ORML: Project 3

Rory Kolster, SNR: 2070539, ANR: 736326

December 2024

1 Approximating the optimal TSP objective value

1.1 Simple approximation strategies

I implemented two simple strategies, with one having an extension.

The first strategy uses the Nearest-Neighbour or Greedy heuristic. The heuristic starts at a given node and then takes the path to the nearest neighbour that hasn't already been visited. This heuristic is good at giving a quick upper bound to the objective value, but it can perform quite poorly. One thing to note, is that the last edge in the tour is often quite long, as it has no degree of freedom to choose, and the greedy heuristic can move quite far from the initial node. Thus I use both the greedy heuristic soluton as well as the solution without the last edge. For the generation of instances for the training of the Neural Network, I also used these to determine a linear regression model to approximate the optimal solution using the two heuristic solutions. As independent variables I had the number of cities/nodes in the graph, the average length of the edges in the graph and the heuristic solution. The linear regression model, where the coefficients are to be determined is as follows:

$$\text{optimal value} \approx a + b \times n + c \times (\text{average edge length}) + d \times (\text{Greedy heuristic objective value}). \tag{1}$$

I know that for the Greedy heuristic regarding the TSP, that the objective value does not scale linearly in n, and so for future recommendations I would look into non-linear scaling in n.

For the second strategy, I used the gurobi implementation and set the time limit to 1 second. The way gurobi solves a problem, is such that it works on making the lower bound and upper bound converge together to yield th optimal objective value. I use this by extracting the best found lower bound and upper bound after 1 second of running time. The best found upper bound is the best found objective value. This works well in low instance sizes, yielding decent bounds, or even sometimes optimality, however, in larger instances this works very poorly, and sometimes returns ∞ as an upper bound. Hence, I exploit the found greedy approximations for that instance, and take a linear combination, i.e.

$$\text{gurobi approximation} = 0.2 \times (\text{gurobi lower bound}) + 0.8 \times (\text{greedy approximation}). \tag{2}$$

Although, if optimality is reached, then I just use the optimal value.

I saved the regression coefficents for the greedy approximations in a file, and I generated the gurobi approximations for the NN training instances, as I use it as an extra feature.

1.2 Neural Network approximations

(a) The instance features which I collected to be used as an input layer for the Neural Network are as follows:

- Average edge length in the Graph (upper triangle)

- Standard deviation of the lengths in the Graph (upper triangle)
- Number of nodes with mean edge length below lower quartile of edge lengths
- Number of nodes with mean edge length above upper quartile of edge lengths
- Greedy heuristic solution (objective value)
- Greedy heuristic minus last edge solution (objective value)
- Area of the convex hull containing all of the points
- Perimeter of the convex hull
- Number of nodes used in the convex hull
- LP relaxation solution (objective value)
- Gurobi approximation solution (objective value)

I then generated the training set instances, extracted the above features and determined the greedy heuristic approximations and the gurobi approximation. The training set had a total of 2000 instances; 1000 were of size 20, 500 were of size 40 and 500 were of size 60.

(b) I used the following Neural Network architecture:

- An input layer with 10 nodes and relu activation function,
- 1 Hidden layers, with 10 nodes and relu activation function,
- an output node, with a linear activation function,
- I use Adam as the optimizer with a learning rate of 0.01,
- mean absolute error (mae) loss function,
- I train for 300 epochs with a batch size of 256.

For the layers, I also use batch normalization because some of the input features are not scaled well together with the other features. I use relu as activation function for the hidden and input layers as it overcomes the vanishing gradient problem, allowing the model to learn faster, but more importantly because it is recommended in the literature when dealing with regression. For the same reason, we use the linear activation function in the output node and the mean absolute error (mae) for the loss function.

Fine tuning resulted in choosing 0.01 for the learning rate, which did not learn too quickly but fast enough to converge before the final epochs. The batch size of 256 was not too important as training epochs is relatively quick in this model, but it is small enough to allow the model to actually learn something. I decided upon this small and not too complex architecture as I did not want to overfit on the training data, or over-complicate the model leading to poorer error loss in the test set.

(c) I implemented the above Neural Network and below is the obtained results. I tested the methods on 500 new instances. The number of cities was chosen at random to be an integer between 20 and 80. Then the instance was accordingly generated.

The mean absolute difference (mae) was calculated for each method and the results can be seen in the following table.

Method	**Mean Absolute Error**
Greedy	203.11
Greedy Minus Last	197.44
Gurobi	164.87
Neural Network	165.62

Table 1: Mean Absolute Error for Different Methods

In the appendix, there are histograms illustrating the errors for each method. As you can see the Greedy approximations are the worst, but still not bad, and the greedy minus the last edge slightly outperforms the regular greedy heuristic approximation. The gurobi approximation will always improve on the prior two as it uses it to do its approximation. The NN performs similarly to the gurobi approximation, also particularly because it takes the result as part of its input layer.

However, I suspected that the gurobi approximation would do worse in comparison with the Neural Network approximation over far larger instances, so I tested a single instance at size 100 to investigate this. Unfortunately, solving this to optimality took longer than 8 minutes so I took the midpoint between the upper bound and lower bound found so the near optimal solution was within 100 of the true optimal. The NN approximation was 436 below the near optimal, while gurobi approximation was 525 over, so it does somewhat perform better over larger instances, and potentially other tested instances, if more testing could be done.

1.3 Code and files

In order to allow for quick reproduction of results, numerous files are saved in the zip file, which can be called in the code. In the zip file there is:

- The distance matrices of the training sets in three text files, named "Training_instances_20.txt", "Training_instances_40.txt", "Training_instances_60.txt'.
- The extracted features of the training sets: "Training_set_20.txt", "Training_set_40.txt", "Training_set_60.txt".
- The optimal values for the training set instances: "Optimal_values.txt".
- The gurobi approximations for the training set instances: "Gurobi_approximations.txt".
- The regression models for the greedy and greedy minus last approximations: "model_greedy.pkl", "model_greedy2.pkl".
- The Neural Network keras model: "Model_TSP.keras".
- The predictions of the methods on the test set: "greedy_predictions.txt", "greedy2_predictions.txt", "gurobi_predictions.txt", "NN_predictions.txt", "Optimal_test_values.txt".

2 DQL for OKPU

2.1 Simple Strategies

I implemented 2 simple strategies to play the online knapsack problem with uncertainty (OKPU). The first is the greedy heuristic. The logic behind this one, is that if the estimated weight of the newly offered item fits the remaining capacity, it adds it. This logic is simple, but decently effective for the online problem.

The second strategy tries to estimate the relationship between the weight and profit. It observes the first 15% of items and then based on the mean value density (profit/weight), it adds or does not add the offered item. If the value density is larger than the 80% observed ratio, add the item, otherwise don't. If the estimation is poor, leading to a relatively empty knapsack (less than 30% full) by the last quarter of remaining items, the strategy has built in that it reverts to greedy if the latter happens.

I also implemented a clairvoyant agent, which knows all items and their true weights and profits. Thus this clairvoyant agent can determine the optimal solution by simply solving the regular 0-1 knapsack problem.

Below we show the performance of the greedy heuristic, the estimation strategy and the clairvoyant agent.

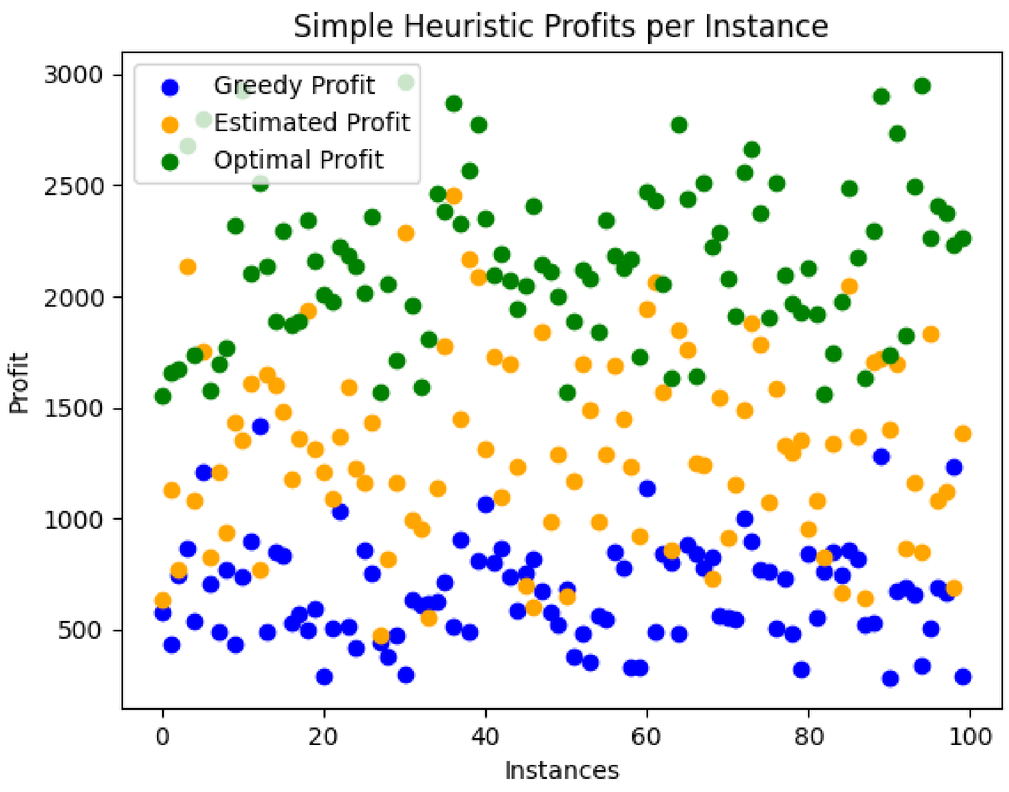

Figure 1: Comparison between simple heuristics.

From the above figure, we can see that on large size instances, estimating the value density proved fruitful and performed better than the greedy heuristic, which was quite surprising.

2.2 Deep Q-learning states and actions

In this section, I give a way to define the state space and actions. A state is defined as $4+3\times(N-1)=3N+1$ integers, where N is an upper bound on the number of items offered in an instance. These integers are as follows:

$$\left[n, c, w_i^{(est)}, p_i, w_1^{(est)}, w_1, p_1, \dots, w_{i-1}^{(est)}, w_{i-1}, p_{i-1}, 0, \dots, 0\right]$$

There are the first 4 integers that are the remaining n items, c the remaining capacity, $w_i^{(est)}$, p_i the estimated weight and profit of the item being currently offered. The remaining integers are the already observed items before step i. Naturally, when at step 1, all of the remaining $3\times(N-1)$ are zeros.
There are two possible actions at each state: either to take the offered item, or not to.

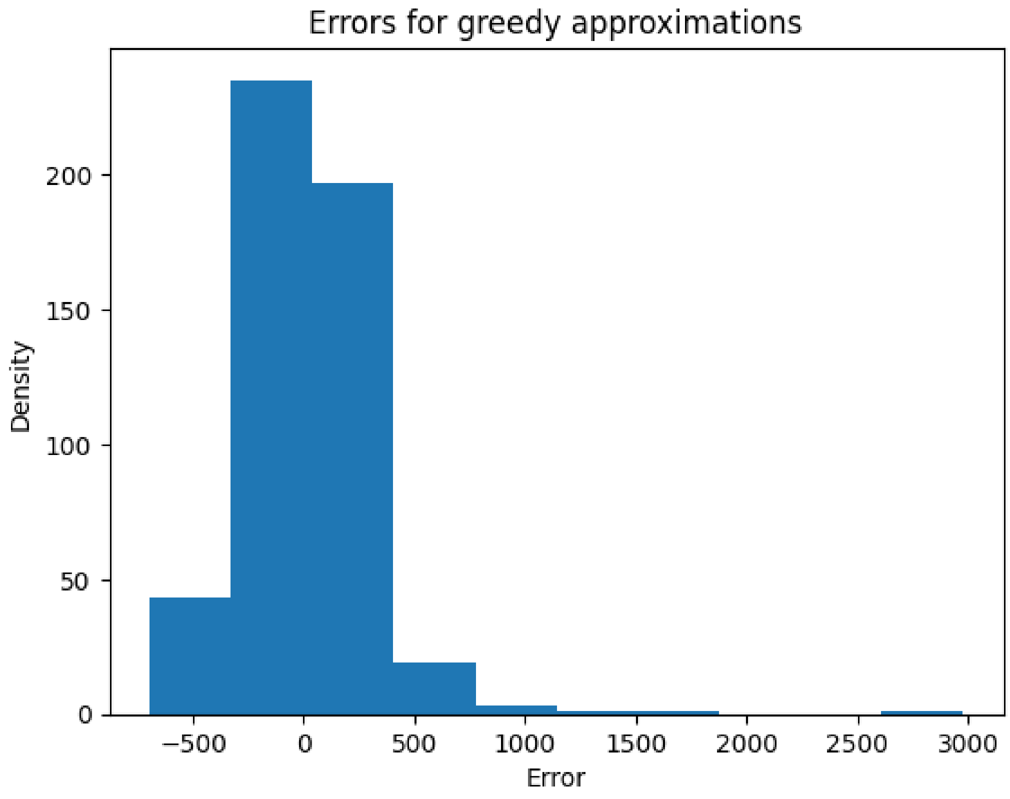

Figure 2: Histogram of Errors for Greedy Approximations.

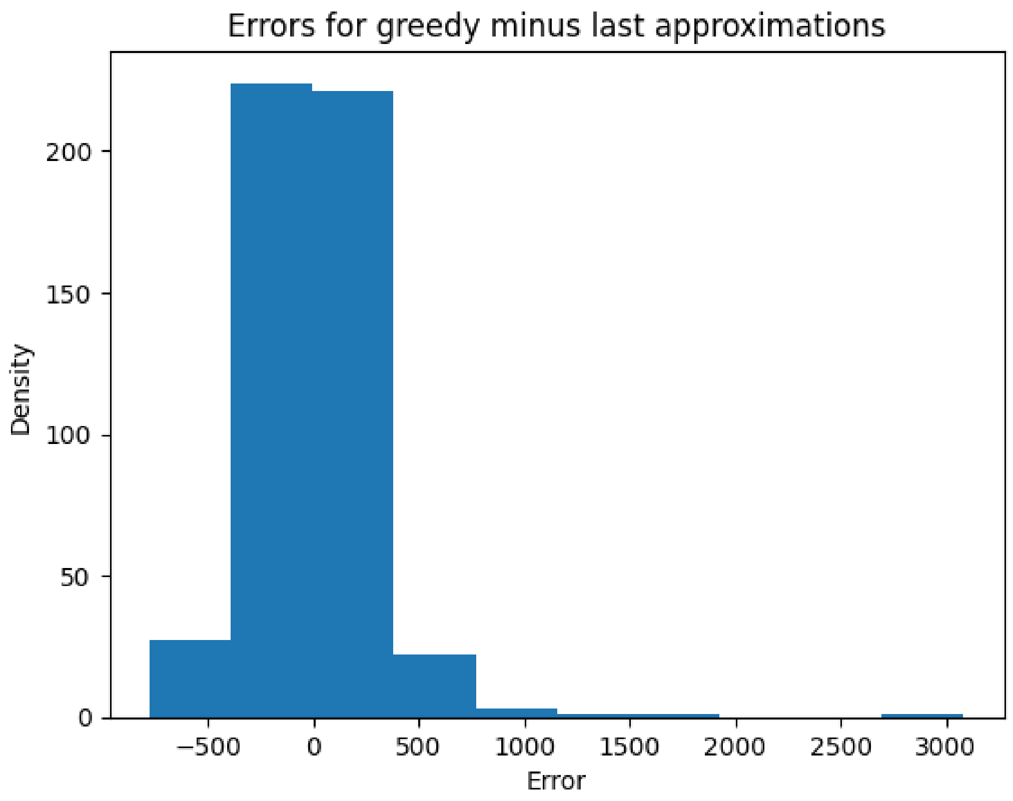

Figure 3: Histogram of Errors for Greedy Minus Last Approximations.

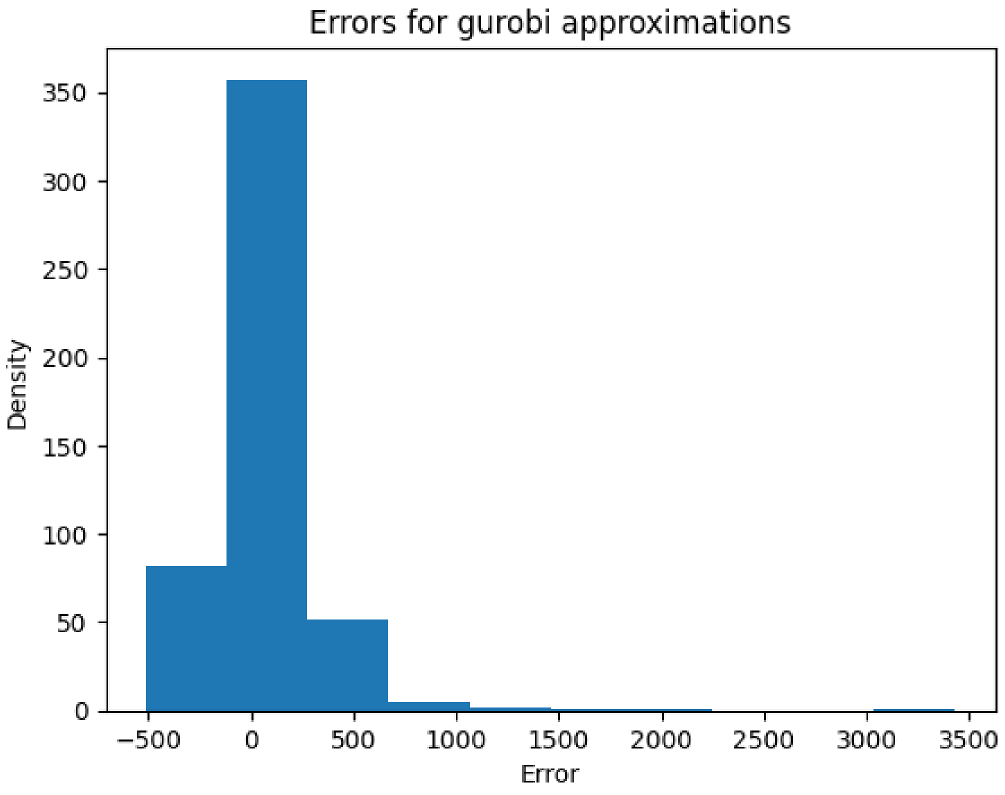

Figure 4: Histogram of Errors for Gurobi Approximations.

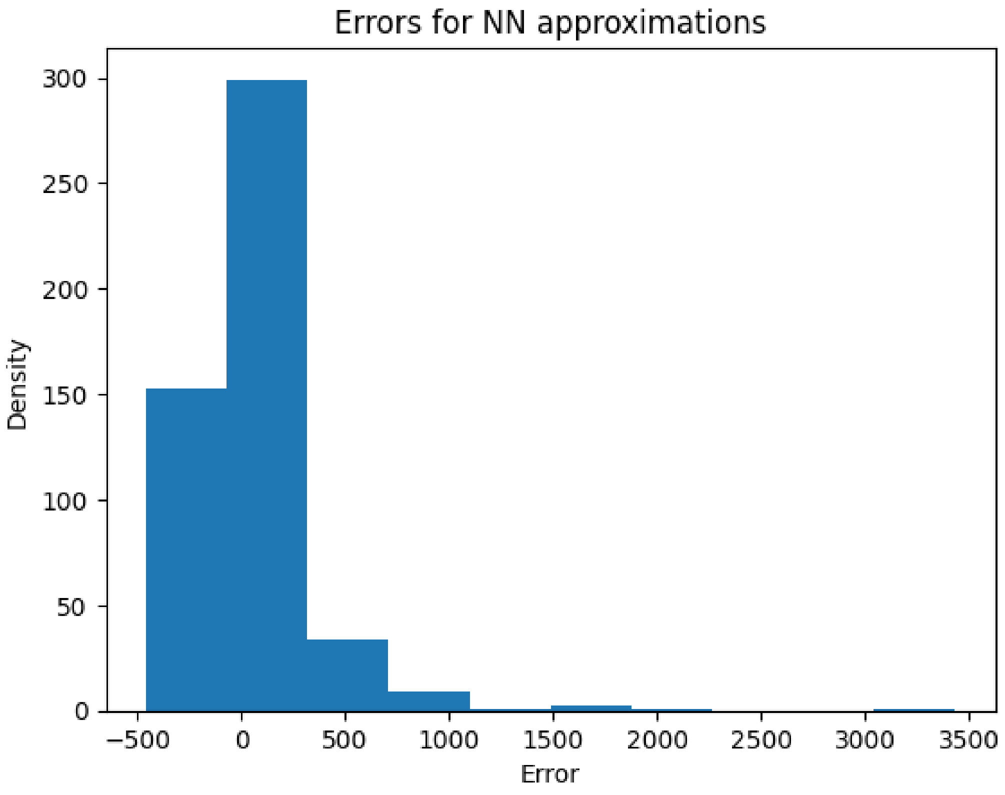

Figure 5: Histogram of Errors for Neural Network Approximations.