APMTH 220 Geometric Methods for Machine Learning Spring 2025

E(2)-Equivariant Graph Neural Network Improves Data Efficiency of Traveling Salesman Problem

Name: Olav Førland *Submission Date: 05/09/25*

Abstract

We introduce an E(2)-equivariant graph neural network encoder for the 2D Euclidean Traveling Salesman Problem (TSP). By adapting Nequip's [2] E(3)-equivariant architecture to the 2D setting of TSP, our model efficiently encodes translation, rotation, and reflection symmetries in the input graph. We compare our encoder in an end-to-end supervised link-prediction pipeline against a state-of-the-art Graph ConvNet encoder [5], where we due to computational constraints only train on 4 % of the standard dataset (51 200 samples) for 20- and 50-city instances. Our model reduces the optimality gap by about 25 % and 10 %, respectively. Moreover, t-SNE visualizations of the learned graph embeddings reveal that our encoder clusters geometrically equivalent tours in the same latent region, demonstrating explicit enforcement of E(2)-equivariance. These results highlight our models ability to improve data efficiency when learning the TSP.

1 Introduction

Combinatorial optimization problems are NP-hard constrained integer optimization problems that seek an optimal object from a finite set of discrete elements. Due to their NP-hard nature they become intractable to solve optimally for large instances [6]. The Traveling Salesman Problem (TSP) is perhaps the most well-studied of these [8]; given a set of cities and an origin, the salesman aims at finding the shortest path that visits all cities while returning to the origin. This problem arises frequently in real-world logistics, where exact solvers rely on hand-crafted heuristics to scale to realistic instances, with degrading quality guarantees as problem size increases [13]. The Concord TSP solver [1] is able to optimally solve problems with thousands of nodes, but entails large execution times. This makes it infeasible for large-scale, real-time logistics. To strike a balance between solution quality and computational efficiency, machine learning is a natural choice.

Recently, Graph Neural Networks (GNNs) has seen great success on a number of machine learning tasks [12][17][9][7][2]. This has naturally sparked interest in applying them to the TSP, which has a natural graph-formulation. [6] propose a GNN-based model that learns edge-selection heuristics competitive with the Concorde solver on benchmark instances. They formulate the problem as a supervised link-prediction tasks, encodes the graph using a GNN, and predicts whether each edge belongs to the optimal solution using a final Multi -Layer-Perceptron (MLP). During inference, the solution is decoded using Beam-Search [10]. Continuing this work, [5] augment the pipeline with a final attention layer that auto-regressively builds the solution, which improves generelizability. [14] introduce a deep Reinforcement Learning (RL) framework that enforces equivariance through explicit preprocessing of training samples and integrates local search to further refine solutions. The model is trained directly on the length of the predicted tour via policy gradient. Recently, [11] use a novel Unsupervised Learning (UL) approach that trains a GNN on a surrogate loss that incorporates both the length of the tour and the constraint that the tour should follow a Hamiltonian cycle.

We adapt the pipeline of [5] and aim to improve the graph encoding through a novel E(2)-equivariant GNN. This GNN is a two-dimensional version of the E(3)-equivariant Nequip-architecture that works

very well for computing interatomic potentials [2]. The intuition behind such a seemingly leap of fate from quantum chemistry to operations research is that the graph structure that emerge in both problems are very similar, and should therefore adhere to the same symmetries. This will be motivated in Section 3.

2 Background & Notation

Problem Definition Given a set of cities and a distance between each pair of cities, the TSP aims at finding the shortest possible route that visits each city and returns to the origin city. Formally, given N nodes $V = \{1, 2, \ldots, N\}$ and edges given by a distance metric $d : V \times V \longrightarrow \mathbb{R}_{\geq 0}$, we aim to find a permutation of the nodes $\pi : \{1, 2, \ldots, N\} \longrightarrow V$ that minimizes the total tour length $L(\pi)$:

$$\pi^* \;=\; \arg\min_{\pi} \; L(\pi) \;=\; \arg\min_{\pi} \; \sum_{m=1}^{N-1} d\big(\pi(m),\, \pi(m+1)\big) \;+\; d\big(\pi(N),\, \pi(1)\big)$$

We restrict ourselves to the case where each city i is represented by a point $(x_i, y_i) \in \mathbb{R}^2$, and the metric we use is the Euclidean distance: $d(i, j) \;=\; \sqrt{(x_i - x_j)^2 \;+\; (y_i - y_j)^2}$, i.e., the 2D TSP.

Graph Neural Networks A GNN aims to compute a d-dimensional embedding of each node $i \in V$ and, optionally, each edge $(i, j) \in E$. It does this by iteratively exchanging messages between neighbors over L layers, thus building representations of the L-hop neighborhood of each node. The embedding of node i and edge (i, j) at layer l is denoted $\mathbf{h}_i^{(k)}$ and $\mathbf{e}_{ij}^{(l)}$, respectively, where $\mathbf{h}_i^{(0)}$ and $\mathbf{e}_{ij}^{(0)}$ are functions of the input. At each layer $l = 1, \ldots, L$, every node i aggregates information from its neighbors $\mathcal{N}(i)$ and updates its embedding via two functions — a message function M and an update function U. Note that $\mathbf{e}_{ij}^{(l)}$ may or may not be updated through a function E:

$$\begin{aligned}
\mathbf{m}_i^{(l+1)} &= \mathrm{AGG}\Big(\big\{M(\mathbf{h}_i^{(l)},\, \mathbf{h}_j^{(l)},\, \mathbf{e}_{ij}^{(l)}) \mid j \in \mathcal{N}(i)\big\}\Big), \\
\mathbf{h}_i^{(l+1)} &= U(\mathbf{h}_i^{(l)},\, \mathbf{m}_i^{(l)}) \\
\mathbf{e}_{ij}^{(l+1)} &= E(\mathbf{e}_{ij}^{(l)}, \mathbf{h}_i^{(l+1)},\, \mathbf{h}_j^{(l+1)}).
\end{aligned}$$

Here AGG is a permutation-invariant aggregator, usually sum, mean, or max. After L layers, the final node embeddings are given by $\{\mathbf{h}_i^{(L)}\}_{i=1}^{N}$.

Graph Attention Mechanism Attention is a mechanism that allows each element in a set to selectively focus on other elements when computing its representation. Given a set of queries $\mathbf{Q} \in \mathbb{R}^{N \times d}$, keys $\mathbf{K} \in \mathbb{R}^{M \times d}$, and values $\mathbf{V} \in \mathbb{R}^{M \times d_v}$, the scaled dot-product attention is defined as

$$\mathrm{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \mathrm{softmax}\Big(\tfrac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d}}\Big)\,\mathbf{V},$$

where the softmax is applied row-wise and $\sqrt{d}$ is a scaling factor to mitigate large inner products.

In graph settings (Graph Attention Networks [17]), each node i computes attention weights over its neighbors $\mathcal{N}(i)$ by first projecting its embedding $\mathbf{h}_i$ and each neighbor embedding $\mathbf{h}_j$ via a learnable linear map $\mathbf{W}$, and then using a shared attention vector $\mathbf{a}$:

$$\alpha_{ij} = \frac{\exp\big(\mathrm{LeakyReLU}(\mathbf{a}^\top[\mathbf{W}\mathbf{h}_i \,\|\, \mathbf{W}\mathbf{h}_j])\big)}{\sum_{k \in \mathcal{N}(i)} \exp\big(\mathrm{LeakyReLU}(\mathbf{a}^\top[\mathbf{W}\mathbf{h}_i \,\|\, \mathbf{W}\mathbf{h}_k])\big)}.$$

The updated node embedding is then

$$\mathbf{h}_i' \;=\; \sigma\Big(\sum_{j\in\mathcal{N}(i)} \alpha_{ij}\,\mathbf{W}\,\mathbf{h}_j\Big),$$

which is often extended to multi-head attention by concatenating or averaging several such mechanisms.

Equivariance and Invariance Equivariance and invariance are concepts that naturally arises in physical systems, where physical properties have well-defined transformations under rotation, reflection, and translation. In 2D a function is said to be E(2)-equivariant if it is equivariant under these transformations [3]. Formally, a function $f : \mathcal{X} \to \mathcal{Y}$ is said to be *equivariant* with respect to a group G if for every $g \in G$ and every $x \in \mathcal{X}$,

$$f\big(g \cdot x\big) \;=\; g \cdot f(x).$$

In the 2D Euclidean Traveling Salesman setting we care about the group $E(2)$ of translations, rotations and reflections. Concretely, for any rotation $R(\alpha)$, translation t, and reflection M, the following must hold:

$$f\big(R(\alpha)x\big) = R(\alpha)f(x), \quad f\big(x+t\big) = f(x)+t, \quad f\big(Mx\big) = M\,f(x).$$

NequIP: E(3) Equivariant GNNs for Interatomic Potentials NequIP [2] develops E(3)-equivariant convolutions over geometric tensor features of atoms. Each atom a carries feature vectors of discrete rotation orders $l = 0, 1, 2, \ldots$ with parity $p \in \{-1, 1\}$ denoted $V_{acm}^{(l,p)} \in \mathbb{R}^{2l+1}$ where $m \in [-l, l]$ and c is the feature index. These are irreducible representations of the O(3) group of dimension $2l + 1$ encoding angular dependence up to degree l.

When updating the GNN, neighbours are combined using equivariant convolutional filters. The convolutional filters are restricted to be product of learnable radial functions and spherical harmonics, which are equivariant under SO(3).

$$S_m^{(l)}(\vec{r}_{ij}) = R(r_{ij})Y_m^{(l)}(\hat{r}_{ij})$$

where $\vec{r}_{ij}$ is the relative position from node i to node j, decomposed into direction $\hat{r}_{ij}$ and distance r_{ij}. The learnable radial function $R(r_{ij})$ is implemented via an MLP on a Bessel basis, normalized by some cutoff radius r_c:

$$R(r_{ij}) = \mathrm{MLP}[B(r_{ij})], \text{ where } B(r_{ij}) = \frac{2}{r_\mathrm{c}} \frac{\sin(\frac{b\pi}{r_\mathrm{c}} r_{ij})}{r_{ij}} f_{env}(r_{ij}, r_c) \in \mathbb{R}^{N_b}$$

where $f_{env}(r_{ij}, r_c)$ is an envelope function ensuring a smooth transition to zero.

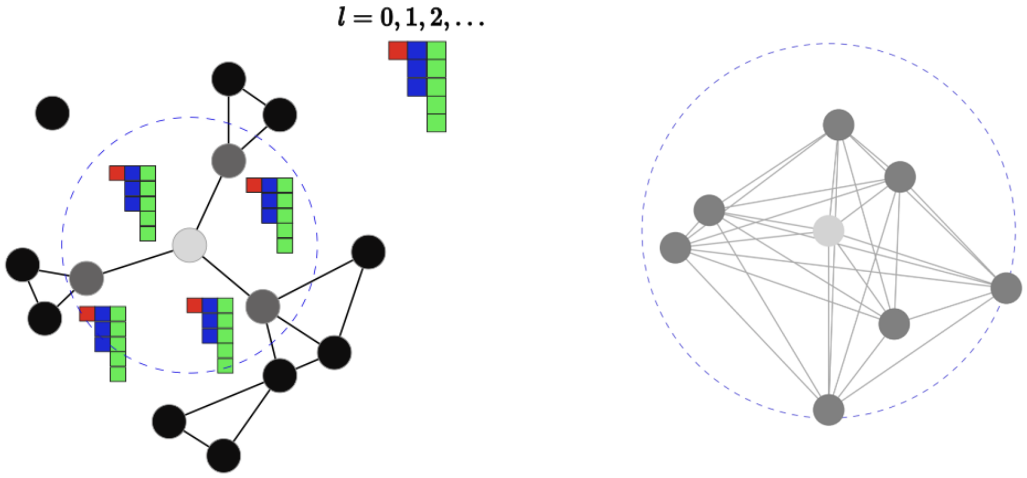
Finally, the node features and the filter are combined in an equivariant manner through a contraction with the Clebsch-Gordan coefficients.

$$\mathcal{L}_{acm_o}^{l_o,p_o,l_f,p_f,l_i,p_i}(\vec{r}_a, V_{acm_i}^{l_i,p_i}) = \sum_{m_f,m_i} C_{l_i,m_i,l_f,m_f}^{l_o,m_o} \sum_{b\in S} R(r_{ab})_{c,l_o,p_o,l_f,p_f,l_i,p_i}\, Y_{m_f}^{l_f}(\hat{r}_{ab})\, V_{bcm_i}^{l_i,p_i}.$$

After convolution, features from different (l, p) channels are mixed via atom-wise linear layers and gated nonlinearities. A residual skip connection adds the block's input features to its output. NequIP stacks K such blocks, which they call *interaction blocks*. The $l = 0$ feature vector of the final block is passed to downstream tasks.

3 Proposed Approach

We adapt the architecture of NequIP for the traveling salesman problem. The intuition of using an architecture devised for interatomic potentials on a classic operations research problem such as the 2D Euclidean TSP, is that the graph structures of the two problems are very similar. In both cases, the important aspect of the graph is the relative distance between each pair of nodes. The rotation, absolute position, or parity of the graph is completely irrelevant, as well as the permutation order of the nodes. Therefore, transforming the 3D architecture of NequIP into 2D should yield E(2)-equivariant graph embeddings for the TSP.



(a) NequIP graph model of a molecule[2]. Each atom is connected to all within a cutoff radius r_c

(b) TSP graph model. Can be seen as a special case of NequIP with cutoff radius $r_c = r_{max}$

Figure 1: Comparison of NequIP and TSP graph structure.

Further, we adapt the pipeline of [5] and formulate the TSP as a supervised link prediction task, where given a set of training examples and ground truth shortest paths, we train a model to output a probability distribution over the outgoing and incoming edges from each node. The model consists of a GNN that encodes each node to a latent space, and an attention based decoder that, given a partial tour and the graph encoding, predicts the probability for each node not visited yet to be the next in the tour. The contributions of this paper lies in improving the graph encoding by making it E(2)-Equivariant.

Equivariant GNN Encoder We adapt the E(3)-equivariant convolutional filter of NequIP to 2D. The learnable basis function $R(r_{ij})$ remains a trainable Multi-Layer-Perceptron (MLP) on a basis embedding consisting of Bessel radial functions of the distance, but we drop the envelop function and instead set the cutoff radius to the largest distance between any two nodes:

$$R(r_{ij}) = \text{MLP}[B(r_{ij})], \text{ where } B(r_{ij}) = \frac{2}{r_{\max}} \frac{\sin(\frac{b\pi}{r_{\max}} r_{ij})}{r_{ij}} \in \mathbb{R}^{N_b}$$

Furthermore, spherical harmonics is replaced with its 2D counterpart, often called circular harmonics, where the irreducible components is just the Fourier basis, and equivariant under O(2) [4]:

$$Y_m^{(l)}(\hat{r}_{ij}) = \{1,\ \cos(\theta_{ij}),\ \sin(\theta_{ij}),\ \ldots,\ \cos(l \cdot \theta_{ij}\},\ \sin(l \cdot \theta_{ij})\}$$

As in NequIP, the node features and the filter is combined in an equivariant manner through a contraction with the Clebsh-Gordan coefficients. This is also simplified in the 2D setting, where the Clebsh-Gordan coefficients reduces $C^{\ell_o,m_o}_{\ell_i,m_i,\ell_f,m_f} = \mathbb{1}_{\{m_i+m_f=m_o\}}$ [3]. Our convolutional layer then becomes:

$$\mathcal{L}^{\ell_o,\ell_f,\ell_i}_{ncm_o}(\vec{r}_v, h^{\ell_i,m_i}_{vc}) = \sum_{m_i=-\ell_i}^{\ell_i} \sum_{m_f=-\ell_f}^{\ell_f} \mathbb{1}_{\{m_i+m_f=m_o\}} \sum_{u\in\mathcal{N}_v} R_{\ell_i,\ell_f,\ell_o}(r_{uv}) Y^{\ell_f}_{m_f}(\hat{r}_{uv}) h^{\ell_i,m_i}_{uc}$$

where h^{ℓ_i,m_i}_{uc} is the representation of node u. For the scope of this project, we restrict ourselves to having scalar input features, i.e., $\ell_i = 0$.

Attention Decoding We follow the decoding approach of [5] by constructing the solution tour by sequentially predicting one next-city at a time, treating each selection as a link-prediction problem with an attention-based decoder. The final node embeddings produced by our GNN, $\{h_i^L \in^d\}_{i=1}^N$, are used to form a context at decoding step t, given the partial tour $\pi_{<t} = (\pi_1, \ldots, \pi_{t-1})$:

$$\hat{h}_t^C = W^C\big[h^G;\, h^L_{\pi_{t-1}};\, h^L_{\pi_1}\big], \quad h_t^C = \mathrm{MHA}\big(Q = \hat{h}_t^C,\ K = \{h_j^L\}_{j=1}^N,\ V = \{h_j^L\}_{j=1}^N\big),$$

where $W^C \in \mathbb{R}^{3d\times d}$ and MHA denotes multi-head self-attention. We then compute unnormalized logits

$$\hat{p}_{t,j} \ = \ \begin{cases} \dfrac{(W^Q h_t^C)^\top (W^K h_j^L)}{\sqrt{d}} & j \notin \pi_{<t}, \\ -\infty & \text{otherwise}, \end{cases}$$

with $W^Q, W^K \in \mathbb{R}^{d\times d}$.

The architecture is trained by minimizing the cross-entropy loss over the next-city, and backpropagating the gradient through both decoder and encoder to update all parameters.

Experimental Setup We compare our GNN encoder with that of [5] for problem instances of 20 and 50 cities, respectively. We compare against their best model, using a autoregressive decoding scheme, 3 GNN encoder layers, and a attention decoder head. The GNN layers are Graph ConvNets with MAX neighborhood aggregation function and BatchNorm, and the attention head has a hidden dimension of $d = 128$. Note that we only replace the Graph encoder, and use an embedding dimensionality of 64 in both cases.

Models are implemented in Pytorch [15], and trained using the Adam optimizer for 10 epochs with batch size of 16 and a fixed learning rate 10^{-4}. The training data is generated by uniformly drawing locations within the unit square. Ground truth labels are obtained by the Concorde solver [1]. Due to lack of computational resources, we train all models from scratch on only 4% of the original data (51,200 samples) on a Apple M1 Max CPU. We adapt the code of [5] to ensure a fair comparison with their model when training from scratch.

We compare the models on a held-out test set of 1,280 TSP instances with the number of cities the model was trained on. We use the *optimality gap* as evaluation metric, defined as

$$\Delta_{\text{opt}}(\hat{\pi}) = \frac{L(\hat{\pi}) - L(\pi^*)}{L(\pi^*)}$$

Code and reproducible experiments are available here.

4 Results

Figure 2 compares the optimality gap using our encoder and the encoder from [5] on instances of 20 and 50 cities on a held-out validation set. The plot shows that our approach reduces the optimality gap by about 25% (from about 4% to 3%) in the smaller instance and by about 10% (from about 11% to 10%) on the larger instance compared to that of [5]. Note that [5] is able to close the optimality gap when training on the full data with an embedding dimension of 128. However, despite training on a significant smaller dataset, these results indicate that making the encoder E(2)-equivariant has performance benefits.

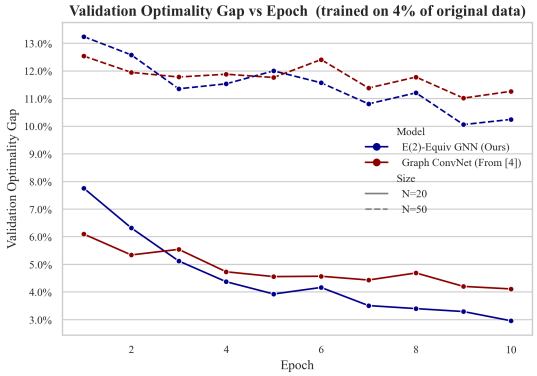


Figure 2: Optimality gap of our GNN model vs. that of [5] for problem instances with 20 and 50 cities.

Figure 3 shows the graph embedding for each of the approaches on the validation set using t-SNE [16]. The difference between the embeddings is clear. Our mangled representation shows that the GNN preserves rotations and reflections within the same space as the original example, in accordance with the definition of equivariance. ConvNet, on the other hand, separate these transformations into distinct regions of its latent space, effectively embedding three symmetrical TSP graphs into vastly different latent representations.

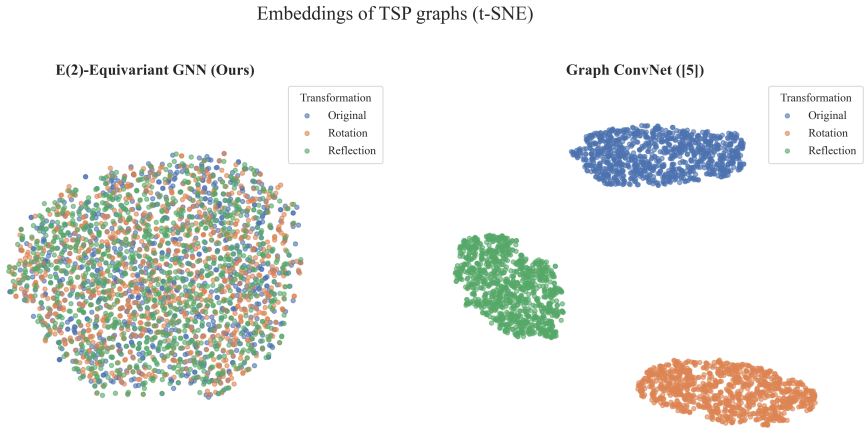


Figure 3: t-SNE projection of the graph-level embeddings for our E(2)-Equivariant GNN and the Graph ConvNet from [5] on the held-out validation set.

5 Discussion & Conclusion

Our experiments indicate that adapting the E(3)-equivariant architecture of NequIP for the Traveling Salesman Problem encodes meaningful inductive biases for learning the efficiently learning it. In particular, including our adapted GNN architecture reduces the optimality gap significantly quicker than the state-of-the-art implementation of a Graph ConvNet encoder from [5]. The t-SNE visualizations further imply that our GNN embeds geometrically equivalent TSP instances into the same neighborhood, whereas the baseline ConvNet seperate these into distinct regions. These results strongly suggest that the architecture proposed enforces E(2)-equivariance that can help improve sample efficiency for TSP.

Our current implementation relies solely on scalar ($\ell = 0$) node features. As mentioned earlier, Nequip showed that incorporating higher-order tensor features can capture richer geometric information and further boost performance. Moreover, our training regimen is limited to relatively small instances (up to 50 cities) due to computational constraints. In practice, these are trivial to solve for solvers such as Concord. Therefore, these results are only meaningful if we are able to scale to larger graphs both for training and inference. Finally, by operating in E(2), our model assumes a planar Euclidean geometry, which may not hold for real-world routing problems on the Earth's surface; therefore, for such applications, we might need to go back to the original E(3)-equivariant formulation.

Future Directions Based on these limitations, there are several directions for future work. The architecture should be scaled up to realistic instances where modern solvers fall short of finding good solutions within reasonable time. This entails accessing specialized hardware such as GPUs for training and inference, and adapting the architecture to be able to generalize from smaller graphs seen in training to larger graphs during inference. There are several promising ways to do this. NequIP has already shown great success with extending the node features to include higher order tensors ($\ell > 0$), and [5] shows promising results by making the graphs sparser K-nearest neighbors.

6 Acknowledgements

The code for this project has been based on the repository of [5]. Their pipeline has been invaluable for training models with our proposed architectures, and making fair comparisons to their work on little data. The original repository is found here

References

[1] David Applegate, Robert Bixby, Václav Chvátal, and William Cook. Concorde tsp solver. `https://www.math.uwaterloo.ca/tsp/index.html`, 2006. (Cited on page 1).

[2] Simon Batzner, Albert Musaelian, Lixin Sun, Mario Geiger, Jonathan P Mailoa, Mordechai Kornbluth, Nicola Molinari, Tess E Smidt, and Boris Kozinsky. E (3)-equivariant graph neural networks for data-efficient and accurate interatomic potentials. *Nature communications*, 13(1):2453, 2022.

[3] Matthew Foster. Rotations in 3d, so(3), and su(2). Lecture Notes LA_1_v2.1, Rice University, January 2021. Version 2.1.

[4] Jacob Helwig, Xuan Zhang, Cong Fu, Jerry Kurtin, Stephan Wojtowytsch, and Shuiwang Ji. Group equivariant fourier neural operators for partial differential equations. *arXiv preprint arXiv:2306.05697*, 2023.

[5] Chaitanya K Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning the travelling salesperson problem requires rethinking generalization. *Constraints*, 27(1):70–98, 2022.

[6] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the travelling salesman problem. *arXiv preprint arXiv:1906.01227*, 2019.

[7] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *nature*, 596(7873):583–589, 2021.

[8] Jan Karel Lenstra and AHG Rinnooy Kan. Some simple applications of the travelling salesman problem. *Journal of the Operational Research Society*, 26(4):717–733, 1975.

[9] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

[10] Mark F. Medress, Franklin S Cooper, Jim W. Forgie, CC Green, Dennis H. Klatt, Michael H. O'Malley, Edward P Neuburg, Allen Newell, DR Reddy, B Ritea, et al. Speech understanding systems: Report of a steering committee. *Artificial Intelligence*, 9(3):307–316, 1977.

[11] Yimeng Min, Yiwei Bai, and Carla P Gomes. Unsupervised learning for solving the travelling salesman problem. *Advances in Neural Information Processing Systems*, 36:47264–47278, 2023.

[12] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017.

[13] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid Von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. Solving mixed integer programs using neural networks. *arXiv preprint arXiv:2012.13349*, 2020.

[14] Wenbin Ouyang, Yisen Wang, Paul Weng, and Shaochen Han. Generalization in deep rl for tsp problems via equivariance and local search. *SN Computer Science*, 5(4):369, 2024.

[15] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

[16] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[17] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018.