Attention-Based Learning for Combinatorial Optimization

by

Carson Smith

B.S. Computer Science and Engineering, Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2022

Author .....
Department of Electrical Engineering and Computer Science
May 6, 2022

Certified by .....
Hamsa Balakrishnan
Professor, Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Attention-Based Learning for Combinatorial Optimization

by

Carson Smith

Submitted to the Department of Electrical Engineering and Computer Science
on May 6, 2022, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Combinatorial optimization problems, such as the Traveling Salesman Problem (TSP), have been studied for decades. However, with the rise of reinforcement learning in recent years, many of these problems are being revisited as a way to gauge these new models in different environments. In this thesis, we explore the use of a new type of model, the Decision Transformer, which is a Self-Attention Transformer architecture that was recently developed for training on reinforcement learning problems. To analyze the model, we structure the Traveling Salesman problem as a reinforcement learning problem and, by continuously varying parameters of the environment, measure its generalizability and success in this environment. This thesis aims to conduct an initial study of applying Decision Transformers to combinatorial optimization problems.[1]

Thesis Supervisor: Hamsa Balakrishnan
Title: Professor, Aeronautics and Astronautics

[1]Research was sponsored by the United States Air Force Research Laboratory and the United States Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notion herein.

Acknowledgments

I would like to primarily thank Professor Hamsa Balakrishnan for her support and help with this thesis. She took me into the Puckboard team in my last semester of the program and spent considerable effort to get me up to speed. I would not have been able to complete this without her continuous help. I would also like to thank Siddarth Nayak for his help on defining and finding my way forward in this project.

I would also like to say thank you to my two dogs Lulu and Ohana for love and support for my past five years at MIT.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

The Air Force Crew scheduling process is currently an extremely rigorous and time intensive process. Crew schedules have to include pilot training requirements, mission requirements, availability, and schedule disruptions. Each individual schedule can take up to 27 person-hours to create, thus harming the Air Force's efficiency and taking time away from more meaningful duties for the airmen to complete.

Additionally, with this time-intensive and manual labor, even the most experienced of schedulers can run into problems with unforeseen disturbances or disruptions in estimated mission timelines. These can be caused by a variety of reasons but can result in major schedule changes or even previously created schedules to be discarded. Thus, the Air Force has begun to look towards optimization algorithms and machine learning in order to increase the efficiency of creation and the quality of the schedules produced, even if the these algorithms act solely as additional tools used by the schedulers.

However, given the complexity of the crew scheduling problem, this thesis considers a similar, yet simpler, combinatorial optimization problem. To this end, this thesis considers the traveling salesman problem (TSP), a classical combinatorial optimization problem that shares some characteristics with crew scheduling.

1.2 Background

Combinatorial optimization problems have been around for decades and have had almost countless different methods and algorithms applied to them [16]. Since the definition of combinatorial optimization is extremely broad: finding an optimal object from a finite set of objects, many problems and research fit into this field. This thesis focuses, throughout the background and literature review, on areas that specifically cover attention-based learning, the TSP problem, reinforcement learning, or some topics between them.

The traveling salesman problem is defined as, given a salesman and a network of cities, find the shortest route for the salesman to visit every city and return to the city that they started at. The TSP has been proved to be NP-Hard but not NP-Complete, which has led some researchers to explore the application of faster methods of solving [9] as brute force methods have extremely high polynomial complexities [5]. Instead of solving TSP problems through a brute force approach, much faster solutions have been found through the application of simulated annealing, an optimization technique to approximate the global minimum of a function [4]. These solutions, while providing a slight decrease in solution quality, run much faster, reduce time complexity to a polynomial performance [15].

Additionally, it is worth noting that the TSP-type problems have become more common-place with the rise of ride share and food delivery apps. Google Maps has even recently allowed users to add up to eight stops through their Maps UI [21]. This is intuitively the same as a small version of the TSP problem. Also, it is worth noting that this problem is also commonly referred to as the school bus routing problem or the vehicle routing problem in which the salesman is simply replaced by a vehicle and the cities are bus stops or destinations. In these cases, some formulations have even been used to develop routes for different schools and bus stations globally [11].

1.3 Prior Works

The first part of the prior works section is dedicated to crew scheduling problem. Even though the research in this thesis is not yet applied to the crew scheduling problem, this section will explain the explored areas and the gaps in that research that this thesis fills.

The second part will cover the background of transformer research and the decision transformer architecture used by this thesis. A majority of this will cover the creation of the field and how transformers came to be used for a problem such as this, while a technical review of the decision transformer will be covered in Chapter 3.

The final part will be a brief review of the research done on TSP problems and how they are primarily solved in the status quo. Additionally, we will discuss the recent developments in reinforcement learning and their applications to TSP and combinatorial optimization problems.

1.3.1 Crew Scheduling

In a sense, the crew scheduling problem itself is a combinatorial optimization problem, as we are looking for an optimal object (the schedule) from a finite set of objects (the pilots and missions). This section will serve to provide greater context on the potential weaknesses of past approaches, and motivate the work in this thesis

Two recent theses, by Matthew Koch and Chris Chin, have addressed crew scheduling problem [10] [3]. While both theses use an integer programming approach, they differ in the functionality that these algorithms were aiming to provide. The focus of Koch's thesis was pertaining to the increasingly complex qualifications and operational constraints, and training requirements of both missions and pilots. He presented optimization formulations over many objectives, such as overqualification on training requirements. These optimization functions were all able to accomplish individual goals and, in his recommendations, a user could choose a specific objective to fit a specific goal of the scheduler. While, Chin's thesis also focused on a integer programming approach, it incorporated further features to account for crew rest,

availability, and disruptions [3].

Finally, a recent paper presents NICE (Neural network IP Coefficient Extraction), which allows integer programming formulation to use the coefficients generated through reinforcement learning on schedules [12]. Through this novel formulation, NICE was able to produce significantly more robust schedules, handling large amounts of flight delays without many, if any, disruptions to the schedule as a whole. This method performs great, on the order of approximately 40%-45% at handling disruptions than the baseline integer programming formulations presented in the previous theses.

1.3.2 Transformers

Recurrent neural networks, such as long short-term memory networks (LSTMs) [6] have been used for sequence modeling, time series prediction, and language modeling, for many decades. These models all have a common and fundamental attention mechanism, which allows models to focus on specific elements of data and ignore others. These large and complex networks led the forefront of this research for many years. However, it was recently discovered that the attention mechanism of these models allow remarkably more parallelization between inputs and outputs while training and predicting outputs at a significantly faster pace [19]. Thus, through this discovery, Vaswani et al. developed the Transformer, a model architecture made up of entirely attention mechanisms. This model was able to beat the accuracy of many single-model state-of-the-art language translation with only a fraction of the training time.

Since their debut, transformers have been used in a variety of different research areas, with a great amount of success. They have been used to make realistic zero-shot caption to image generations that have beat long standing generation networks [14], demonstrating their ability to learn sequential semantics in language and apply them to generation. Similarly, after illustrating great success in NLP and computer vision applications, transformers naturally became a focus in time series modeling. Throughout these tests, transformers demonstrated a strong ability to recognize long-range dependencies, beating out many long-standing time-series prediction benchmarks [20].

From a transformers demonstrated ability to model long sequences, recognize temporal patterns, and perform credit assignment directly, it posed the question of whether a transformer would be able to effectively model in time-sequences of sparse data while maintaining greater generalization than other models. This question was the direct topic of a recent paper, which inspired the research of this thesis [2].

In this paper, Chen et al. use a transformer architecture to cast reinforcement learning problems as conditional sequence modeling [2]. They structure this problem by conditioning the model on states, actions, and rewards from random walks. Through this structure, the model is able to discover the shortest path, or path with the highest reward, through a given action graph. Through this training, the model performs extremely well, beating out benchmark learning strategies in Atari, OpenAI Gym, and Minigrid, all of which present extremely different problems and complexities. As mentioned previously, a much more thorough examination of the models architecture will be presented in Chapter 3 of this thesis, as we adapt it to the TSP problem.

1.3.3 Combinatorial Optimization

While the simulated annealing approach has also been used for many combinatorial optimization problems, there has been some research into the application of recurrent neural networks (RNN) to this area [18] [17]. While RNNs have not been able to achieve average errors as low as annealing, the results were promising and the speed of solving was greatly increased. Thus, this can simply be a further trade off of accuracy for extreme efficiency that might be desired by companies such as Google Map's users and even crew schedulers for their schedules.

In Mazyavkina et al. [13], the authors explore a broad set of combinatorial problems, including the TSP. They go on to define a very generalized architecture and method that they will use for solving. This paper is especially relevant to that of this thesis as it discusses the replacement of hand-crafted heuristics by the proposed reinforcement learning formulations. This is in a very similar vein to this thesis, which hopes to accomplish with the replacement, or augmentation, of the scheduler's, do-

main expert's, as defined by [13], own heuristics.

1.4 Outline

The outline of this thesis is as follows. In Chapter 2, we describe the problem environment that will be used with our model. This will cover the similarities between the TSP environment and the crew scheduling problem, and what we hope to discover about the model. Additionally, we will discuss how we generated the dataset used by the model. In Chapter 3, we will discuss, in a more technical sense, the transformer presented in the previous section. This will cover both inputs and outputs expected from the data, and the alterations that were needed to have the model work with our project. In Chapter 4, we will cover the metrics that were used to analyze the performance of the model along with a discussion of the findings. Finally, in Chapter 5, we will draw conclusions from the work and provide recommendations for further research in this area.

Chapter 2

Dataset Creation

For the purposes of this thesis, it was determined early on that we needed a simple problem, with enough parallels to that of the Air Force Crew scheduling problem, to determine if the Decision Transformer was a model that would be worth testing on crew scheduling data. The sample problem needed to be able to represent temporal constraints, similar to that of availability and mission timing, while also being able to represent individual requirements, such as the training requirements of each pilot.

2.1 Problem Selection

With the requirements for the sample problem in mind, an additional requirement for the model needed to be met. Since the model itself is designed to interact with OpenAI Gym [1], the sample problem would need to have an environment that is gym compatible, with the same step and state functions. In research, we found OR-gym, and open source library for creating reinforcement learning algorithms [7]. While some of the problems are more specific in their creation, such as packing and inventory management, the environment also includes more traditional operations research problems, such as the knapsack problem and TSP.

After examining all of the problems that were available in the OR-gym environment, the traveling salesman problem was selected. This was done due to it containing many similar, yet simplified, requirements when compared to the crew scheduling

problem. Further defined from Chapter 1, the traveling salesman problem proposes that, given n cities, with variable road lengths between any given city and another, determine the shortest route a *salesman* can travel in which every city is visited and they return to the city they started. This specific formulation, although on a different problem, has similar requirements to crew scheduling.

When looking at the crew scheduling problem from Chapter 1, we can identify two main requirements for a successful solution. Primarily, we have that of mission assignment. For every given mission inputted for the schedule, there must be a pilot assigned. Similarly, in the TSP problem, we must have that the salesman visits every single city. Additionally, each of the missions and pilots have different training requirements that must be met for a successful schedule. In the TSP, we loosely capture these requirements by negatively rewarding by the distances between all the cities. While there are many different paths that a *salesman* can take, there is only one optimal solution between the cities.

2.1.1 Problem Environment

For the environment itself, the interaction with the problem space is quite simple. There are three main variables that the model uses to interact with the problem. The first is the *state*, which is represented as an $N + 1$ array for a problem with N cities. The first index of this array indicates which city $\in (0, 1, ..., N - 1)$ the *salesman* is currently located at. The rest of the indices indicate the number of times the salesman has already visited that city. The second variable is that of *action*, which indicates the city the *salesman* will travel to after being at *state*. Finally is that of *reward*. *Reward* indicates the reinforcement reward amount that the salesman will receive by traveling to *action* from its current *state*. The reward is of the following structure: if the *salesman* is visiting a city that they have not already been to, their reward is $-1 * D$, where D is the distance between the two cities. If the salesman has already visited the city that they are going to, they will automatically have -100 added to their reward, in addition to D. Finally, once the *salesman* has visited all of the cities, they are given reward of 1000 in addition to D.

2.2 Random Walks

Finally, with the structure of the environment and TSP problem in mind, we will now cover the use of this environment to create the dataset used in our experiments. We first start by scraping 1,000 individual distance matrices, that represent the distance from any given city to another, from the OR-gym environment. From there, we handle each of the individual instances of the problem separately. We start by using both brute force to calculate the optimal solution for each problem.

Once this is completed, we perform random walks on each problem. These random walks have a few properties that are changed throughout the testing of the model. The first is the simplest, N, which is simply how many cities exist in a given problem. The second is that of iterations per problem, how many random walks are performed on a given distance matrix. We additionally have that of the starting node. In this thesis's experimentation, the starting node is both randomized and held constant to test generalization. To limit the length of a specific walk, we define the variable timesteps, which will cut off a random walk if has taken |timesteps| actions without visiting all of the cities. Finally we will randomly choose x percent in which the model is only able to choose cities that it has not been to previously. Additionally, to augment the random walk, we append the optimal solution, which we calculate using the TSP-Solve package [5], for the individual TSP problem, in the same format as the other random walks calculated previously.

For each random walk, we use these variables to collect five individual arrays that can be used by the model. The first is *observations*, which is an array of state variables, defined in Section 2.1.1. There is a single state variable for each time the *salesman* travels. It is worth mentioning that, in the current setup, a *salesman* can travel to the city that they are currently at. While there is no negative reward for travel distance, they will still receive the -100 reward for revisiting a city. The array is actions, which will be a one-dimensional array for each action that the *salesman* takes. Similarly, we will track an array of rewards, defined similarly as previously mentioned. We will also track a new variable in a one dimensional array, *terminals*. This array is True

at index i, if taking $actions[i]$ from $states[i]$ will result in the *salesman* visiting all nodes, and will return False everywhere else. Finally, the environment also has the ability to pass an action mask, an array of size $|cities|$, with each index being True if it has been visited, and False otherwise.

Chapter 3

Deterministic Decision Transformer

3.1 Decision Transformer

As discussed in Chapter 1, the work done in this thesis was largely inspired by the recent creation of the Decision Transformer by Chen et al.[2]. The architecture takes advantage of a casual transformer and positional embeddings to convert from a given state, action, and reward to the prediction of a next action. This section will summarize the technical aspects of the Decision Transformer paper [2].

3.1.1 Model Methodology

The idea of using a decision transformer came from the goal to replace traditional reinforcement learning algorithms with that of an attention block. Specifically, the paper focuses on offline reinforcement learning, in which the model is fed sub-optimal and noisy training data, as discussed in Chapter 2. Here, we aim to have the model discover optimal behavior, even from very limited experience. In Figure 3, we have an illustration from the Decision Transformer paper, applied to our TSP problem. This details how, given only random walks, the Decision Transformer is able to find optimal pathing around all of the nodes, by setting a prior to maximize rewards. Thus, the model should be able to use sequence modeling and prior information from the random walks to generate optimal solutions without using a brute force approach.

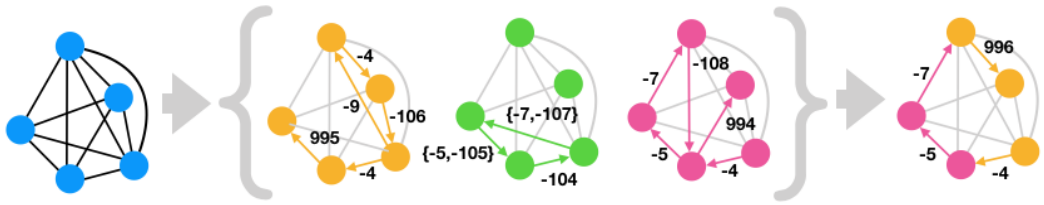


Figure 3-1: Decision Transformer adapted to the TSP, illustrating how the decision transformer explored the random walks during training to find the shortest reward path as reinforcement learning (adapted from [2])

3.1.2 Model Input

Here, as described in Chapter 2, the model needs to be fed a sequence of rewards, states, and actions. However, simply having the model learn off of modeled rewards is not substantial. Thus, the paper defines an input, returns to go (RTG), $\hat{R}_t = \sum_{t'=t}^{T} r_{t'}$. By using this, the model will be able to generate future desired returns when deciding upon an action to select. Altogether, the models combines RTG, states, and actions for training in a sequential format:

$$\tau = (\hat{R}_1, s_1, a_1, \hat{R}_1, s_1, a_1, ..., \hat{R}_T, s_T, a_T) \tag{3.1}$$

3.1.3 Model Architecture and Training

The model itself, defined by Chen et al., is made by stacking self-attention layers with residual connections between them. They specifically use the GPT2 architecture developed by OpenAI. Each of the layers receive n embeddings from unique input tokens and similarly output n embeddings to the next layer. Defined by Vaswani et al. [19], each individual embedding i produces a key, query, and value. These values are combine together to produce the output at i by computing:

$$z_i = \sum_{j=1}^{n} softmax((\langle q_i, k_{j'} \rangle)_{j'=1}^{n})_j \cdot v_j \tag{3.2}$$

Thus, combining this formula and the model input, from the previous section, the model begins by taking K timesteps, where each timestep includes one token for

each of the inputs; RTG, state, and action. To calculate the token embeddings, the model learns a linear layer for each of the three and projects these raw inputs to the embedding. We also have that, for any given timestep less than the max time length, the model learns an individual embedding and adds it to each token.

To practically learn all of this, the model defines a training portion in which a batch size of K is fed into the model. Each of the individual random walks, τ, is not fed into the model entirely. Instead, the model takes a subset of the sequences from index i, such that $0 < i < T$, producing a subsequence $\tau[i : T]$. From these sequences, we have the prediction head of the model, corresponding to the state, learn to predict the action of the same timestep. This is done, in the Gym environment this paper implements, using mean-squared error.

3.2 Model Alterations

While Chen et al. achieve results, rivaling those of traditional reinforcement learning algorithms in many environments, this model will not be able to solve the TSP on the OR gym environment out-of-the-box. Thus, changes were made to the architecture and training of the model.

3.2.1 Architecture Changes

As detailed above, the model to begin this thesis was originally used on the OpenAI Gym environments. For all of these environments that this code was tested on, the output and inputs were continuous values. Thus, in its current state, the model would not be able to correctly output a discrete action or action probabilities to give a new action during evaluation.

To fix this on our problem, we need to change the final layer of the model. It is currently predicting individual continuous values that correspond to different outputs for the various environments that were tested. Although we need the model to output one action, we also need the model to be able to return probabilities for all actions in the action space. Thus, we need to incorporate the vocab size, which defines the

different tokens that can be represented in the dataset. This is crucial as we are moving to a model with specific actions, and thus the model needs a way to represent them.

Additionally, we need to change the output from the last linear layer to be the size we now are representing. In this case, we only have one action that can be any of the n cities in the problem. Thus, we need to print n different probabilities where n_i is the probability that the *salesman* should choose city i. After that, we choose the action with the highest probability as our selected action. This will also allow the format of the data to match up correctly during training.

We additionally experimented with making the last layer of the model a softmax layer, as these layers are the most popular to use with multi-class classification. The effects of this will be discussed in later sections.

3.2.2 Training Changes

In Section 3.1.3, it is mentioned that the Decision Transformer continuously trains using mean-squared error (MSE) formula to calculate loss. However, MSE does not work for classification problems such as these [8]. This stems from the fact that MSE assumes that the data was generated on a normal distribution, whereas a multi-class problem is not generated with a Gaussian Prior and can't be placed on that distribution. Here in the training, given that we are now prediction action probabilities, this thesis changes the loss function that that of cross entropy loss. Thus, for all probabilities of an action x, $Pr(x)$, and actual chosen actions, $Q(x)$, we have:

$$L_{CE} = -\sum_{x} Q(x) log(Pr(x)) \tag{3.3}$$

This equation was chosen as we want the model to be able to predict the chosen action and thus minimize the difference between the probability of the selected action and the correct one.

Chapter 4

Experiments and Findings

This chapter discusses the various experiments that were completed throughout testing. We will describe the tests in terms of the variables defined in Chapter 2. After discussing the different tests that were completed, we will break down the findings in terms of the Loss, Accuracy, and Computational Efficiency so that we can analyze the success or failure of individual results in different areas.

4.1 Experiment Setup

To define each of these experiments, we will define the problems in terms of N, the number of cities that exist, S_n, how we choose to select the starting node for the problem, L the maximum length of the random walk and x, the amount of random walks that are limited to choosing only cities that the *salesman* has not visited yet.

4.1.1 Variable Changes

First, it is worth mentioning that, as described in Chapter 2, this thesis had scraped 1,000 individual distance matrices from OR Gym. However, initial testing and model ideology showed that training on the variable distance matrices is not optimal. To understand the reasoning behind this, consider Figure 3-1. If the model attempts to learn from all these random walks and each city-city connection has different reward

values, it will be very difficult for the model to generalize to this. Thus, for the testing, we vary the distance matrix between individual experiments, but not within an experiment.

For the varying N, we ran experiments that contained both 5 cities and 15 cities. This allowed us to see the limitations of the model, and whether or not the model could generalize to larger cases or not. It is worth mentioning that a majority of testing was completed on the networks with 5 cities.

Additionally, through varying S_n, we calculated experiments in which the random node was held constant vs. variable. We initially started the testing by continuously selecting a random starting node for both the random walks and the evaluation. However, this thesis argues that, by setting the starting node to a constant value, the model would be able to determine a more distinct starting and goal node, thus improving model performance.

For both cases of N, we tested them by varying the length L between two values. For both, we can define these length as $\{N, 3N\}$. This was to determine if the model was limited to only training on runs of optimal length, though not optimal actions, would it learn the optimal actions for that specific length.

Finally, we chose the number of random walks in which the no repeat actions can be taken, x, to be either 25% or 75% of the random walks used to train. Initially, this variable was just created to ensure that the model would have enough coverage of the distance matrix; however, we then theorized that, by witnessing more optimal walks, the model would be able to use the returns-to-go arrays to better understand the TSP problem.

4.2 Metrics

Now that we understand the the environments that the model will be training on, we discuss the performance that the model achieved throughout the different variations. In this section we start by analyzing the loss, since its important position in actually configuring the model. Then we will move to reward to get a better picture of how

the model performed. Finally, we will discuss the problem in terms of efficiency to analyze the benefit of using this approach over brute force and simulated annealing. As mentioned before, even though testing was completed on networks with 15 nodes, this section will only present and discuss the findings of networks with 5 nodes. This is done to simplify the analysis.

4.2.1 Loss

When analyzing the performance of the model, or more the lack thereof, it is important to start with loss as this is what helps train the model. The loss in this case, as presented in Section 3.2.2, is calculated between the chosen action and the predicted action. The model itself predicts the action given the current state and the RTG. In the case of the TSP setup, this can lead to easy errors for the model. The reason behind this is two fold.

Primarily, the model itself will have a difficult time determining where the reported negative reward will be coming from. To illustrate this, consider the following example: let us define an state, $s' = [1, 1, 1, 1, 0, 0]$ and a corresponding reward, $r' = 685$. From this, we know a few nodes are revisited and that nodes 4 and 5 are visited at some point to achieve the +1000 reward; however, determining the exact order is incredibly difficult for the model due to the number of actions and series that are possible.

Additionally, the model will accidentally punish action choices that are still very accurate. Consider a state in which cities 4 and 5 are not visited yet and a RTG of ~ 1000. In this case, we can fairly easily see that both nodes are visited as the next two actions. However, the order can either be 4-5 or 5-4. In the case that the model predicts the opposite order. Even with an optimal prediction of .5 for either actions, the model will receive significant loss.

After thorough testing, it seems as though these theories are empirically backed as well. We calculate both the action error and the training loss and the action error that the model received over all permutations of the 5 node case. As you can see from Figures 4-1 and 4-2, the loss mean, while variable between experiments, is

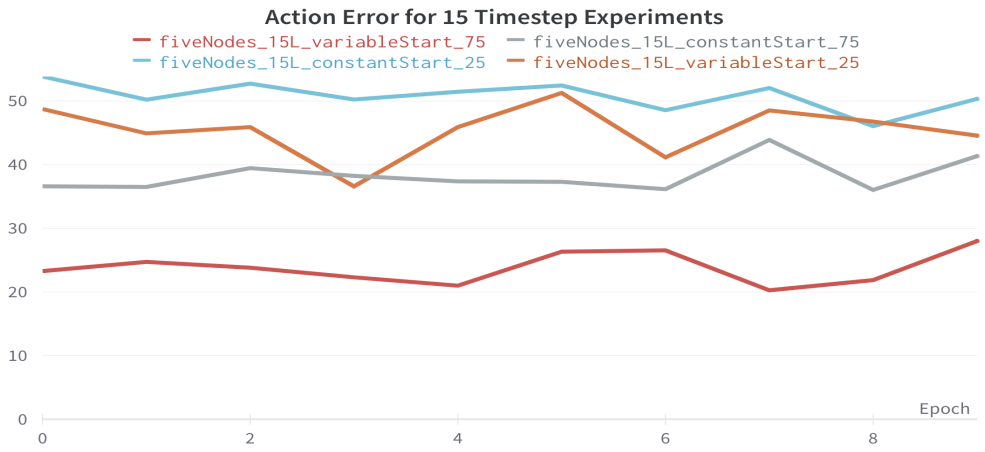


Figure 4-1: Action loss for all experiments limited to 15 timesteps

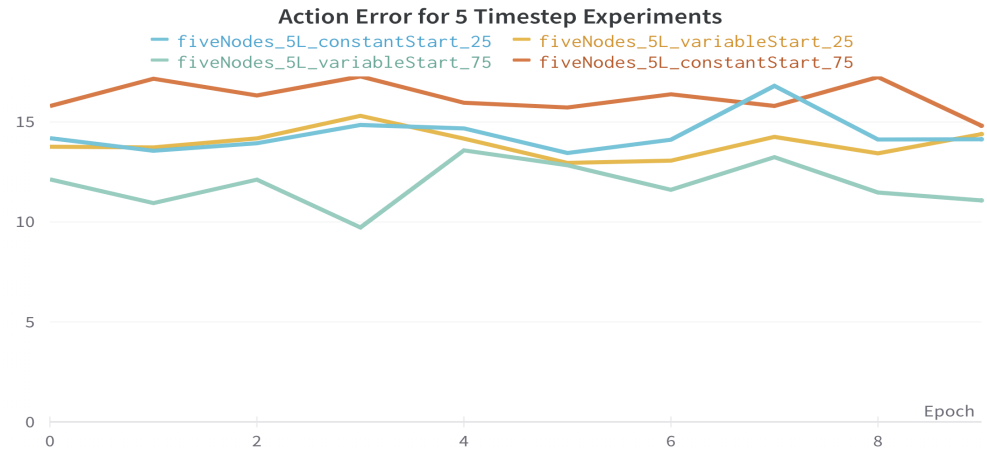


Figure 4-2: Action loss for all experiments limited to 5 timesteps

fairly constant across all epochs. This tells us that the model, no matter what the performance reflects, is not learning during the training iterations.

Thus, with the current formalization of loss from the Decision Transformer, continuously decreasing the loss of this model during the training epochs is very difficult. And without being able to do so, the model won't be able to configure its attention blocks correctly, leading to an inaccurate model.

4.2.2 Reward

As discussed in the previous section, when we look at the loss, we can not expect promising results for the reward either. As you can see from Figures 4-3 and 4-4,

Figure 4-3: Average reward (Left) and standard deviation (right) for experiments limited to 15 timesteps

Figure 4-4: Average reward (Left) and standard deviation (right) for experiments limited to 5 timesteps

we see extremely low rewards across the board, considering that the optimal reward is $\sim$ 998. Additionally, over all of the cases, we can see extremely high standard deviation meaning that, during the evaluation, the performance is close to random selection. It is worth noting that, in the 5 timestep experiments, the average rewards and standard deviations are much smaller and closer to each other. This is solely due to the fact that the evaluation iterations are also limited to 5 timesteps.

However, there are some interesting takeaways from this training reward. Specifically looking at the variable starting experiment with 25% optimal limit in Figure 4-3. Here we see that the reward is considerably higher than the other examples and the standard deviation much lower. Throughout other initial experiments, that unfortunately weren't tracked, we saw some random behavior that was similar to this

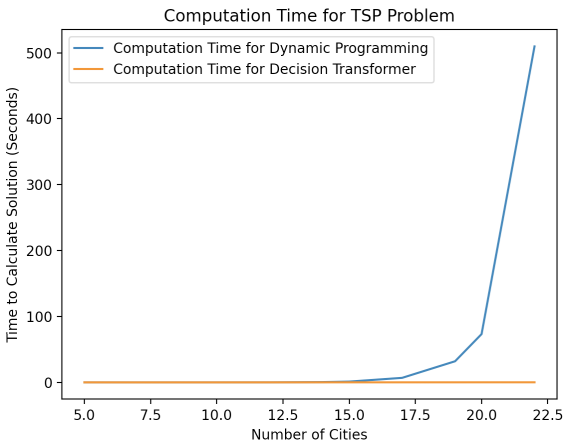
and even achieved much higher average rewards (upwards of 700 average reward). This tells us that the model might have a propensity to learn this environment with some changes.

Another interesting pattern we can see is the, generally, inversely correlated Reward and Standard Deviation, which holds for all experiments except for the variable start with 75% optimal limit in Figure 4-3. This can possibly point to a few conclusions of the model that we will discuss in the final section of this chapter.

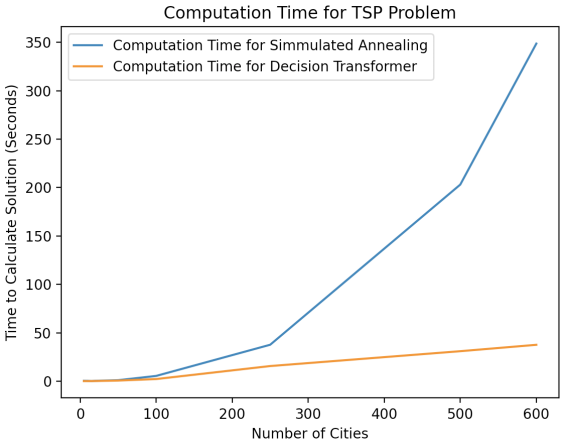
4.2.3 Computational Efficiency

The one advantage that this thesis found throughout experimentation was the computational efficiency of the Decision Transformer. Throughout testing, and through following logical reasoning, the Decision Transformer scales with pseudo-linear time as it needs to only predict once per node in the route (the model does slightly increase prediction time as more nodes are added and the model grows in size). However, throughout testing of the TSP solver in both dynamic programming and simulated annealing we witness polynomial and exponential increases with polynomial and exponential complexity, respectively.

Specifically, we have that Dynamic Programming scales with a complexity of $O(2^n * n)$ and simulated annealing scales with a complexity of $O(n^4)$ [5] [15]. Thus, this approach, if good reward can be achieved, will potentially be able to rival that of simulated annealing as a possible way to achieve quick speeds for TSP and other combinatorial optimization problems. The best illustration of this is through analysis of Figure 4-5. In this Figure, we can see the extremely consistent increase in computation time for the Decision Transformer vs the standard methods. Even simulated annealing began increasing to impractical times as the network increased past 500 cities.



(a) Dynamic Programming and the Decision Transformer



(b) Simulated Annealing and Decision Transformers

Figure 4-5: Computational Performance between Decision Transformer (both), Dynamic Programming (left), and Simulated Annealing (right). Comparison for Dynamic Programming is scaled significantly down due to inability to efficiently compute complex examples.

4.3 Findings Discussion

From the perspective of this thesis, we are able to draw findings from the work analyzed in this section. While the model wasn't able to achieve a highly consistent reward or demonstrate the ability to learn from decreasing loss, the different experiments did illustrate some key properties of the model that have potential.

Primarily, if we look at the average rewards in Figures 4-3 and 4-4, we notice that there are varying rewards, even performing with an average positive reward, in the orange line in Figure 4-3. This experiment also had the lowest standard deviation (scaled) out of all of them.

This thesis hypothesizes that while the model wasn't able to directly learn from the current loss function, that it might be aware of when it is performing well. This is backed both by the inverse correlation between rewards and standard deviation, indicating that as it does better, the model becomes more sure of what to do in specific situations. Additionally, the fact that the model stays constant suggests that it is either over fitting, which was checked to be false by thoroughly varying the learning rate of the model, or that the reward is just set to the randomized initial

state of the model. If the second case is true, then some of the results are promising for the future of the model. Since, in this case, as we have seen high average rewards throughout testing, it suggests that the model has the ability to get to that point if it is restructured to learn.

All of these problems most likely stem from Section 4.2.1. The fact that the model doesn't learn the TSP problem at all points to either a failure at the architectural level of the model or that of the loss function. Thus, this thesis acknowledges that a much more thorough analysis of the architecture and potentially the use or creation of another loss function more specifically made for this problem is required to improve performance.

Finally, as we discussed in Section 4.2.3, this model outperforms many current TSP algorithms in term of computational efficiency. Specifically, if we were to greatly improve the performance of this model, we would be able to apply it to very complicated TSP problems (>500 nodes) with much more efficiency. While this would probably not be realistically necessary for the TSP problem individually, it can easily be applied to different algorithms such as a mapping function for driving or planning logistical deliveries for storage trucks that have complicated schedules.

Chapter 5

Conclusions

The implementation of the reinforcement learning techniques on combinatorial optimization problems will continue to be a primary focus of research as newer and more complex models are created. In this thesis, we presented an initial exploration of the Decision Transformer, one of the newest RL configurations, applied to the Traveling Salesman Problem. We presented the basic architecture of this model and laid out how we augmented this model to our environment. We then presented the various experiments we ran on the model and the metrics we used to analyze each of these experiments. Finally we discussed the results of these metrics and possible conclusions we can take away from them.

Although these experiments did not lead to largely successful results, we believe that the results suggest an ability for this model to work. These conclusions were mostly drawn from the ability of the model to become more deterministic as the average reward increased, and the fact that the random initialization of the model could actually return substantial returns without any beneficial training done.

5.1 Future Works

With these conclusions in mind, we believe that there exists substantial further work that can be completed to determine whether or not the Decision Transformer can work on combinatorial optimization problems and further application to that of crew

scheduling.

- Primarily, the research needs to determine the limit to the Decision Transformer's learning. This can possible be resolved by creating a custom loss function, redoing the final activation layers, or changing the sizing for the attention and linear layers that make up the model.

- Additionally, the dataset that was experimented with could be leading to potential problems. For example, the +1000 reward for finishing could be substantially increased proportionally to the negative reward to allow the model more incentive to seek it out. Also, we can revert the dataset to the original environment such that the exploration is canceled whenever a node is revisited instead of providing -100 reward. This could simplify the dataset considerably.

- Finally, there can be further experiment into the various environments that the Decision Transformer. For example, the Decision Transformer was also experimented with in the Atari environment and has a separate base code for that. The configuration and architecture in these environments are declared substantially different than that of OpenAI-Gym's. Thus, that architecture might be able to produce more fruitful results.

Bibliography

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *CoRR*, abs/2106.01345, 2021.

[3] Christopher Ho-Yen Chin. Disruptions and robustness in air force crew scheduling. 2021.

[4] Bruce L. Golden Christopher C. Skiscim. Optimization by simulated annealing: A preliminary computational study for the tsp, 1983.

[5] Fillipe Goulart. Python tsp, 2020.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[7] Christian D. Hubbs, Hector D. Perez, Owais Sarwar, Nikolaos V. Sahinidis, Ignacio E. Grossmann, and John M. Wassick. Or-gym: A reinforcement learning library for operations research problems, 2020.

[8] Rafay Khan. Why using mean squared error(mse) cost function for binary classification is a bad idea? 2019.

[9] Evan Klitzke. The traveling salesman problem is not np-complete, 2017.

[10] Matthew J Koch. Air force crew scheduling: An integer optimization approach. 2021.

[11] L. Y. O. Li and Z. Fu. The school bus routing problem: A case study. *The Journal of the Operational Research Society*, 53(5):552–558, 2002.

[12] Christopher Chin Hamsa Balakrishnan Luke Kenworthy, Siddharth Nayak. Nice: Robust scheduling through reinforcement learning-guided integer programming. 2022.

[13] Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey, 2020.

[14] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation, 2021.

[15] Galen Sasaki. Optimization by simulated annealing: A time-complexity analysis. 1987.

[16] Alexander Schrijver. *Combinatorial Optimization*. 2003.

[17] Paulo Siqueira, Sergio Scheer, and Maria Steiner. *A Recurrent Neural Network to Traveling Salesman Problem*. 09 2008.

[18] Mikhail S. Tarkov. Solving the traveling salesman problem using a recurrent neural network, 2015.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[20] Qingsong Wen, Tian Zhou, Chaoli Zhang, Weiqi Chen, Ziqing Ma, Junchi Yan, and Liang Sun. Transformers in time series: A survey, 2022.

[21] Dale Williams. How to optimize delivery routes with google maps, 2022.