KDD-BR 2021: Using Graph Neural Networks for Link Prediction in TSP problem

Bruno Klaus de Aquino Afonso[1], **Willian Dihanster Gomes de Olivera** [1]
Jéssica Domingues Lamosa[1], **Lilian Berton**[1]

[1]Institute of Science and Technology – Federal University of Sao Paulo (UNIFESP)
São José dos Campos 12247-014 – SP – Brazil

{bruno.klaus, dihanster.willian, jd.lamosa, lberton}@unifesp.br

Abstract. *In the traveling salesman problem (TSP), the objective is to find a route that goes through all the cities and returns to the city of origin with the shortest total distance covered. As an NP-complete problem, finding solutions is not easy, and several heuristics have been proposed. Currently, with the advance in the use of Machine Learning (ML), it is possible to use ML to predict connections between cities in the optimal solution. Our model, based on graph neural networks, obtained an F1-score of* `0.10645` *in the public leaderboard of KDD-BR 2021, enough to outperform the greedy baseline. This shows ML can be a good technique to generate solutions faster or even build better solutions.*

1. Introduction

The traveling salesman problem (TSP) was proposed by [Menger 1932]. Given a list of cities, the task is to find a route that goes through all cities exactly once, returning to the origin with minimal total distance covered. The problem is classified as a combinatorial optimization problem, and known to be NP-complete. Therefore, it is not easy to obtain optimal solutions. Brute force can solve the problem by testing all solutions, but the computational time is extremely high and unfeasible even for medium data sets.

Several heuristics have been proposed to solve the problem [Cormen et al. 2009]. One idea is the greedy approach, where the closest city is always chosen. Another approach is the evolutionary algorithms that always seek to improve the solutions found, allowing for greater flexibility due to the several different algorithms and techniques. However, these approaches do not guarantee finding the optimal solution, although they can be useful for many applications. Currently, Machine Learning (ML) techniques are being applied in several applications with good results. The TSP problem can be modeled using ML, in which cities are examples with their respective coordinates and the output (which we will predict) is a binary matrix encoding which links comprise the solution.

In 2017, the Brazilian competition on Knowledge Discovery in Databases (KDD-BR) started. Since then, competitors have been asked to predict: images of meteors; production of palm oil harvests; the similarity between partitions produced by a manual clustering of molecular markers and those obtained by an auto-clustering tool; the unavailability of cars in a car rental agency. This year, the goal of KDD-BR 2021[1] was to develop methods that predict edges belonging to solutions in TSP.

[1]http://c4ai.inova.usp.br/bracis/kdd.htm

In this paper, we work on the requested solution. The input data contained 100k training examples of TSP problem-solution pairs ranging from 50 to 200 city nodes and was provided by the Loggi company. Our results indicate the proposed approach using graph neural networks is sound, outperforming the greedy baseline and attaining 1st place in the (temporary) public leaderboard.

The paper is organized as follows. Section 2 goes over related work, Section 3 presents the material and methods used in the research. Results and discussions are detailed in Section 4. Finally, Section 5 draws some concluding remarks.

2. Related work

From the point of view of the graph and combinatorial problems, Graph neural networks (GNNs) can handle a trainable dataset in different configurations for the relational structure of each problem instance and make a promising technique for combinatorial domains [Prates et al. 2019].

[Joshi et al. 2019] unified state-of-the-art architectures and learning paradigms into one experimental pipeline using an efficient graph neural network for the TSP that promote generalization to instances larger than your training dataset.

Thinking about the increased use of GNNs [Dwivedi et al. 2020] introduced a reproducible GNN benchmark so that other researchers could add new models to rigorously evaluate the performance of GNNs on medium-scale datasets.

In a recent survey, [Joshi et al. 2020] proposed a network architecture that computes h-dimensional representations for each node and edge in the graph with an approach to minimize cross-entropy loss through gradient descent. Then, they compare the results with other deep learning techniques and get better results in different aspects such as solution quality, highly parallelized implementation, and learning optimal solutions.

3. Material and methods

3.1. Choosing a model

The objective of this challenge was to detect whether a given edge is part of the TSP solution. The evaluation did not require us to produce a valid tour, which influenced our decision to build our model as a simple edge classifier.

One of the most important choices we have to make is the choice of graph convolution, which enables us to perform differentiable message passing through the graph. We opted for the **EdgeConv** (edge convolution) operator [Wang et al. 2019]. If a given node i has its features denoted by $\boldsymbol{x}_i$ and neighborhood by $\mathcal{N}(i)$, we calculate the updated representation as

$$\forall i : \mathbf{x}'_i = \sum_{j \in \mathcal{N}(i)} h_{\mathbf{\Theta}}(\mathbf{x}_i \,\|\, \mathbf{x}_j - \mathbf{x}_i) \tag{1}$$

In this equation, $h_{\mathbf{\Theta}}$ denotes a differentiable model with parameters Θ, and $||$ the concatenation operator. Translation invariance is a characteristic of TSP, which encouraged us to use a convolution that incorporates node differences.

Another important consideration for the problem is the loss function. According to the challenge rules, solutions are rated according to the *F1-score*, which is not a differentiable loss. If the targets were balanced, using the mean squared error would suffice.

However, the TSP binary target matrix is sparse, which means that this naive approach would heavily bias predictions towards the "0" class. An appropriate weighting scheme is proposed in [Joshi et al. 2019], which we also adopt. Any nodes that are part of the TSP solution will have a weight w_1, whereas the rest are assigned weight w_0. These weights are given as

$$w_1 = \frac{n^2}{c \times 2n} \quad \text{and} \quad w_0 = \frac{n^2}{c(n^2 - 2n)}, \tag{2}$$

where $c = 2$ is the number of classes, and n is the number of nodes. Using the inverse of the proportions of the classes, we mitigate the bias towards the majority class.

3.2. Implementation details

One of the requirements for this competition was that the code should be reproducible. To achieve this, we opted to use the Kaggle Notebooks environment. A notebook is divided into *code cells* and *markdown cells*, which we use to add useful annotations that explain the idea behind our code. This environment enables us to share our notebooks directly through the Kaggle platform and saves the final output obtained by the code[2]. In our case, this output is simply the predictions on the test set.

Our code is written in Python 3.7, and accesses the NVIDIA TESLA P100 GPU provided by Kaggle. Most of the code related to inference relies on the Pytorch and Pytorch Geometric (PyG), providing us with easy access to graph convolutions and neural network layers.

A separate notebook was created to convert the CSV files to a Pytorch file (`.pt` extension). This file is simply a list of `Data` objects, which is an interface provided by PyG. Each of those `Data` objects represents a graph, with the attributes:

- `x`: the 2D coordinates of all nodes
- `m`: the size of this TSP problem
- `y_index`: the indices of nodes belonging to the target TSP solution

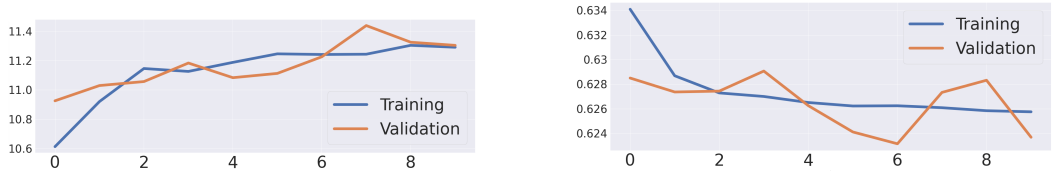
The original CSV files contained target predictions in a dense format, encoded by a dense square matrix of size 200. Storing the nonzero indices in `y_index` results in a significantly reduced memory footprint. Targets are selectively converted from sparse to dense whenever we are calculating the loss of a mini-batch. We similarly calculate the KNN graph with $k = 20$ neighbors on-demand, storing the result in the attribute `edge_index`.

We stacked a total of 5 EdgeConv layers, followed by a multilayer perceptron (MLP) comprised of 2 hidden layers, with respective sizes 512 and 256. The differentiable model $h_{\mathbf{\Theta}}$ of each convolutional layer was chosen to be an MLP with a single hidden layer of size 128. Adam was used to optimize the model, with 0.001 as the learning rate.

4. Results

Figure 1 shows the progress of the F1-score and loss function metric over each epoch. After nine epochs, the model maxed out, with the validation set having an F1-score of `11.30` (or `0.1130`). Subsequent iterations did not yield any improvement. When submitted to the leaderboard, an F1-score of `0.10645` was attained. This was good enough for 1st place in this temporary leaderboard; The 2nd place achieved `0.07906`, and the greedy baseline given by organizers had a score of `0.07667`.

[2]https://www.kaggle.com/isonettv/kdd-br-parana-solution



(a) F1 Score **(b)** Loss function

Figure 1. Metrics on the training and validation sets on each epoch.

5. Conclusion

The KDD-BR 2021 Challenge asked competitors to develop methods capable of performing link prediction on TSP, detecting edges that are likely to be part of the solution. We accomplished this by using a Graph Neural Network with stacked edge convolution operators and a loss function addressing the unbalanced classes. Despite of little hyperparameter and model tuning, we surpassed the challenge baseline and attained first place in the provisional leaderboard.

6. Acknowledgements

The authors thank CAPES Grant Numbers 23038.014333/2020-46 and 88887.508531/2020-00.

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to algorithms*. MIT press.

Dwivedi, V. P., Joshi, C. K., Laurent, T., Bengio, Y., and Bresson, X. (2020). Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*.

Joshi, C. K., Cappart, Q., Rousseau, L.-M., Laurent, T., and Bresson, X. (2020). Learning tsp requires rethinking generalization. *arXiv preprint arXiv:2006.07054*.

Joshi, C. K., Laurent, T., and Bresson, X. (2019). An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*.

Menger, K. (1932). Das botenproblem. *Ergebnisse eines mathematischen kolloquiums*, 2(4):11–12.

Prates, M., Avelar, P., Lemos, H., Lamb, L., and Vardi, M. (2019). Learning to solve np-complete problems: A graph neural network for decision tsp. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:4731–4738.

Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., and Solomon, J. M. (2019). Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12.