Deep Learning Fall 2019 - Project Report
ConvNet-Pointer Networks for the Traveling Salesman Problem

Meziane Chauvin * **Thomas Gaillard** * **Maxime Laharrague** *

Abstract

This paper introduces an approach for trying to solve the Traveling Salesman Problem with graph convolutional and sequence-to-sequence networks. The first step relies mostly on an approach followed by C. K. Joshi, T. Laurent, and X. Bresson [1] aiming at building efficient TSP graph representations with Graph ConvNets, on top of which we tried to implement a structure described as Pointer Network by O. Vinyals, M. Fortunato, N. Jaitly [2]. This project was conducted in the context of a Deep Learning course taught by Pr. Iddo Drori from the Columbia University Computer Science Department.

1. Introduction

The Traveling Salesman Problem (TSP) is one of the most intensively studied combinatorial optimization problems in the Operations Research community. It can formally be expressed as finding, within a graph, the space of permutations leading to an optimal sequence of nodes (a 'tour') with minimal total edge weights ('length'). While the scientific community has already produced handcrafted algorithmic solutions to this problem, formulating it as a sequence of decision-making tasks on graphs with a highly structured nature urges towards the use of new graph neural network techniques.

Recently proposed deep learning approaches for solving the 2D Euclidean TSP include a graph convolutional neural network technique proposed a few months ago by C. K. Joshi, T. Laurent, and X. Bresson [1]. The underlying idea is to take 2D graph representations (ie points space coordinates) and output a probability matrix (a heatmap) reporting the probabilities for each node to be connected to another node in a TSP path. They relied on various heuristics including greedy search, beam search and shortest tours to then single out the best valid path, which is not implemented in the work presented here.

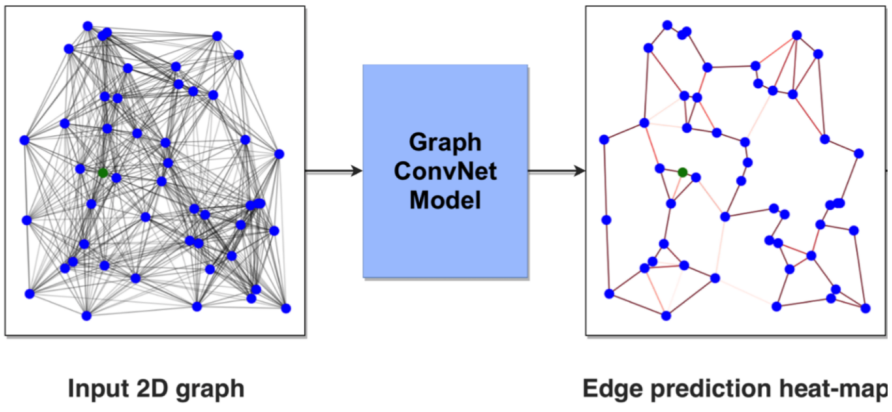

Figure 1. Initial structure from *Chaitanya K. Joshi et al.*

Instead of a heuristic, our approach aims at learning ordering relationships straight from the heatmap using a neural sequence-to-sequence architecture called Pointer Network brought up by O. Vinyals, M. Fortunato and N. Jaitly in 2017 [2]. Derived from Recurrent Neural Networks, seq-to-seq architectures are built on one RNN to map an input sequence to an embedding, and another (possibly the same) RNN to map the embedding to an output sequence.

But one limitation of these structures is that they require the size of the output dictionary to be known a priori, which is limiting in the case of the TSP as we want to be able to control the number of nodes to solve the TSP for as input. Pointer Networks deal with this issue by using at each iteration of the encoding RNN a softmax distribution - pointer - with dictionary size equals to the size of the input.

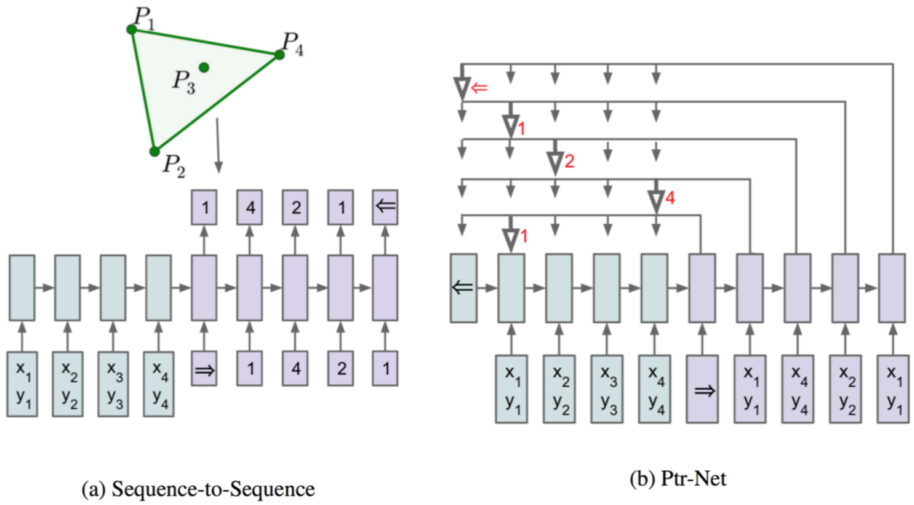

Figure 2. From *Oriol Vinyals et al.*. **a.** The classic RNN structure only allows same structure across inputs and outputs. **b.** Pointer networks are encoding and decoding inputs with 2 consecutive RNNs, then modulating a content-based attention mechanism.

Often used simply to sort 1D numerical sequences, Pointer Networks can also be implemented as direct solutions of the TSP. In this project, we will use Pointer Networks as an alternative to the beam search implemented in [1], which derives the most likely path based on the probability heatmaps returned from the graph ConvNet model.

2. Related Work

Due to its various practical applications in an array of modern industries (transportation, supply chain, energy, finance, etc.), there has been extensive research on optimizing solutions to the Traveling Salesman Problem in both the combinatorial and deep learning communities. Finding the optimal TSP solution is NP-hard, even in the 2D Euclidean case where the nodes are 2D points and edge weights are Euclidean distances between pairs of points [Papadimitriou, 1977]. In practice, TSP solvers rely on carefully handcrafted heuristics to guide their search procedures for finding approximate solutions efficiently for graphs with thousands of nodes.

Today, state-of-the-art TSP solvers such as Concorde [Applegate et al., 2006] make use of cutting plane algorithms to iteratively solve linear programming relaxations of the TSP in addition to a branch-and-bound approach that reduces the solution search space. Thus, designing good heuristics for combinatorial optimization problems often requires significant specialized knowledge and years of research work. Due to the highly structured nature of these problems, neural networks have been used

to learn approximate policies instead, especially for problems that are non-trivial to design heuristics for [Bengio et al., 2018].

Recurrent Neural Networks (RNNs) have been used for learning functions over sequences from examples for more than three decades [Rumelhart, Hinton and Williams, 1985]. However, their architecture limited them to settings where the inputs and outputs were available at a fixed frame rate. More recently, Bahdanau et. al. [2014] augmented the decoder used in sequence-to-sequence models by propagating extra contextual information from the input using a content-based attentional mechanism. These developments have made it possible to apply RNNs to new domains, achieving state-of-the-art results in core problems in natural language processing such as translation [Sutskever, Vinyals, and Le, 2014] and parsing [Vinyals, Kaiser, Koo, Petrov, Sutskever, and Hinton, 2014].

3. Dataset & Methods

In this paper, we focus on the 2D Euclidean TSP which can be modeled as a sequence of n cities (nodes) in the two dimensional unit square $S = \{x_i\}, 1 <= i <= n$ where each $x_i \in [0, 1]^2$, we are concerned with finding a permutation of the points $\hat{\pi}$ - a tour - that visits each node once and has the minimum total length. The solution is a permutation π of the nodes such that the length is minimal:

$$L(\hat{\pi}|s) = ||\mathbf{x}_{\hat{\pi}(\mathbf{n})} - \mathbf{x}_{\hat{\pi}(\mathbf{1})}|| + \sum_{\mathbf{i=1}}^{\mathbf{n-1}} ||\mathbf{x}_{\hat{\pi}(\mathbf{i})} - \mathbf{x}_{\hat{\pi}(\mathbf{i+1})}||$$

One extension of our work has to do with the norm used to compute a tour length. While most former approaches relied quasi-exclusively on the Euclidean distance, we decided to make it possible to use any norm defined in a two-dimensional vector space.

We generated TSP instances for $n = 10, 20$ and 50 nodes by randomly sampling point coordinates on the unit circle, and we obtained optimal path solutions for these instances using Concorde solver [Applegate et al., 2006].

4. Model Architecture

4.1. ConvNet architecture

Given a graph as input, we first train a graph convolutional network to output probabilistic matrices. Each edge is being mapped to a vector of probabilities standing for its likelihood to be associated with each node. The edge representations are linked to the ground-truth TSP tour through a softmax output layer, and the model is trained by minimizing the cross-entropy loss via gradient descent.

The ConvNet structure consists of the following layers:

- **Input Layer**: This layer takes the nodes coordinates as inputs and embeds them into h dimensions. The same mechanism is applied to edges features.
- **GCN Layers**: For each of them, a combination of dense attention map (core element of seq-to-seq), sigmoid, ReLU activation and batch normalization is applied on top of the input layer.
- **Multi-Layer Perceptron**: The edge embedding of the last layer is used to compute the probability of that edge being connected in the TSP tour of the graph. This probability can be seen as the step when the probabilistic heatmap is computed.

4.2. Ptr-Net architecture

This network receives the heatmaps from the ConvNet model as input and returns the associated valid tour as output. As explained below, this pointer model is trained by minimizing a certain loss, build as a combination of the cross entropy loss and a complimentary term, to prevent the model to revisit the same node several times.

This network is made of the following elements:

- **Encoding GRU**: Taking the heatmap probability matrix as input and embedding it to a sequence of vectors of hidden dimension h
- **Decoding GRU**: Generative network that produces the output symbols from the encoded representations.
- **Content Based Input Attention Layer**: Taking a blend combination of the decoder steps, this module applies a tanh cell activation followed by a softmax normalization on top of it.

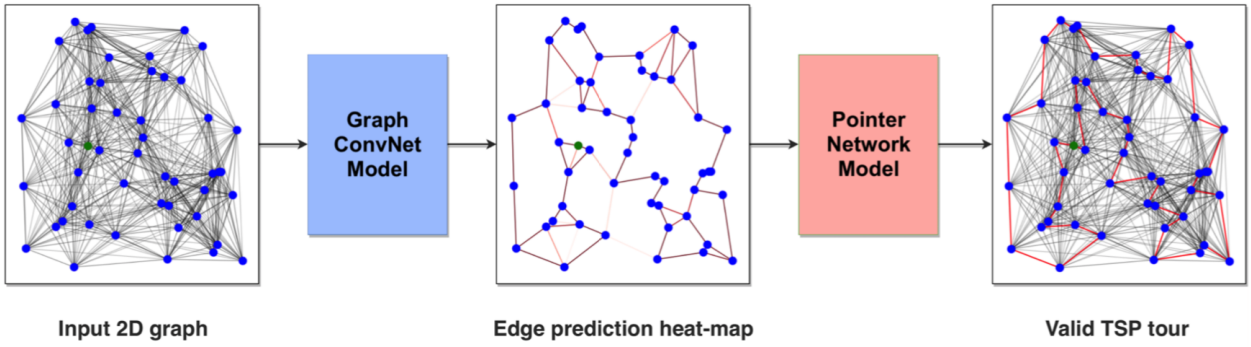

Figure 3. Logical flow of our proposed architecture. Heatmaps generated from GraphConvNet *(Chaitanya K. Joshi et al.)* are fed into the pointer networks in order to return valid tours, learned from the data.

5. Implementation

5.1. Adjusted loss implementation

When we substituted the beam search previously implemented by C. K. Joshi et al with a pointer network, the new model was not able to learn properly on the heatmap data and to return an appropriate predicted tour. Instead, the pointer network was generally returning unvalid tours with same nodes repeated more than once, and with missing nodes. To fix this and force the model towards considering viable tours, we decided to adjust the loss function by adding a complementary term. Now, this new term is designed to measure how far the predicted tour is from a valid tour, while the initial term in the loss is already accounting for the accuracy of the prediction.
Indeed, the initial loss is the cross entropy function of the output against the groundtruth. This loss is essentially summing up negative logs of predicted probabilities for each node in the tour, and is mainly used for classification problems.

On the other hand, our additional term is not focusing on accuracy of predictions but rather on how valid the tour is. In order to do so, and for the loss function to stay differentiable, we applied matrix transformations from the probability matrix P, returned by the pointer network for a given graph. On each column j, this matrix contains the probabilities returned by the pointer network for all N nodes to appear at j^{th} place in the tour. Now, taking the argmax of each column of P outputs the list of ordered predicted nodes. Below is an example of a potential matrix P returned by the model, for a 4-node graph:

$$\begin{bmatrix} 0.1 & 0 & 1 & 0.251 \\ 0.1 & 0.99 & 0 & 0.25 \\ 0.2 & 0.01 & 0 & 0.25 \\ 0.6 & 0 & 0 & 0.249 \end{bmatrix}$$

and the respective predicted tour would therefore be $\pi = [3, 1, 0, 2]$.

Since we require each node to be used exactly once, we would need P to be the closest as possible to a permutation matrix. Indeed, a permutation matrix has exactly one entry of 1 in each row and each column and 0s elsewhere. Setting this as our objective, we decided to measure how close P was from a permutation matrix by using orthogonal matrices' property. We defined the additional loss term as follows:

$$l_P = ||P^T P - I_N||$$

Indeed, this term is supposed to be zero if P is a permutation matrix (therefore an orthogonal matrix).

6. Further Development

6.1. Analogy with game strategy algorithms

Our tentative to map a Pointer Network to learn the optimal path directly from the heatmaps can be summarized as an attempt to make the algorithm learn the "rules" of the TSP without prior knowledge of the problem. In its conception, this is analog to the initiatives that have been taken to extend the performance of best game-solving AI agents to situations which they would have to learn the game environment from scratch.

With that idea in mind, Google DeepMind recently published a paper unveiling a 2.0 version of AlphaZero called MuZero, which can master several strategy games by learning the rules by itself. Relying on model-based reinforcement learning, MuZero focuses precisely on scenarios in which it needs to understand a new environment prior to mastering specific tasks within it. MuZero's planning process is based on two separate components: a **simulator** implements the rules of the game, which are used to update the state of the game while traversing the search tree; and a **neural network** jointly predicts the corresponding policy and value of a board position produced by the simulator. The model receives the observation (e.g. an image of the Go board or the Atari screen) as an input and transforms it into a hidden state. The hidden state is then updated iteratively by a recurrent process that receives the previous hidden state and a hypothetical next action. At every one of these steps the model predicts the policy (e.g. the move to play), value function (e.g. the predicted winner), and immediate reward (e.g. the points scored by playing a move). The model is trained end-to-end, with the sole objective of accurately estimating these three important quantities, so as to match the improved estimates of policy and value generated by search as well as the observed reward.

6.2. Solving the TSP as a game-reward problem

We believe that the Traveling Salesman Problem has all the required characteristics to be solved with a game strategy algorithm. It has states - associated with a cost - and rules to be learned at every step and can be modeled as a Markov-decision process.

Let us represent by $a_{k_1,k_2,\ldots,k_n}$ the running total length of the path when passing by nodes $1, 2, ...n$ in that order. From that state, we can consider every possibility consisting of going to node $k_m, m \notin [1, n]$, each of them associated to a cost c_m and leading a state $a_{k_1,k_2,\ldots,k_n,k_m}$. At this stage, MuZero would explore all possible nodes from this state while trying to maximize its reward, most likely through a Monte-Carlo Tree Search. This search would output at the end a recommended policy with a given value.

With that approach, it becomes possible to search over hypothetical future trajectories from a given known state and given past observations. The analogy with the TSP makes all the more sense that it is exactly the task that we are trying to teach the Pointer Network to learn from heatmaps.

6.3. Suggested Architecture for TSP solving

An idea of structure we could follow to solve TSP with a model-based reinforcement learning approach is describe in the following.

- Each state is defined by the list of nodes with which the current optimal tour is computed.
- At each step, MCTS is performed iteratively to explore all possible tours and output the best one.
- Implementing a Convolutional Neural network to train and learn the behavior of the player and outcomes of the game, without knowing the rules.

As a loss for the CNN, we could reuse the term we defined in part 5, aiming at penalizing tours which are not permutations. We could add it to the 3 usual terms of model-based reinforcement learning for reward, value and policy. That is, if we are at time t of the algorithm search, the loss would be defined as:

$$l_t(\theta) = \sum_{k=0}^{K} l^r(u_{t+k}, r_t^k) + l^v(z_{t+k}, v_t^k) + l^p(\pi_{t+k}, p_t^k) + l^a(u_{t+k}, z_{t+k}, \pi_{t+k})$$

where l^a is the loss term defined in part 5, measuring how close the state probability matrix is from a permutation matrix.

7. References

[1] Chaitanya K. Joshi, Thomas Laurent and Xavier Bresson. An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem. *arXiv*:1906.01227, 2019.

[2] Oriol Vinyals, Meire Fortunato, Navdeep Jaitly. Pointer Networks. *arXiv*:1506.03134v2, 2017

[3] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d?horizon. *arXiv*:1811.06128, 2018.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In ICLR 2015, *arXiv*:1409.0473, 2014.

[5] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In CVPR 2015, *arXiv*:1411.4555, 2014

[6] Pointer-Networks GitHub repository, by Jojonki. *https://github.com/jojonki/Pointer-Networks*

[7] Tobias Skovgaard Jepsen, How to do Deep Learning on Graphs with Graph Convolutional Networks. Medium article, January 2020

[8] Thomas Kipf, Graph Convolutional Networks. Blog article, September 2016

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

[10] David Applegate, Robert Bixby, Vasek Chvatal, and William Cook. Implementing the Dantzig-Fulkerson-Johnson algorithm for large traveling salesman problems. Mathematical programming, 97(1-2):91?153, 2003.

[11] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[12] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Advances in Neural Information Processing Systems, pages 3104?3112, 2014.

[13] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, David Silver
Mastering Atari, Go, Chess and Shogi by Planning with a Learned Model

*Equal contribution . Correspondence to: Meziane Chauvin <mc4568@columbia.edu>, Thomas Gaillard <tg2661@columbia.edu>, Maxime Laharrague <ml4203@columbia.edu>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.