Review of An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem

Adib Habbou
ENS Paris-Saclay x ENSIIE
adib.habbou@ens-paris-saclay.fr

Teddy Alexandre
ENS Paris-Saclay x ENSIIE
teddy.alexandre@ens-paris-saclay.fr

1 INTRODUCTION

The **Travelling Salesman Problem (TSP)** is a combinatorial optimization that represents one of the famous NP-hard problems currently. A NP-hard problem is an optimization problem that cannot be solved at large scale (large number of points), in a reasonable amount of time (it has a $O(n!)$ complexity, where n is the number of points). More precisely, the TSP aims at answering the following question: "Given a set of cities, their distances to each other, and a city of origin, find a path that goes to every city and comes back to the origin city, where the total travelling distance is minimal".

The statement of this problem is very simple, but its solution is very complex to obtain, even today. The TSP has overall many applications in different tasks, such as vehicle routing problem, supply chain, telecommunications, scheduling or planning, for example.

Over the years, many heuristics have been employed to try to find an approached solution to this NP-hard problem. But with the recent advances in the domain of artificial intelligence, especially machine learning and deep learning, different approaches based on these domains have appeared, and one of them is discussed and analyzed in this article: the **Graph Convolutional Network (Graph ConvNet)**, based on the work of [2] and [7]. The model is analyzed, experimented with different graph sizes and other parameters, and then pushed to its limitations to quantify its performance. Finally, some extensions to the model are proposed in order to improve it.

2 ALTERNATIVE METHODS

There is a wide range of TSP solvers, primarily categorized into two families: **heuristic-based** and **learning-based** approaches. The Concorde stands out as the most efficient heuristic-based TSP solver, employing cutting plane algorithms to iteratively address linear programming relaxations of the TSP [4].

But recently, a novel category of TSP solvers has emerged, leveraging innovative techniques like sequence-to-sequence learning, attention mechanisms, and geometric deep learning. One good example, is the Pointer Network, which employs a sequence-to-sequence model with attention mechanisms to generate a permutation of an input sequence [9].

An alternative method, involves representing the TSP using Graph Neural Networks (GNN) to more accurately capture its combinatorial structure compared to sequence-to-sequence models. Specifically, a structure2vec graph embedding model is trained to determine the order in which nodes are added to a partial tour [5].

Concurrent approach, substituted the structure2vec model with a Graph Attention Network. It employed an attention-based decoder trained through reinforcement learning to construct TSP solutions in an autoregressive manner [8].

In parallel with autoregressive techniques, it is possible to supervise the training of a **Graph Neural Network (GNN)** to directly generate a tour represented as an adjacency matrix. This matrix is then transformed into a viable solution using **Beam Search** [1]. The **non-autoregressive** method introduced in [6] builds on top of this work.

3 ENCODING STEP: GRAPH CONVNET

The Graph ConvNet works as a convolutional neural network adapted to 2D graphs. It is decomposed in three main layers:

Input Layer: The first layer of the Graph ConvNet is the input layer. Given a graph as an input, the layer embeds the nodes and the edges into some h-dimensional representations. The nodes $x_i \in [0,1]^2$ are linearly mapped to form node features $\alpha_i \in \mathbb{R}^h$. The edge input features β_{ij} are also computed as a linear combination of the distance weights d_{ij}. An indicator function of a TSP edge is defined to compute β_{ij}, based on a k-nearest neighbors algorithm.

Graph Convolution Layer: The second group of layers are the convolution layers. These layers, similarly to a classical convolutional neural network, perform feature extraction in the graph. To compute node and edges features, we apply successively linear combinations of the features, perform batch normalization on the results, followed by the application of an activation function Rectified Linear Unit (ReLU). The equations are detailed in the article [6].

MLP Classifier: The last class of layers of the neural network is a multi-layer perceptron classifier. The edge embedding of the last graph convolution layer is used as an input to the MLP to compute the probabilities of each edge to figure in the TSP tour at the end. The probabilities can be seen as a probabilistic heat-map over the adjacency matrix of tour connections.

4 DECODING STEP: BEAM SEARCH

At the end of our Graph ConvNet model, we obtain a **probabilistic heat map over the adjacency matrix of tour connections**. While it may be tempting to directly convert this heat map into an adjacency matrix representation of the predicted TSP tour by using an argmax function, doing so often results in tours that are invalid, containing either excessive or insufficient edges.

Indeed we can use a search strategies to efficiently make our conversion. In [6], three methods where implemented:

Greedy Search: Starts from the first node and greedily picks the next node as the neighbor with the highest probability of an edge's presence and masks odes that have been visited previously. The search concludes once all nodes have been visited.

Beam Search: Starts from the first node and explores the heat map by expanding the b most probable edge connections among the node's neighbors. We iteratively extend the top-b partial tours at each stage until all nodes are visited. The final prediction is the tour with the highest probability among the b complete tours.

Beam Search and Shortest tour heuristic: Basically, the same as Beam Search but at the end, the final prediction is the shortest tour among the set of b complete tours *(directly comparable to reinforcement learning because it involves sampling a set of solutions from the learned policy and then selecting the shortest tour).*

5 OUR EXPERIMENTS

In order to reproduce the experiments of the paper and also new ones, we used the code provided in the GitHub repository of the paper [6] author Chaitanya K. Joshi. The code was designed for an old PyTorch version (0.4.1) that is not not available nowadays.

We had to make some modification in the implementation in order to make it run with a new PyTorch version (2.0). We had to make sure that an element after a permutation is still contiguous by using `.contiguous()`. We also had to make sure that we obtain an integer when doing a specific division in the BeamSearch by replacing \ by \\. Finally, we also had to gather elements of a tensor along the second dimension using the PyTorch method `Y = Y.gather(1, X)`.

5.1 Reproduction of the paper results

Before seeking any limits and extensions of our GraphConvNet, we first sought to reproduce the results presented in the paper. Our plan was to train the graph convolutional network for different sizes of data and compare the ground-truth result to the one predicted by the graph via beam-search decoding.

We tried for five different instance sizes, similarly to the procedure presented in the article. Here, the number of nodes in the graph vary in $\{10, 20, 30, 50, 100\}$. The results are displayed below, where N denotes the number of nodes in the input graph.

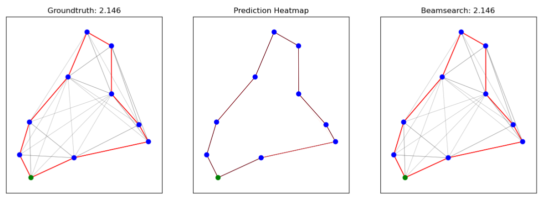

Figure 1: Reproduction for $N = 10$

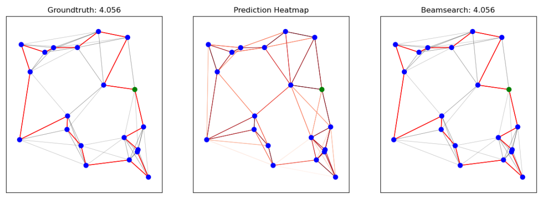

Figure 2: Reproduction for $N = 20$

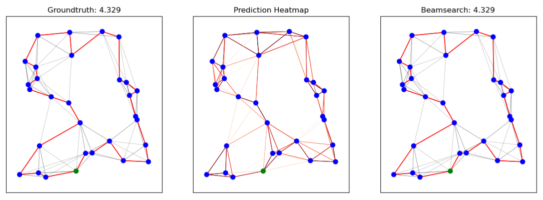

Figure 3: Reproduction for $N = 30$

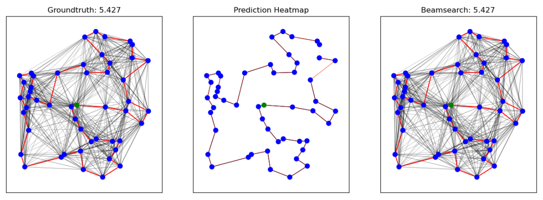

Figure 4: Reproduction for $N = 50$

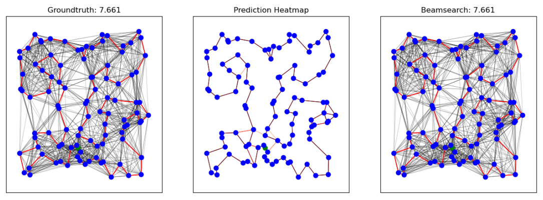

Figure 5: Reproduction for $N = 100$

The results observed are coherent with the ones presented in the Table 1 and Figures 6, 7 and 8 of the original paper [6], where the optimality gap is very close to zero. This indicates a highly efficient approach in comparison to classical operations research solvers, with Concorde serving as the reference solver.

5.2 Generalization to larger graphs

The reproduction of the results being done, one could think of trying to generalize to graphs with larger number of nodes sizes. Hence, our plan was to train the model on graphs with small number of nodes sizes (e.g $N = 20, 50$) and infer the graph convolutional network on graphs with larger number of nodes (e.g $N = 50, 100$).

Here are some results we obtained considering this approach:

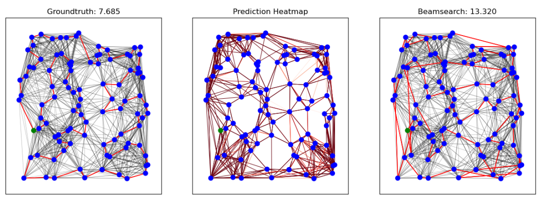

Figure 6: Training on $N = 20$, testing on $N = 100$

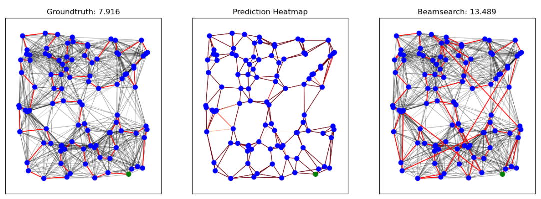

Figure 7: Training on $N = 50$, testing on $N = 100$

As we can see on the figures, the predicted distance in the traveling salesman problem is much more large than the ground-truth value. Indeed, the predicted distance is almost twice the ground-truth distance, and this is associated to a much more complex heatmap that doesn't really reflect the desired heatmap.

5.3 Generalization to smaller graphs

We can think the other way around and consider smaller graphs. This time, we train the Graph Convolutional Network on graphs with a large number of nodes (e.g $N = 100$) and test it on graphs with a smaller number of nodes (e.g $N = 20, 50$):

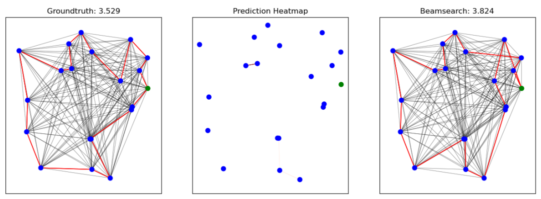

Figure 8: Training on $N = 100$, testing on $N = 20$

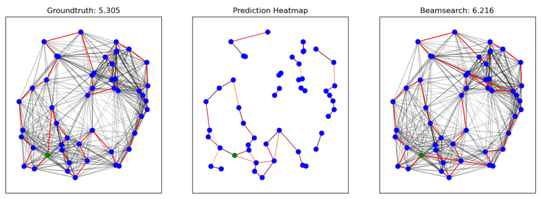

Figure 9: Training on $N = 100$, testing on $N = 50$

The model is not able to approximate the ground-truth solution closely, though the predicted distance is closer than those we could obtain by the generalization on larger graphs. This is certainly due to a very simple heatmap where the models considers that all edges have a low probability to be in the tour. Hence, the model lacks in finding the optimal tour after the decoding of this poor heatmap.

5.4 Training on fewer data

After performing dual approaches to test the generalization ability of the GNN, another approach we had is to train the model on fewer input graphs. Compared to the article where the model is trained on 10 epochs of 500 mini-batches of 20 instances [6], we train the model on 10 times less data by either using only 50 mini-batches (instead of 500) or using only 2 instances in each mini-batch.

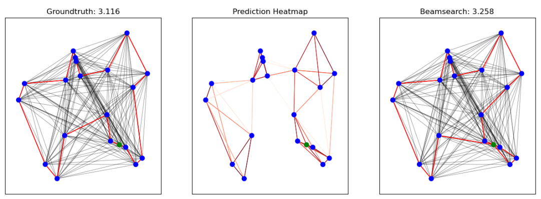

Figure 10: TSP20: batch size of 20 and 50 batches per epoch

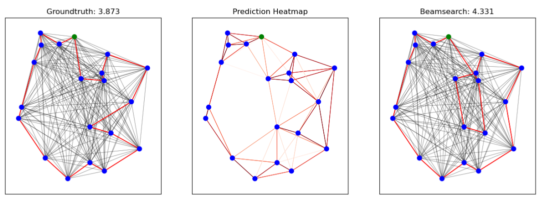

Figure 11: TSP20: batch size of 2 and 500 batches per epoch

When using a reduced number of batches per epoch, the obtained solutions tend to closely resemble the ground truth, typically deviating by a range of 150 to 350. However, when employing a smaller number of instances in each batch during training, the results tend to be suboptimal, with discrepancies from the ground truth typically falling within the range of 450 to 1000.

6 LIMITATIONS

Following the experiments made just above, we can already make a few observations about the limitations of our model :

Limitations when generalizing on larger graphs: The model has in memory graphs of smaller size. What happens when it gets a larger graph on the instance part, is that the convolutional network is kind of biased. This bias manifests as the network treating the larger graph as if it were composed of smaller instances. Hence, the GraphConvNet will generate a very complex and erratic probabilistic heat map. Following this heat map, the BeamSearch would be less likely to find an optimal tour, since there are much more combinatorial possibilities for the predicted tour at the end (cf. figures 6 and 7).

Limitations when generalizing on smaller graphs: When generalizing to smaller graphs, the model had in memory graphs of larger size. What happens here is that the model would not be able to deal with a smaller graph on the instance part, since we trained it to perform on larger graphs. Therefore, the Graph Convolutional Network assigns probabilities that are very close to zero, since each edge is not likely to figure on the optimal tour at the end. This would be the reason of why we obtain sparse-edges probabilistic heat map, resulting in a predicted optimal tour that is not exactly the optimal tour that would be predicted by operations research solvers like Concorde (cf. figures 8 and 9).

Limitations when training on fewer graph data: We wanted to observe the behaviour of the Graph Convolutional Network when we provide few input graphs, to see if the returned solution was approaching the ground-truth solution. It appears that the obtained solutions are not quite far from the ground-truth solution, and some of them being actually quite close from the ground-truth. This is evidenced by the experiment made in figure 10, with a batch size of 20. However, when reducing the batch size to 2 and keeping the number of batches per epoch to 500, the model has poorer performance, with a predicted tour that recedes from the ground-truth solution (cf. figure 11). This proves that the model is really efficient only when trained on large number of data.

7 EXTENSIONS

The authors of the original paper [6] published a new article on May 2022 where they are bringing together several architectures and learning paradigms for learning-driven TSP solvers [3]. For instance, we can define four of them:

Leveraging equivariance and symmetries: The autoregressive Attention Model constructs TSP tours sequentially, but does not consider the underlying symmetries of routing problems. Many approaches have been employed, such as reinforcement learning to train the attention model, or finding invariances in the graph (rotation, reflection, translation) to simplify the problem afterwards.

Improved graph search algorithms: Many papers have focused on an improved search algorithm with powerful algorithms, based for example on Dynamic Programming or Monte-Carlo Tree Search. Such algorithms allow to solve the algorithms for various scales of graph instance sizes. The problem is solved on smaller graphs, then the solutions are merged before applying the Monte-Carlo Tree Search procedure.

Learning with local search heuristics: Recent studies have explored an alternative approach to constructive Auto-Regressive (AR) and Non Auto-Regressive (NAR) decoding schemes. This alternative involves developing the capability to iteratively improve sub-optimal solutions or learning to perform local search.

Learning paradigms that promote generalization: Future work could investigate innovative learning paradigms explicitly designed to emphasize generalization beyond supervised and reinforcement learning. The objective would be to train model parameters with a specific emphasis on fast-adaptation and fine-tuning to new data distributions and varying problem sizes. Mutli-task pre-training for routing could be a good approach to acquire neural network representations that are broadly applicable and can effectively transfer to new and unfamiliar routing problems.

8 CONCLUSION

In conclusion, this report provided an overview of an efficient Graph Convolutional Network (Graph ConvNet) technique for solving the Travelling Salesman Problem (TSP). The TSP is a complex optimization problem with various real-world applications, and the Graph ConvNet offers a machine learning-based solution.

The report introduced the TSP and highlighted the challenges it poses, setting the stage for the discussion of learning-based approaches like the Graph ConvNet. The architecture of the Graph ConvNet was explained, emphasizing its application to 2D graphs. Alternative methods for TSP, such as sequence-to-sequence learning and attention mechanisms, were briefly mentioned.

Experiments were conducted to replicate the results from the original paper and explore the model's generalization capabilities. The Graph ConvNet showed limitations in handling graphs of different sizes and struggled when trained on a reduced amount of data.

While the Graph ConvNet presents a promising approach, its limitations in generalization and sensitivity to training data volume indicate the need for further research. The report suggested potential extensions, including leveraging symmetries, improving search algorithms, and exploring paradigms for better generalization.

The Graph ConvNet offers a novel perspective on solving the TSP through machine learning, but ongoing research is crucial for refining its performance and addressing its limitations. Future studies should focus on enhancing generalization, exploring new architectures, and improving the robustness of learning-driven TSP solvers.

REFERENCES

[1] Afonso S Bandeira Alex Nowak, Soledad Villar and Joan Bruna. 2017. A note on learning algorithms for quadratic assignment with graph neural networks. arXiv:1706.07450

[2] Xavier Bresson and Thomas Laurent. 2018. Residual Gated Graph ConvNets. arXiv:1711.07553 [cs.LG]

[3] Louis-Martin Rousseau Thomas Laurent Chaitanya K. Joshi, Quentin Cappart. 2022. Learning the Travelling Salesperson Problem Requires Rethinking Generalization. (2022). arXiv:2006.07054v6 [cs.LG]

[4] Vasek Chvatal David L Applegate, Robert E Bixby and William J Cook. 2006. The traveling salesman problem: a computational study.

[5] Yuyu Zhang Bistra Dilkina Hanjun Dai, Elias Khalil and Le Song. 2017. Learning combinatorial optimization algorithms over graphs. , 6348-6358 pages.

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

[7] M.F. Medress, F.S. Cooper, J.W. Forgie, C.C. Green, D.H. Klatt, M.H. O'Malley, E.P. Neuburg, A. Newell, D.R. Reddy, B. Ritea, J.E. Shoup-Hummel, D.E. Walker, and W.A. Woods. 1977. Speech understanding systems: Report of a steering committee. *Artificial Intelligence* 9, 3 (1977), 307–316.

[8] Alexandre Lacoste Yossiri Adulyasak Michel Deudon, Pierre Cournut and Louis-Martin Rousseau. 2018. Learning heuristics for the tsp by policy gradient. , 170–181 pages.

[9] Meire Fortunato Oriol Vinyals and Navdeep Jaitly. 2015. Pointer networks. , 2692-2700 pages.