HIGH SCHOOL EDITION
Journal of Student Research
Volume 10 Issue 4 (2021)

Linear Function Approximation as a Resource Efficient Method to Solve the Travelling Salesman Problem

Rolan Guang[1] and Sajad Khodadian[#]

[1]Mentor High School
[#]Advisor

ABSTRACT

This paper presents an approach to combinatorial optimization problems using linear function approximation (LFA) to solve the Travelling Salesman Problem (TSP). We create a reinforcement learning model in which we parameterize our policy using linear function approximation instead of the more commonly used neural networks. We then evaluated our models based on two factors: training time and optimality. When we compared our results with a state-of-the-art neural network solver, we found that our model was able to solve the TSP accurately while using drastically less computational resources and time to train than the neural network algorithm (Kool et al., 2019).

Introduction

The Travelling Salesman Problem (TSP) is a classic example of a non-deterministic polynomial-time or NP-hard combinatorial optimization problem: problems that are impossible to solve using pure computational power (Drori et al., 2020). The most widely studied state of the TSP problem is as follows: given a set of cities, how can we generate the shortest path for a salesman to travel to each city exactly once and return to the original city. The TSP is classified as a NP-hard combinatorial problem because as the number of cities for the salesman increases, the number of possible permutations of paths through these cities increases exponentially.

Throughout literature, the TSP has been studied extensively because of its many applications. First and foremost, the problem is representative of a larger class of problems called combinatorial optimization problems. Thus, if one can find an efficient algorithm for the TSP, the algorithm can be applied to other combinatorial optimization problems as well (Bello et al. 2017).

A solution to the TSP has many practical applications. First, it has obvious applications in logistics and transportation making a TSP solver valuable to the military, commercial delivery services, and bus companies, etc. (Cappart et al. 2020). A TSP solver can also be more generally applied in the fields of microchip design, genome sequencing, fiber optic network design and others, in which a "city" would represent soldering points, DNA fragments, and wiring points (Cappart et al. 2020). The TSP can even be used to solve optimal control problems, with one such example being in astronomy, where astronomers want to minimize the time spent moving the telescope between the sources.

Gurobi (Gurobi Optimization, LLC, 2021) and Concorde (Applegate et al., 2003) are two main solvers created for the TSP, but both solvers are made with heuristics. Solvers using heuristics can be incredibly accurate and fast but they require a lot of time and specialized knowledge to create. In addition, it is difficult to create heuristics that can solve every type of TSP. As such, a desire for solvers with increased generality has led to the creation of other types of algorithms to solve the problem. In particular, more and more neural network models have been proposed to solve this TSP. These neural networks are very accurate but they have a very complicated framework and thus require a large amount of computational resources and time to train.

In this paper, we propose a different method for solving the TSP. Specifically, we parameterize our sampling policy with Linear Function Approximation. This parameterization offers a simple and computationally efficient method for solving the TSP. Meanwhile, it does not reduce the accuracy of the solver significantly. In our model, we use linear functions to parameterize the softmax function for the sampling of a tour. After sampling a tour, we then calculate the cost of our tour and use gradient descent to adjust our policy to minimize the cost. We then compare our linear function parameterization algorithm with a neural network algorithm and an exact TSP Solver by training and running each model, given 10, 20, 30, and 40 random cities. We measure the effectiveness of our model based on two criteria:

1. Solution accuracy: We calculate the optimality gap by comparing the tour generated by our LFA model and the exact tour generated by the Gurobi TSP Solver. Our model is able to achieve similar optimality when compared to the neural network models.
2. Training Time: Our model is able to train extremely quickly compared to a state-of-the-art neural network model. For instance, our TSP solver for 20 cities trained in just 2.62 seconds while it takes over 11 hours to train the deep learning model using attention layers.

Literature Review

The Travelling Salesman Problem is an extremely simple problem to explain. The problem asks, "Given a list of cities and the distances between pairs of cities, what is the shortest possible route that visits each city and returns to the origin city?" (Bello et al., 2017). However, it is not an easy problem to solve. Finding the solution to the TSP is NP-hard, even in simpler 2D Euclidean cases where nodes, or cities, are 2D points and edge weights are Euclidean distances between pairs of points. In efforts to solve the TSP, many models have been created, including supervised learning models and reinforcement learning models. These models also use a variety of algorithms such as the greedy algorithm, beam search algorithm, and other heuristics (Mazyavkina et al., 2020). A simple solution to the TSP uses the Greedy Algorithm, where the salesman travels to the closest city, minimizing short term loss but not accounting for long term loss. The Greedy Algorithm works well for TSP instances with a small number of cities, but as the number of cities increases past a low threshold the Greedy Algorithm becomes more and more inaccurate.

As the TSP became studied more and more, researchers have been able to create incredibly precise TSP solvers using more complicated heuristics. According to (Mulder and Wunsch, 2003), Concorde (Applegate et. al., 2003) is "widely regarded as the fastest TSP solver, for large instances, currently in existence." Concorde uses the cutting-plane method to iteratively solve linear programming relaxations of the TSP and the powerful solver has been applied to problems with gene mapping, protein function prediction, vehicle routing, and in studying the scaling properties of combinatorial optimization problems among other applications (Applegate et al., 2003). Gurobi (Gurobi Optimization, LLC, 2021) is another optimization solver designed to solve LP, QP, QCP, and MIP problems. Their model uses handcrafted cutting plane routines and advanced MIP heuristics to quickly find feasible solutions to the TSP.

While Gurobi (Gurobi Optimization, LLC, 2021) and Concorde (Applegate et al., 2003) are extremely proficient in solving most TSP instances, the problem with the heuristic solvers is that designing heuristics for combinatorial optimization problems requires significant specialized knowledge and years of research work and heuristic solvers cannot be applied to every problem. This challenge has led to an interest in raising the level of generality at which optimization solvers can operate.

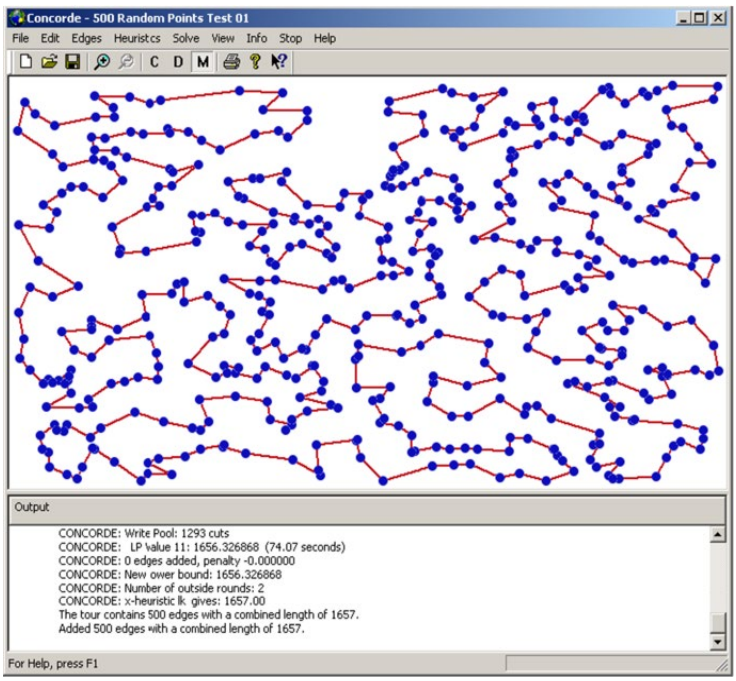

Figure 1: Concorde TSP Solver Solving a 500 City TSP Instance

As a result, many researchers have started to create neural networks to solve the TSP in efforts of creating a model that can solve every TSP. One notable deep learning model to solve the TSP is the recently created Graph Attention Network (Kool et al., 2019) based on reinforcement learning and neural networks. It uses an attention-based decoder trained with reinforcement learning to autoregressively build TSP solutions. Compared to other neural network reinforcement learning models, the Graph Attention Network (Kool et al., 2019) uses a more powerful decoder and trains the model with a greedy rollout baseline to achieve state-of-the-art results in both speed and accuracy.

Another deep learning approach to the TSP uses Graph Convolutional Networks and beam search (Joshi et al., 2019). The model takes in a graph as an input and extracts compositional features from its nodes and edges by stacking several graph convolutional layers (Joshi et al., 2019). The neural network can convert the input to an edge adjacency matrix denoting the probabilities of edges occurring on the most optimal TSP tour. Then, the edge predictions are converted to a tour using a post-hoc beam search technique (Joshi et al., 2019).

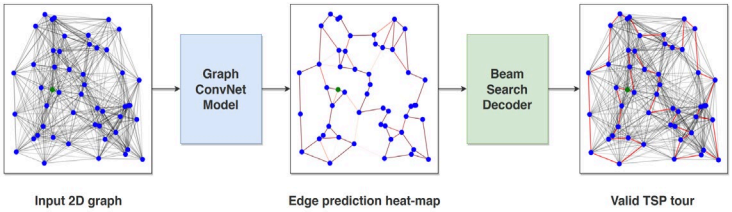

Figure 2: Graph Convolutional Network TSP Solver Architecture

Currently, these neural network implementations have not been able to match the algorithmic methods in terms of speed and solution quality. In addition, they use up a lot of computational resources and time. Even for simple problems with as few as 20 cities, these neural networks have to train for hours on end. We believe that we can use Linear Function Approximation as a substitute for neural networks to create a simpler reinforcement learning model that can effectively solve lower dimensional instances of the TSP with a high degree of accuracy, but faster than the neural network models.

Purpose

For this project, we define two research questions:

1. How resource efficient will our linear function approximation model be when compared to a deep learning model designed to solve the TSP?
2. Will our linear function approximation model effectively solve the random TSP problems it is given?

Methodology

We will apply our model to solving the 2D Euclidean TSP. Given an input graph of n cities in a two-dimensional space, our goal is that given to find a tour which is a one to one function $\hat{\pi}(\cdot)$: $[n] \rightarrow [n]$ that visits each city once and has the minimum tour length where the tour length is calculated using the following:

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

where $|| \cdot ||_2$ is the ℓ_2 norm.

To find the optimal tour, we use our policy $\theta \in \mathbb{R}^n$ which can generate an effective tour when given the locations of the cities s.

$$P_\theta(\pi|s) = P(\pi_2|\pi_1) * P(\pi_3|\pi_2) * \cdots * P(\pi_n|\pi_{n-1}) \ (2)$$

Here, P_θ is the probability of sampling a tour.

Our algorithm then uses gradient descent to minimize the loss of the paths generated by the model. First, we define a way to calculate a tour $\hat{\pi}$. Simply put, we sample a tour by obtaining a set of linear function approximators using our ϕ function and then parameterize our policy with our policy $\theta \in \mathbb{R}^n$ using our linear function approximators. We generate a tour by going from one city to the next until every city has been visited. To go from city i to city j, we define a suboracle that chooses the next city via the following equation:

$$P(i \rightarrow j)^h = \frac{\exp(\theta^T \phi(h, j, i))}{\sum_{j': j' \neq h} \exp(\theta^T \phi(h, j', i))}$$

where h is an array of the past cities the suboracle has travelled to. In neural network algorithms created to solve the TSP, the neural networks estimate the approximators given to the policy. In our linear function approximation model, we generate these approximators via our ϕ method.
Initially, we implemented a simple ϕ function:

$$\phi_{h,s,h_m} = \begin{bmatrix} \| city_1 - h_m \|_2 + \| h_{m-1} - h_m \|_2 \\ \| city_2 - h_m \|_2 + \| h_{m-1} - h_m \|_2 \\ \cdots \\ \| city_n - h_m \|_2 + \| h_{m-1} - h_m \|_2 \end{bmatrix}$$

where h_{m-1} is the city most recently visited and h_m is the potential next city for the suboracle to travel to. This ϕ function only considers two distances: the distance between the current city and the next city and the distance between the next city and every other city. We tried to create as simple a ϕ as possible to minimize computational time, but because of this ϕ function's simplicity the suboracle followed the same path as the greedy algorithm, where the suboracle always travels to the closest city. While the first ϕ function does work well for small instances of the TSP, it would quickly generate increasingly suboptimal tours for more complex instances.

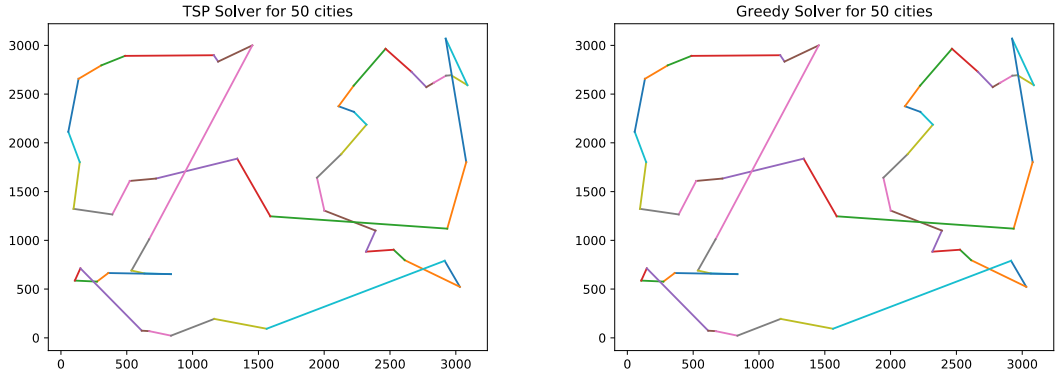

Figure 3: Comparison between Greedy Solver and LFA Solver Using Initial ϕ Function

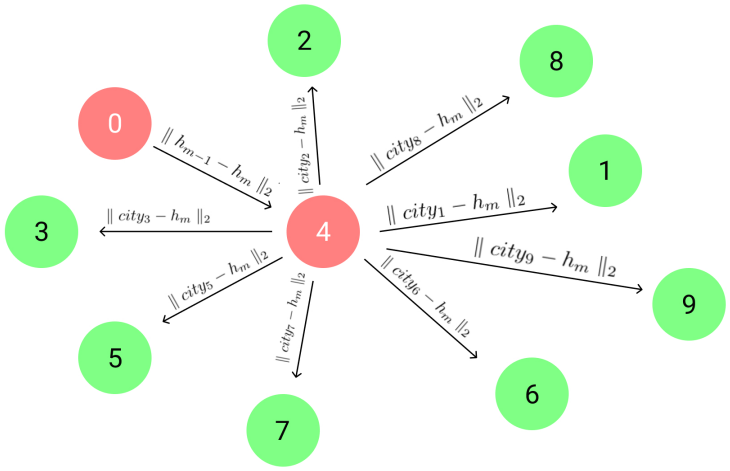

Figure 4: Visualization of First Phi Function

Because of this, we design a more complex ϕ function that accounts for the previous three cities that the suboracle has visited as well as the next city it is travelling to. This new ϕ increases the performance of our model while not affecting computation time. ϕ is an n-dimensional vector $\phi_{h,s,h_m} \in \mathbb{R}^n$ where

$\phi^i_{h,s,h_m} = \| city_i - h_m \|_2 + \sum_{i=m-3}^{m-1} \| h_i - h_{i+1} \|_2$ (3)

After we use our ϕ function and policy θ to sample a city to travel to, the city is added to the history h and the suboracle travels to new cities until every city has been travelled to. When every city has been visited, the suboracle then returns to the origin city.

Algorithm 1 Sample a Tour

$\phi_{h,s,h_m} \leftarrow []$
for i=1...i-1 **do**
  for j=1...j-1 **do**
    $\phi^j_{h,s,h_m} = \| city_j - h_m \|_2 + \sum_{i=m-3}^{m-1} \| h_i - h_{i+1} \|$ (3)
  end for
  $d_i = \theta^T \phi_{h,s,i}, \forall i \in [n] \backslash h$
  $p_i = \frac{\exp d_i}{\sum_{j \notin h} \exp d_j}, \forall i \in [n] \backslash h$
  $h_m \sim p$
  $h \leftarrow h \cup h_m$
end for

Algorithm 1: Sample a Tour

Now that we have defined how to sample a tour, we need to update our policy. We first define the cost of a tour, which we will minimize using gradient descent. We define the cost of a tour as:

$L(\pi_i|s_i)\nabla \log P_\theta(\pi_i|s_i)$ (4)

where,

$$\nabla \log P_\theta(\pi_m|\pi_{m-1}) = \phi_{h,s,h_m} - \sum_i \phi_{h,s,i} * P(i|h) \text{ (5)}$$

Using the cost of a tour, we use standard gradient descent to minimize the cost of the tours generated by our policy θ. We define a hyperparameter N that determines how many times we estimate our gradient before updating it. We then use eq. 4 and estimate our gradient N times

$$\nabla J(\theta) \approx \frac{\sum_{i=1}^{N} L(\pi_i|s_i)\nabla \log P_\theta(\pi_i|s_i)}{N} \text{ (6)}$$

Using the gradient generated, we update our policy θ k times, where k is a hyperparameter that determines how many times to update our policy and γ is the hyperparameter stepsize, which is used to adjust the scale of policy modifications.

$$\theta_{k+1} = \theta_k - \gamma \nabla J(\theta_k) \text{ (7)}$$

Algorithm 2 Estimate Gradient and Update Policy

Input: step size η, initial vector θ_0
for k=1:K **do**
  $G = 0$
  for n=1:N **do**
    Sample state s_n and Tour π_n using algorithm 1
    Evaluate $L(\pi_n|s_n)$ using Equation (1)
    Evaluate $\nabla \log P_\theta(\pi_n|s_n)$ using equation (6)
    $G \leftarrow G + L(\pi_n|s_n)\nabla \log P_\theta(\pi_n|s_n)$
  end for
  $\theta_{k+1} = \theta_k - \eta G/N$
end for

Algorithm 2: Estimate Gradient and Update Policy

After updating our policy θ k times, we have parameterized our policy θ using linear function approximation which we can then use to solve TSP instances.

Results and Discussion

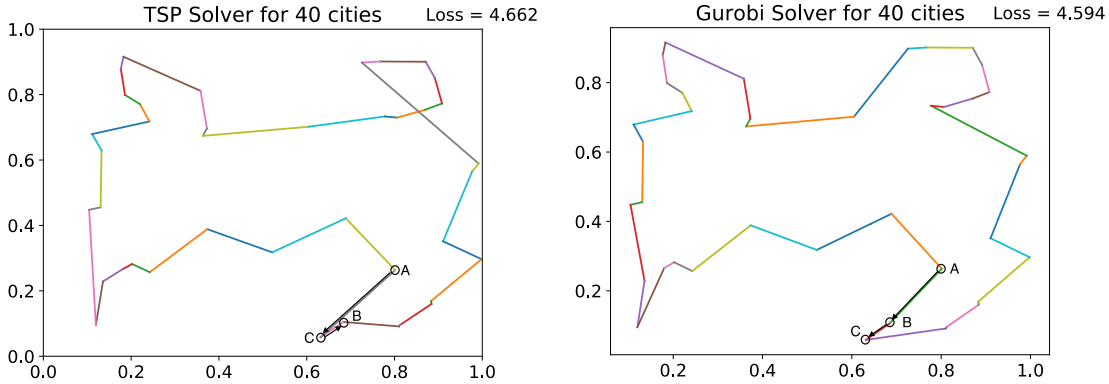

Figure 5: Comparison between LFA Model and Gurobi Exact Solver

When we compare the optimality gap of our LFA model and the Graph Attention Network model (Kool et al., 2019), we find that for TSP instances with fewer nodes, our LFA model is able to generate extremely close to optimal tours. The tours our model generates is often very similar to the exact solution. However, our model sometimes chooses the

wrong point when there are two very close points. This can be shown in our TSP Solver shown in Figure 5, where our model travels from point A to point C instead of point B. This in turn causes the model to take several less efficient tours. However, these errors do not cause a significant detour from the optimal route and our model is still able to generate tours that are very close to the optimal tour.

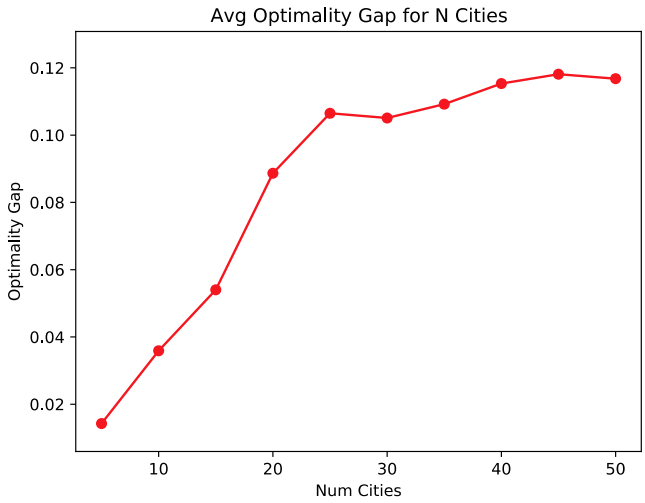

Figure 6: Avg Optimality Gap vs Number of Cities Over 500 Iterations

Moreover, our model can successfully solve TSP instances regardless of the distribution of the cities. The standard deviation for our model is always <0.1% (Figure 6) and does not increase noticeably when the number of cities increases. As the number of cities increases, the average optimality gap of our solver increases, showing that our LFA model is most effective at solving smaller and simpler instances of the TSP. The average optimality gap increases quickly as the number of cities increases from 5 to 25 but converges at around 11%. Our LFA TSP Solver has been shown to be most effective in solving simpler 2D Symmetrical TSP instances, but still effectively solves larger instances of the TSP.

Table 1: Average Optimality Gap Using Gurobi Solver as a Baseline

	TSP10	TSP20	TSP30	TSP40
Gurobi	0.00%	0.00%	0.00%	0.00%
Kool et al. [2019]	0.01%	0.52%	0.72%	1.32%
Our Model	3.59%	8.86%	10.51%	11.53%

We ran our model and (Kool et al., 2019) with the same TSP instances and used (Gurobi Optimization, LLC, 2021) as a baseline and calculated the average optimality gap between the neural network model and our LFA model. When our solver is compared with the neural network TSP solver, it has similar optimality gaps for smaller instances of the TSP problem. As the number of cities increases, our model is still able to generate efficient tours but gradually it would become less accurate. In other words, our solver is able to effectively solve the TSP problem, albeit less accurately than the neural network models, even as the number of cities increases.

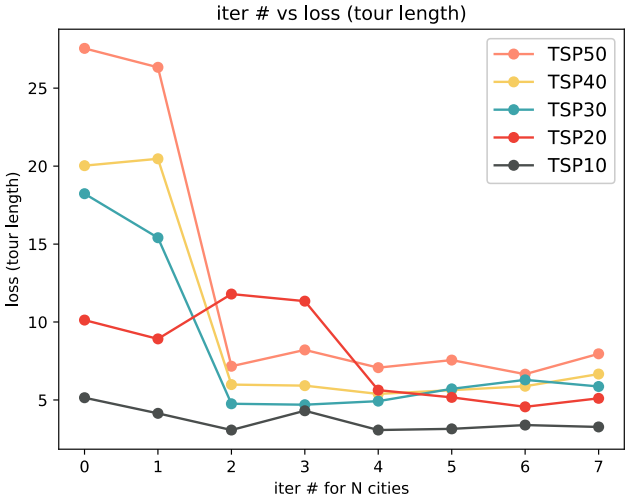

Figure 7: Iterations Needed to Minimize Loss Given Different Amounts of Nodes

The real advantage of our linear function approximation reinforcement learning model is the incredibly low time required to train the model. We can train our model in mere seconds, much quicker than the tens of thousands of seconds needed to train a neural network model (Kool et al., 2019). The LFA model is quick to train because it can find a local minimum quickly using gradient descent and update its policy $\hat{\pi}$ accordingly. In figure 7, it is noteworthy that regardless of the dimensionality of the problem, our model can converge to a local minimum within two gradient updates or four gradient estimates.

Table 2: Training Time of Models (seconds)

	TSP10	TSP20	TSP30	TSP40
Kool et al. [2019]	1203s	6480s	27532s	65022s
Our Model	0.34s	2.62s	11.74s	36.23s

We trained a neural network model (Kool et al., 2019) and our model on an NVIDIA 3070 processor with 8gb RAM and recorded the time taken to train each model. The train time of our LFA algorithm is significantly lower than train time for the neural network algorithm because of our model's simpler architecture. When we compare the train time of the neural network algorithm to the train time of our LFA algorithm, we find that our model trains thousands of times more quickly than the neural network model regardless of the number of cities.
All in all, our results show that linear function approximation can be used to create an extremely lightweight model that could solve lower dimensional instances of the TSP while using far less time and computational resources than a neural network model. Our model trains extremely quickly, much more so than the neural network model while still generating efficient tours for the salesman.

Conclusion

In this paper we introduce a new approach to the Travelling Salesman Problem by applying Linear Function Approximation. We compare our model with the state-of-the-art Graph Attention Network model (Kool et al., 2019) and show that our model is not only effective in solving all the 2D Euclidean TSP instances, but it also generates tours with high accuracy when compared to the Gurobi exact TSP solver (Gurobi Optimization, LLC, 2021). In other words, our model can achieve similar optimality gaps in comparison to the neural network model, even though our model would become less effective in solving higher dimensional instances of the TSP. However, our model makes up for the loss

of optimality by gaining in train time: it is thousands of times faster than the neural network models. The simplified structure of linear function approximation models allows them to train extremely quickly. Essentially, our model is much simpler and requires significantly less time and resources to train than the neural network models. Still, our model generates effective tours that are comparable to the accuracy of the neural network solver's tours. Based on the result, we are optimistic that linear function approximation is an effective method to use to solve computational optimization problems. In the future we would like to improve upon our model's accuracy and effectiveness in solving higher dimensional instances of the TSP by improving heuristics in our model. We want to improve the model's generality so that it can solve different types of TSP. In addition, we would like to apply linear function approximation to other kinds of combinatorial optimization problems. Through these efforts, we can truly discover how powerful linear function approximation algorithms can be in solving the TSP and more broadly, all kinds of NP-hard combinatorial optimization problems.

References

Applegate, D., Bixby, R., Chvatal, V., and Kool, W. (2003). Concorde tsp solver.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2017). Neural combinatorial optimization with reinforcement learning.

Cappart, Q., Chételat, D., Khalil, E., Lodi, A., Morris, C., and Velickovíc, P. (2021). Combinatorial optimization and reasoning with graph neural networks.

Cappart, Q., Moisan, T., Rousseau, L.-M., Pr émont-Schwarz, I., and Cire, A. (2020). Combining reinforcement learning and constraint programming for combinatorial optimization.

Drori, I., Kharkar, A., Sickinger, W. R., Kates, B., Ma, Q., Ge, S., Dolev, E., Dietrich, B., Williamson, D. P., and Udell, M. (2020). Learning to solve combinatorial optimization problems on real-world graphs in linear time.

Gurobi Optimization, LLC (2021). Gurobi Optimizer Reference Manual.

Joshi, C. K., Laurent, T., and Bresson, X. (2019). An efficient graph convolutional network technique for the travelling salesman problem.

Kool, W., van Hoof, H., and Welling, M. (2019). Attention, learn to solve routing problems!

Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. (2020). Reinforcement learning for combinatorial optimization: A survey.

Mulder, S. and Wunsch, D. (2003). Million city traveling salesman problem solution by divide and conquer clustering with adaptive resonance neural networks.