Reinforcement Learning for Traveling Salesman and Bin-Packing Problems

Brian Tan
Department of Computer Science
Johns Hopkins University
Baltimore, MD 21218
ktan7@jhu.edu

Abstract

Deep learning, specifically reinforcement learning has gained significant attention in solving combinatorial optimization problems recently. In this paper, we implement a framework with neural networks and reinforcement learning to tackle combinatorial optimization problems. We focus on the traveling salesman problem (TSP) and train a recurrent neural network that, given a set of city coordinates, predicts a distribution over different city permutations. Using negative tour length as the reward signal, we optimize the parameters of the recurrent neural network with the policy gradient method. We compare learning the network parameters on a set of training graphs against learning them on individual test graphs. We test our implementation to solve TSP on 2D Euclidean graphs with 10 and 20 nodes. Additionally, we also apply the reinforcement learning method to solve the bin-packing, another NP-hard problem.

1 Introduction

Combinatorial optimization is a fundamental problem in operations research, applied mathematics, and theoretical computer science. The traveling salesman problem (TSP) is one of the most well-studied problems in combinatorial optimization. The problem was first formulated in 1930, where the solver needs to search the space of permutations to find an optimal sequence of nodes with minimal total tour length. The TSP has several applications even in its basic formulation, such as planning, logistics, and the manufacture of microchips. TSP can even solve DNA sequencing problems just with slight modification. The TSP also appears in astronomy, as astronomers observing many sources will want to minimize the time spent moving the telescope between the sources.

Finding the optimal TSP solution is NP-hard, even in the two-dimensional Euclidean case [13], where the nodes are 2D points and edge weights are Euclidean distances between pairs of points. In practice, TSP solvers rely on handcrafted heuristics that guide their search procedures to find competitive (and in many cases optimal) tours efficiently. However, one problem with these heuristics is that they are very problem dependent. In other words, the heuristics need to be revised even with minor changes in the problem. To overcome this weakness, machine learning methods are widely studied to solve TSP as they have the potential to generalize over many optimization tasks by automatically discovering their own heuristics based on the training data. The machine learning methods provide a more principal way to tackle TSP as they require less hand-crafted heuristics from humans.

Currently, most of the successful machine learning algorithms are in the category of supervised learning, where the model learns a mapping from training inputs to outputs. Supervised learning is not applicable to most combinatorial optimization problems because the optimal labels are normally unavailable. Hence, it is desired to find machine learning algorithms that work independently from the quality of labels or no labels involved in the training at all. Consider the fact that one can check the quality of a set of solutions and provide some reward feedbacks to a learning algorithm. Reinforcement

learning (RL) [17, 11, 12] paradigm becomes a good candidate to solve the combinatorial optimization problem. RL introduces an agent that takes actions at each state to maximize the cumulative rewards without explicitly labeled input/output pairs. In this project, we combined reinforcement learning and pointer networks, an enhanced neural networks,to solve traveling salesman problem, and applied reinforcement learning on heuristic selection model to tackle bin-packing problem.

2 Related Work

Many exact or approximate algorithms have been proposed in the past for TSP on both Euclidean and non-Euclidean graphs. Christofides proposes a heuristic algorithm that involves computing a minimum-spanning tree and a minimum-weight perfect matching. The algorithm has polynomial running time and it guarantees that the solution is better than 1.5 times to optimality in the metric instance of the TSP. [20] proposed a genetic algorithm-based solution for TSP where all points are on the surface of a sphere. By adopting the genetic algorithms and 2-opt, [20] finds geodesics (shortest distances) between all pairs of cities on the surface of the sphere effectively.

Many algorithms for TSP, such as exact dynamic programming algorithm, are infeasible to scale up to large instances due to their exponentially increasing complexities. Nevertheless, the state of the art TSP solvers can solve symmetric TSP instances with thousands of nodes by introducing carefully handcrafted heuristics. These heuristics describes how to navigate the space of feasible solutions in an efficient manner, which solves TSP on a significantly larger scale. One of the widely accepted exact TSP solvers is Concorde [3] that iteratively solves linear programming relaxations of the TSP jointly with the branch-and-bound method. Similarly, the state of the art approximate search heuristic, Lin-Kernighan-Helsgaun heuristic [5, 9], has been proved that finds the optimal solution of symmetric TSP up to hundreds of nodes.

The discovery of applying neural networks to solve combinatorial optimization has a long history. One of the earliest proposals is the use of Hopfield networks [6] for the TSP. The authors modify the network's energy function to make it equivalent to TSP objective and use Lagrange multipliers to penalize the violations of the problem's constraints. A limitation of this approach is that it is sensitive to hyperparameters and parameter initialization as analyzed by [22]. Even though these neural networks have many appealing properties, they are still limited as research work. When being carefully benchmarked, they have not yielded satisfying results compared to algorithmic methods [15]. Perhaps due to the negative results, this research direction is largely overlooked since the turn of the century.

Recently, neural networks have gained significantly favor over other methods as their outstanding performance [8, 4, 19]. Neural networks nowadays are widely applied in computer vision, machine translation and other related areas [1, 14]. Motivated by the advancements in sequence-to-sequence learning models, the TSP is revisited since the inputs and outputs of TSP are considered as sequences as well. By adopting recurrent networks with non-parametric softmax functions in the sequence-to-sequence models, the model predicts the sequence of visited cities. Note that the model is trained in a supervised manner with the ground-truth labels produced by an approximate solver.

3 Method

We will illustrate the method centering around TSP in this section. Given an input graph, represented as a sequence of n cities in a two-dimensional space $s = \{x_i\}_{i=1}^{n}$, where each $x_i \in \mathbb{R}^2$, we are concerned with finding a permutation of the points π, termed a tour, that visits each city once and has the minimum total length. We define the length of a tour defined by a permutation π as

$$L(\pi|s) = ||x_{\pi(n)} - x_{\pi(1)}||_2 + \sum_{i=1}^{n-1} ||x_{\pi(i)} - x_{\pi(i+1)}||_2,$$

where $||\cdot||$ represents the L_2 norm.

The objective is to learn the parameters of the stochastic policy $p(\pi|s)$ that produces the optimal tour with given input s.

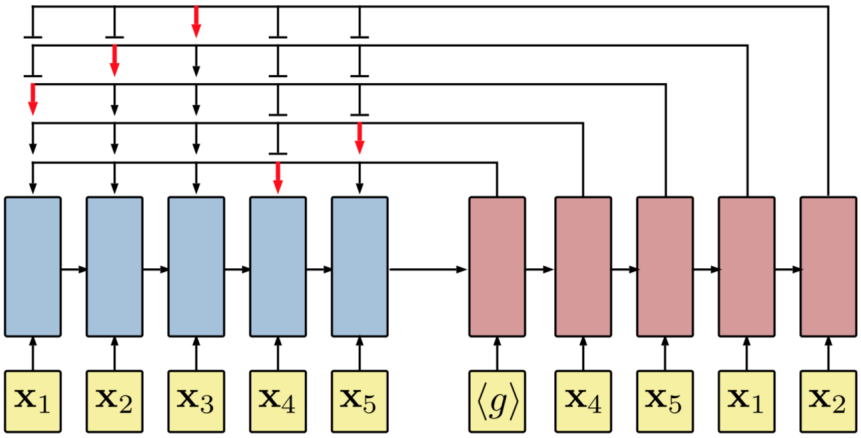

Figure 1: Simple illustration of pointer networks. [21]

3.1 Pointer Networks

Consider the inputs and outputs of TSP, it is very clear that a sequence to sequence (seq2seq) model is suitable for this problem. The result is further computed by the factorization based on the chain rule. However, this approach is not favorable for the following reasons. First,the network has limited flexibility to generalize to inputs with more than n cities. Second, the backpropagation mechanism requires the ground-truth to optimize the parameters but ground-truth labels (optimal tour) are normally unavailable. Pointer networks [21] are introduced to solve the aforementioned problems.

Pointer networks are proposed to solve this issue aforementioned. It is composed of an encoder and a decoder, both of which consist of LSTM cells. The encoder converts the input sequence to a latent feature that is then fed to the generating network. At each step, the generating network produces a vector that modulates a content-based attention mechanism over inputs. The output of the attention mechanism is a softmax distribution with dictionary size equal to the length of the input. The special pointing mechanism allows pointer networks to handle variable length input dictionaries, which is particularly useful in solving TSP.

The input to the encoder network at time step i is a d-dimensional embedding of a 2D point x_i, which is obtained via a linear transformation of x_i shared across all input steps. The decoder network also maintains its latent memory states and, at each step i, uses a pointing mechanism to produce a distribution over the next city to visit in the tour. Once the next city is selected, it is passed as the input to the next decoder step. As shown in Figure 1, the input of the first decoder step is a d-dimensional vector treated as a trainable parameter of our neural network.

3.2 Attention Mechanism

In the regular seq2seq model, we embed our input sequence $x = x_1, x_2, ..., x_T$ into a context vector c, which is then used to make predictions. However, the fixed-length context vector design is incapability of remembering long sentences, which limits the model performance over long-term sequences. The attention mechanism [10, 1] was born to help memorize long source sentences in neural machine translation. Rather than building a single context vector out of the encoder's last hidden state, the attention layer creates shortcuts between the context vector and the entire source input. The weights of these shortcut connections are customizable for each output element.

Figure 2 provides high-level ideas of the attention mechanism. Suppose we have a source sequence $\{x_1, \cdots, x_n\}$ and try to output a target sequence $\{y_1, \cdots, y_m\}$. The conditional probability of output with the attention mechanism is written as:

$$p(y_i|y_1, \cdots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i),$$

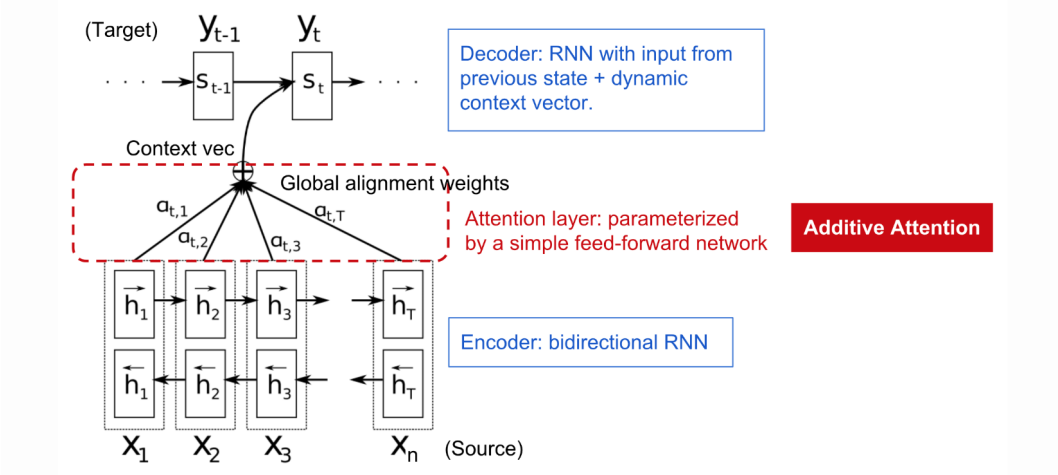

Figure 2: The encoder-decoder model with additive attention mechanism. [1]

where f and q are nonlinear functions, s_i is an hidden state for time i, computed by

$$s_i = f(s_{i-1}, y_{i-1}, c_i)$$

The context vector c_i depends on a sequence of annotations $\{h_1, \cdots, h_{T_x}\}$ to which an encoder maps the input sentence. Each annotation h_i contains information about the whole input sequence with a strong focus on the parts surrounding the i^{th} word of the input sequence. The context vector c_i is, then, computed as

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

The weight α_{ij} of each annotation h_j is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})},$$

where $e_{ij} = a(s_{i-1}, h_j)$ is an alignment model which scores how well the inputs around position j and the output at position i match. $a()$ is another nonlinear function that is suggested to be parameterized by a feed-forward neural network which is jointly trained with all the other components of the proposed system.

3.3 Policy Gradients

[21] utilizes pointer network in a classic supervised way, where the parameters are learned through the cross entropy objective between the output softmax probability and the optimal solution computed by a TSP solver. This learning strategy has several disadvantages that makes it less desirable. First, the performance of the model is strongly affected by the quality of the supervised labels, which brings highly uncertainty to the training process. Second, it's generally very expensive to get high-quality labeled samples.

As discussed above, supervised training scheme has its internal limitations when applying to TSP related problems. On the other hand, Reinforcement learning [18, 16] provides an alternative approach for training neural networks for combinatorial optimization problems. RL learns a agent that maximizes the cumulative reward in a specific environment. Combinatorial optimization problems have simple reward mechanisms, which can be combined with RL very well.

The training objective of a pointer network with parameters θ through model-free policy-based RL is

$$J(\theta|s) = \mathbb{E}_{\pi \sim p_{\theta(\cdot|s)}} L(\pi|s)$$

The stochastic gradient descent is applied jointly with the policy gradient method to learn the parameter θ. The gradient of the $J(\theta|s)$ is

$$\nabla_\theta J(\theta|s) = \mathbb{E}_{\pi\sim p_{\theta(\cdot|s)}}[L(\pi|s) - b(s)\nabla_\theta \log p_\theta(\pi|s)],$$

where $b(s)$ is the baseline function that only depends on s and estimates the expected tour length to reduce the variance of the gradients. Note that the gradient formula can be further approximated with Monte Carlo sampling methods if we assume each optimal tour π_i only depends on the input sequence s_i. The gradient is then rewritten as

$$\nabla_\theta J(\theta|s) \approx \frac{1}{B}\sum_{i=1}^{B}[L(\pi_i|s_i) - b(s_i)]\nabla_\theta \log p_\theta(\pi_i|s_i)$$

3.4 Critic Networks

Since the baseline introduced into gradient calculation of pointer network can generally reduce variance of parameters' updates, Advantage Actor-Critic architecture is introduced. One commonly approximation of baseline value is using its previous value and the moving average of the current solution tour length. But this method do not perform well when faced with some complicated graph. Thus, introducing critic network helps improve the learning process.

The critic network has a Long Short-Term Memory (LSTM) encoder, which is the same as the pointer network. A LSTM processing block contains p steps of glimpsing. In addition, the output architecture of critic network is a two layer fully connected neural network with RELU activation functions. In this way, we can learn the baseline function, and also get benefits from it because we use different inputs sequences in a batch training.

TSP training algorithm:

1. Generate TSP datasets randomly from uniform distribution (0,1) based on the sample size, numbers of cities (nodes).
2. Construct the pointer network *Prt* parameterized by θ, which serves as the agent.
3. Construct the critic network *Crt* parameterized by θ_v, which estimates the baseline value.
4. Design:
   (a) Embed nodes to get the vector representations.
   (b) Choose the type of attention to apply in decoder.
   (c) Glimpsing for input weight aggregation
   (d) Rewards design by calculating the expected tour length
5. Loop:
   (a) Take the vector representation as the input of the pointer network and the critic network and output sample solutions(tour) and Expected tour length prediction(baseline), respectively.
   (b) Based on the reinforce algorithm, compute the gradient with respect to θ.
   (c) Based on mean square error loss, compute the gradient with respect to θ_v.
   (d) Perform backpropagation to update the pointer network and the critic network

3.5 Extended to Bin-packing Problem

Another domain selected to test the proposed technique of combining reinforcement learning and neural networks is the bin packing problem. The bin packing problem is defined as: 'given a list of objects and their sizes, and a collection of bins of fixed capacity, find the smallest number of bins so that all the objects are assigned to a bin'. This problem is considered an NP-hard combinatorial optimization problem. In this project, a sub-domain problem is chosen where bins are with a fixed size capacity M and all pieces are with integer size values from 1 to M-1. For each training instance, pieces are generated based on a uniform probability distribution [1, M-1] in python. Though this project focuses on one-Dimensional bin packing problem, it can also be extended to 2D and 3D, providing even bigger challenges for the model.

3.5.1 Policy Gradients and Hyper-Heuristics

Hyper-heuristics are high-level algorithms that search the space for solutions through a finite set of low-level heuristics. In this project, the policy gradients technique was chosen to generate the heuristic-selection model that interacts with a series of instances and decide the most appropriate heuristic for each state. The policy function is approximated using a neural network and it is used to sample actions which in this case are a set of hand-coded heuristics. The observation (a vector with information about the new state of the environment) and reward functions are also hand-coded.

3.5.2 The Policy

The policy function is approximated using a fully connected neural network. The neural network architecture is composed of the input vector x (a vector of 9 features), a hidden layer of 100 neurons and an output layer of 1 neuron for binary classification, which is designed to choose one of the two hand-coded heuristics.

The input vector features are related to either information about the pieces left to fit and information about the space within the currently opened bins:

1. Total space from pieces left.
2. Average size from pieces left.
3. The size of the current piece to fit in a bin.
4. Number of pieces left.
5. Minimum space left from the current bin list.
6. Maximum space left from the current bin list.
7. The percentage of space used from all the space in the current bin list.
8. Average space left from a bin
9. Number of opened bins

These observations are designed to be compact and contain statistical information about the problem instance to approximate an exact information state. The policy's parameters are initialized so the probability sampled at the output layers are relatively equal at the start of the training. This, along with a stochastic selection during training, encourages exploration of different heuristic combinations. The model is trained using a gradient descent technique with an RMSprop normalization.

3.5.3 The Action

The policy function samples a set of probabilities for each of the possible heuristics to select. During the experiments, the model is optimized to choose between 2 heuristics. For this one-dimensional bin packing problem, the selected heuristics were:

1. InsertLast: Insert the piece in the last open bin
2. InsertFirst: Insert the piece in the first bin that has enough space.

The proposed Heuristics are hand-coded and implemented as functions in python. A new bin is opened if neither of the heuristics is able to insert a piece inside the current bins.

3.5.4 The Rewards

The reward function is hand-coded to give the algorithm a scalar score based on its decisions. A positive score should be assigned when pieces are inserted correctly while the negative score is assigned when bad decisions are made (below some arbitrary used space threshold). It's crucial to assigning rewards correctly since it affects the final direction of the gradient for each of the decisions taken.

Rewards are generated in the following way: The reward function gives 2 points for each full bin and gives the value of the current space used in percent for each bin that is on 70 percent of its capacity or

above. And for those bins with a value below 70 percent threshold, the reward function assigns the value of $v = -(1 - s)$ where s is the current space used and v ranges from -1 to -0.3.

The reason to assign more than double of the points than filling it between 0.7 - 0.99 is to encourage the action to fill out a bin than fit a bin that is not even close to full. Values below .7 are penalized to encourage filling bins rather than opening up new bins. Another reason for these values is extending the reward function's domain to negative values. The idea behind is that samples that gave out too many opened and non-filled bins have a big negative score and, while optimizing, the gradient will move in the opposite direction to prevent doing those same decisions. On the other hand, Samples with the most full bins will have a big positive score and the gradient will be scaled to increase the probability of taking that set of actions. In this project, the reward function is called only at the end of the decision making process.

3.5.5 Model Optimization

Data obtained from running various problem instances and generating the samples are organized in sets of samples called episodes. For each episode, the algorithm calculates the gradient of the parameters and sums them up with the gradients from other episodes until it reaches an arbitrary episode limit 10. The model is tuned using stochastic gradient descent along with RMSprop normalization.

For this model, backpropagation occurs every 10 episodes. Each episode contains 10 samples from randomly generated instances. And each instance is composed of N pieces and thus gives N different decisions of forward propagations through the policy function. This accounts to 100 sequences (10 episodes $*$ 10 samples) or 100 $*$ N decisions taken in different problem instances.

4 Experiments

4.1 TSP

We train our model to solve TSP on 10 and 20 nodes datasets. We set the batch size equals to 128. The learning rate is 1e-3 initially and it decays by 0.9 every 1000 iterations. The number of samples in the training set and test set are 1280000 and 5000, respectively. We compared the performance between moving average baseline applied to pointer network and the Actor-critic network. The plots of these two implementations are shown in the Figure 3 - 6. From the plots, we observe that Actor-critic network have faster convergence speed and the loss of both networks decrease monotonically. The tour length produced by our validate step is close enough to the optimal solution. However, our model still has a strong potential compared to other state of the art works, such as [2, 7].

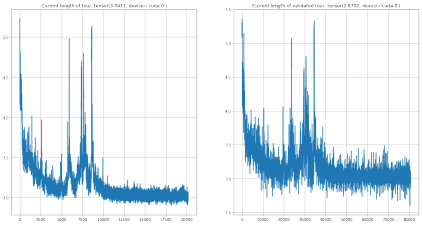

Figure 3: TSP 10 results with the moving-average baseline. The final train and validation tour lengths are 3.0411 and 2.8702, respectively.

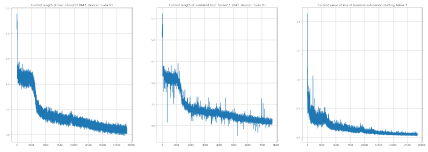

Figure 4: TSP 10 results with the actor-critic network. The final train and validation tour length of the actor network (left two) are 3.0843 and 3.1043, respectively. The plot on the right represents the loss values of the critic network (right one).

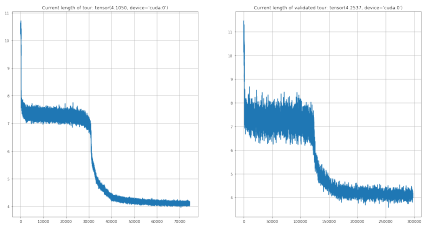

Figure 5: TSP 20 results with the moving-average baseline. The final train and validation tour lengths are 4.1050 and 4.2537, respectively.

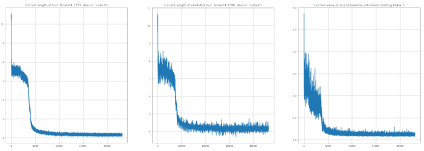

Figure 6: TSP 20 results with the actor-critic network. The final train and validation tour length of the actor network (left two) are 4.1743 and 4.3299, respectively. The plot on the right represents the loss values the of critic network (right one).

The first reason why our implementations have lower performance is that the batch size in all training processes is 128, which may not be large enough to help our sample solution approximate the gradient of parameters in the pointer network. The second possible reason is that we do not apply temperature control in attention method, which might cause our model to act over confidently. Finally, the hyperparameters we applied in the optimization process might not ideal, therefore, it's likely the networks have trouble finding the best local minima.

4.2 Bin-packing

The Heuristics-selection model for the bin-packing problem was trained using 300,000 randomly generated instances. Algorithm's results were measured based on average bin space used (ASU). This is basically the sum of every space used within each bin divided by the total number of bins and it measures whether the filling decisions were made correctly as lower ASU means more bins are relatively open.

In this experiment, we tried different numbers of pieces in each generated instance (list of pieces). There are 10, 20, 50 (e.g. If the number of pieces in an instance is 50, we need to fit all 50 pieces into a collection of fixed size bins). The detailed result plots are shown in Figure 5 - 10.

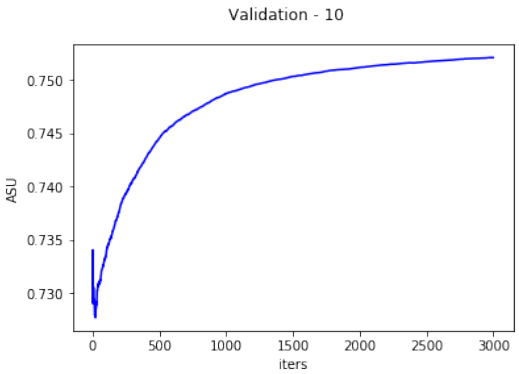

Figure 7: ASU Validation plot with 10 pieces

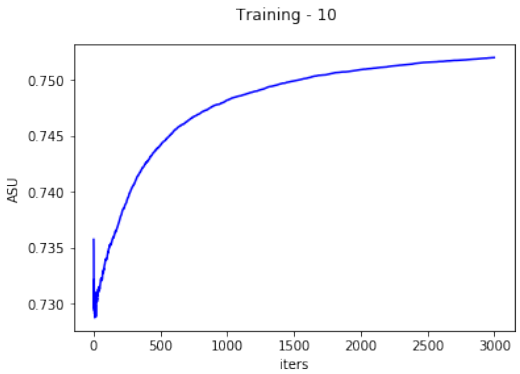

Figure 8: ASU Training plot with 10 pieces

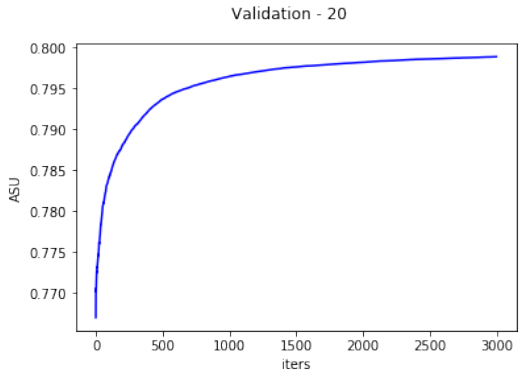

Figure 9: ASU Validation plot with 20 pieces

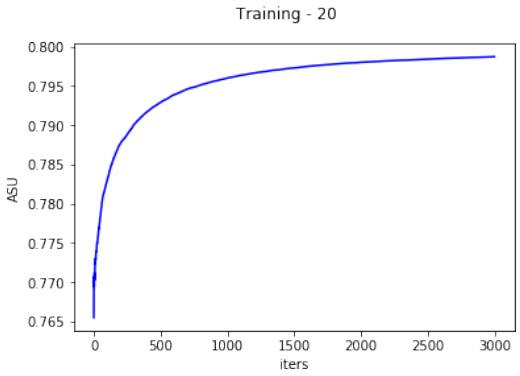

Figure 10: ASU Training plot with 20 pieces

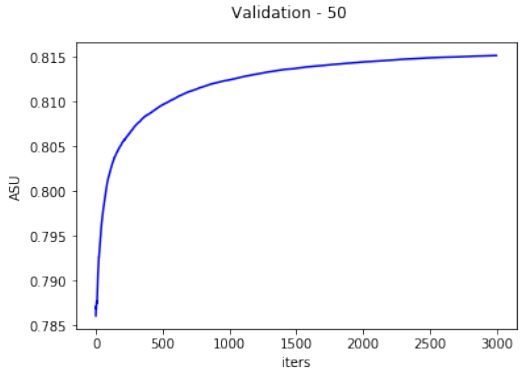

Figure 11: ASU Validation plot with 50 pieces

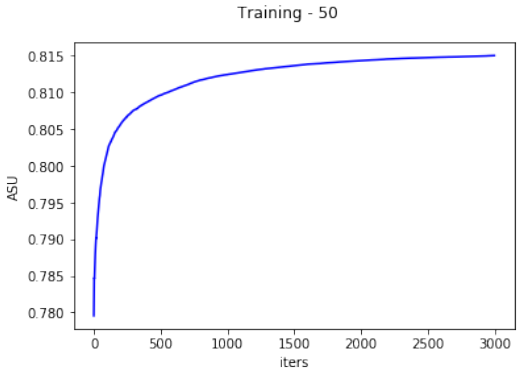

Figure 12: ASU Training plot with 50 pieces

From training and validation plots within all settings, we can clearly see that, generally, ASU value increases as we have more iterations though, for the setting when the number of pieces is 10 in an instance, the ASU value drops at the very beginning of iterations. Thus, this performance shows that our heuristic-selection model is very promising. In addition, after training, the final training ASUs for 10, 20, 50 pieces are 0.7520, 0.7987, 0.8157 and the final testing ASUs for 10, 20, 50 pieces are 0.7521, 0.7988, 0.8152.

5 Conclusion

In this project, we implement two algorithms using reinforcement learning and neural networks to solve traveling salesman and bin-packing problems respectively. Reinforcement learning combined with pointer networks, an recurrent neural network with non-parametric softmax functions, is applied to tackle traveling salesman problem. Our model achieves encouraging performances as the solution is close enough to the optimal solution. For bin-packing problem, we explores the implementation of policy gradient, a technique that combines both reinforcement learning and neural networks, on heuristics-selection models. This proposed algorithm for one-dimensional bin-packing problem achieves promising results as average bin space used monotonically increases after few iterations in all three different settings.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[3] Vasek Chvatal David L Applegate, Robert E Bixby and William J Cook. Concorde tsp solver, 1999.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] Keld Helsgaun. An effective implementation of the lin–kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[6] John J Hopfield and David W Tank. "neural" computation of decisions in optimization problems. *Biological cybernetics*, 52(3):141–152, 1985.

[7] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Shen Lin and Brian W Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations research*, 21(2):498–516, 1973.

[10] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[13] Christos H Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical computer science*, 4(3):237–244, 1977.

[14] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[15] Farah Sarwar and Abdul Aziz Bhatti. Critical analysis of hopfield's neural network model for tsp and its comparison with heuristic algorithm for shortest path computation. In *Proceedings of 2012 9th International Bhurban Conference on Applied Sciences & Technology (IBCAST)*, pages 111–114. IEEE, 2012.

[16] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[17] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[18] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[19] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[20] Aybars Uğur, Serdar Korukoğlu, Ali Çalıskan, Muhammed Cinsdikici, and Ali Alp. Genetic algorithm based solution for tsp on a sphere. *Mathematical and computational applications*, 14(3):219–228, 2009.

[21] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[22] GV Wilson and GS Pawley. On the stability of the travelling salesman problem algorithm of hopfield and tank. *Biological Cybernetics*, 58(1):63–70, 1988.