}^{K} (\hat{y_k} \log(y_k) + (1 - \hat{y_k}) log(1 - y_k))$$

We added batch normalization at each layer to counteract a problem known as internal covariance shift [6]. We also worked to prevent overfitting by adding random drops at each layer. This results in several models generated by randomly dropping out internal nodes from the original network, and the final prediction averages the predictions of these smaller models [7].

In order to identify the best model, we tuned the model with varying architectures and hyperparameters using the hyperas package [8]. Our neural network is guaranteed to have at least one hidden layer with $n \in \{32, 64, 128\}$ nodes and a dropout probability p_1. From here, the network can have $h \in 1, 2, 3$ other hidden layers, each of which are identical with $n_h \in \{32, 64, 128\}$ nodes and a dropout probability p_h. Each of these layers uses $f_a \in \{softplus, relu\}$, where $softplus(x) = \ln(1 + \exp(x))$ and $relu(x) = \max(0, x)$.

Each of the above models were trained for 50 epochs with a 20% validation split using the Adam optimizer and a batch size of 128. As another hyperparameter in tuning, the learning rate for the optimizer was $lr \in \{0.1, 0.01, 0.001\}$. In our hyperparameter search, we used 400 different combinations.

A full table of our hyperparameter search space can be found below in Table 3.2.3.

Hyperparameter	Search Space	Description
n	$\{32, 64, 128\}$	Nodes in first hidden layer
p_1	$[0, 1]$	Dropout probability of first hidden layer
h	$1, 2, 3$	Number of extra hidden layers
p_h	$[0, 1]$	Dropout probability of extra hidden layers
f_a	$\{softplus, relu\}$	Activation function
lr	$\{0.1, 0.01, 0.001\}$	Learning rate

Table 3.2.3: Hyperparameters and Descriptions

3.2.5 Evaluation

When training the neural net, our loss function penalized heuristic predictions that did not exactly match with the potentially randomly chosen label. However, we wanted to use a more informative metric for our context. A correct prediction no longer required an exact match of the label; if the predicted heuristic used the same number of bins as the label heuristic, the prediction was considered correct. To evaluate performance and choose the best model when tuning, tried to maximize this custom metric; this was not used in the actual training, but rather used to make decisions between models.

Another evaluation metric that we were interested in was the top 3 categorical accuracy. This metric looked at three most likely labels predicted by the neural net and counted a correct prediction if the true label was in these top three. The drawback of this metric in our case comes from the random choice in picking a label. In cases that more than three heuristics performed the same, the neural net could have choose three relatively optimal heuristics that do not include the label, and thus it would underestimate our desired performance.

3.2.6 Implementation

All heuristics were implemented with the rectpack library on GitHub [9]. All of our code can be found at https://github.com/akaashsanyal/2DBPP-HeuristicNet.

4 Computational Results

After tuning our neural network, we found the architecture that gave the highest performance on the validation set. The chosen hyperparameters are below in Table 4.1.

Hyperparameter	Search Space
n	32
p_1	004697373821486008
h	2
p_h	0.02213872171606842
f_a	$relu$
lr	0.01

Table 4.1: Hyperparameter Values

We reconstructed our neural network with these new hyperparameters and retrained it with the training data for 100 epochs, keeping track of the training and validation top 3 accuracies, which can be seen below in Figure 4.1.

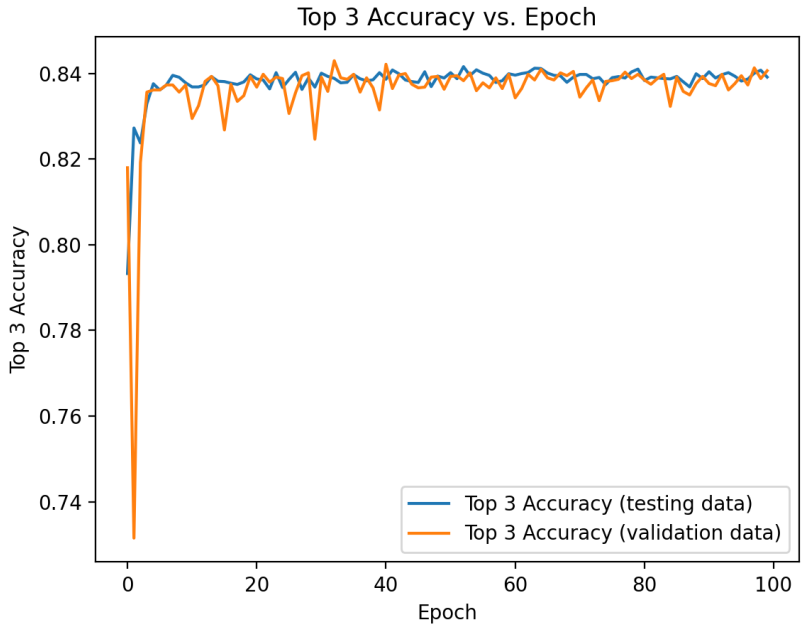

Figure 4.1: Top 3 Accuracy

Once the final model was trained, we evaluated its performance on the testing set. Our neural network had a custom testing accuracy of 81.73% with a top 3 testing accuracy of 83.61%. We also wanted to evaluate the performance of our neural network in comparison to the heuristic choices. We calculated the mean number of bins needed for each heuristic and

predicted by our model, and we compared this to the best choice of heuristic for each given instance. These results can be found in Table 4.2, which demonstrate that our neural net performed better than all of the individual heuristics. The closest heuristic was Heuristic 6, which also performed best in *A Thousand Ways to Pack the Bin- A Practical Approach to Two-Dimensional Rectangle Bin Packing* [2].

Solving Choice	Mean # of Bins	Proportion to Best
Heuristic 2	233.8992	1.3274
Heuristic 1	229.0065	1.2996
Heuristic 0	227.0244	1.2884
Heuristic 8	182.1862	1.0339
Heuristic 5	182.086	1.0334
Heuristic 4	179.3072	1.0176
Heuristic 7	178.6351	1.0138
Heuristic 3	177.0074	1.0045
Heuristic 6	176.5397	1.0019
Neural Net	176.5068	1.0017
Best Choice	**176.2076**	**1.0000**

Table 4.2: Heuristic Performance

5 Conclusion and Future Work

One major limitation of our approach is that it is bounded by the heuristic approximations. This method will never be able to outperform the best performing heuristic for any one instance. While this approach can outperform many single heuristics on a large set of problems, it makes no further guarantee than any heuristic as to its performance compared to the true minimum.

To make the results of our model more meaningful, the difference between the best heuristic and best choice would need to be larger. Currently, the best heuristic is close enough to the best choice that the average bins used is less than one bin different. This means that there is no room for meaningful improvement by our neural net; although our neural net is better than the best heuristic, there is no real world meaning to this improvement. This likely stems from a single heuristic performing best for the majority of scenarios. However, the heuristics considered here represent just a small subset of possible heuristics. The application of our neural net would likely be more meaningful if a wider selection of heuristics was considered so there would not be a single dominating heuristic. An easy and intuitive way to start would be to explore packing more order/sorting algorithms. These, coupled with the heuristics we already consider, would broaden the scope of heuristics considered.

Further work on this approximation method would likely employ a multi-label approach. Our current method assumes mutually exclusive classes and thus employs a categorical cross-entropy loss function with a final softmax activation; the multi-labeling approach uses a binary cross-entropy loss function for each label and a final thresholded sigmoid activation layer to allow for multiple labels.

Lastly, our method could easily be used to study other BPP problems through minor changes and the substitution of the heuristics considered.

References

[1] Mao, Feng, et al. “Small Boxes Big Data: A Deep Learning Approach to Optimize Variable Sized Bin Packing.” 2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService), 2017, doi:10.1109/bigdataservice.2017.18.

[2] Jylänki, Jukka. “A Thousand Ways to Pack the Bin - A Practical Approach to Two-Dimensional Rectangle Bin Packing.” 27 Feb. 2010.

[3] Hu, Haoyuan, et al. “Solving a New 3D Bin Packing Problem with Deep Reinforcement Learning Method.” ArXiv, 2017, abs 1708.05930.

[4] Huang, E., and R. E. Korf. “Optimal Rectangle Packing: An Absolute Placement Approach.” Journal of Artificial Intelligence Research, vol. 46, 2013, pp. 47–87., doi:10.1613/jair.3735.

[5] Bays, Carter. “A Comparison of next-Fit, First-Fit, and Best-Fit.” Communications of the ACM, vol. 20, no. 3, 1977, pp. 191–192., doi:10.1145/359436.359453.

[6] Ioffe, Sergey, and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift.” PRoceedings of the 32nd International Conference on International Conference on Machine Learning, vol. 37, 2015, pp. 448–456.

[7] Srivastava, Nitish, et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” Journal of Machine Learning Research, vol. 15, 2014, pp. 1929–1958.

[8] M. Pumperla & P. Kainz & J. Mackenzie & A. Mallick, ”Hyperas,” Github repository: https://github.com/maxpumperla/hyperas

[9] SecNot & S. Denton & J. Kim & M. Torres, ”Rectpack,” Github repository: https://github.com/secnot/rectpack