Graph Neural Networks for the Travelling Salesman Problem

Chaitanya K. Joshi\textsuperscript{[1]}, Thomas Laurent\textsuperscript{[2]}, and Xavier Bresson\textsuperscript{[1]}

\textsuperscript{[1]} NTU, Singapore, \textsuperscript{[2]} LMU, LA, USA

Boosting Combinatorial Optimization using Machine Learning

(Session at the INFORMS Annual Meeting 2019)

22\textsuperscript{nd} October, 2019
Motivation

• **Operations Research**: solvers for NP-Hard combinatorial problems
  - Backbone of modern industries such as transportation, scheduling, logistics

• **Good OR solvers**
  - expert intuition/domain knowledge
  - years of trial-and-error
Motivation

• Operations Research: solvers for NP-Hard combinatorial problems - Backbone of modern industries such as transportation, scheduling, logistics

• Good OR solvers - expert intuition/domain knowledge - years of trial-and-error

  "We believe that expert intuition can be automated and augmented through Machine Learning" — Bengio, Lodi, Prouvost, 2018 [1]

This talk

• Advances in **end-to-end learning** for OR solvers
  - Results on TSP: intensively studied, practical class of routing problems

• Our focus/specialty: **Graph Neural Networks**
  - New tools for operating directly on the graph structure of problems
Our contributions

An Efficient Graph ConvNet for TSP: arxiv.org/abs/1906.01227

TSP as a graph problem

“Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?”

• Concorde Solver\(^1\): leverages 30 years of research
  - Cutting plane algorithms to iteratively solve linear relaxations
  - Branch-and-bound to reduce solution search space

• End-to-end learning for TSP\(^2,3\): Proof-of-concept for learning previously un-encountered NP-Hard problems

\(^1\) Applegate, Bixby, Chvátal, Cook, The traveling salesman problem: a computational study, 2006
\(^2\) Vinyals, Fortunato, Jaitly, Pointer networks, NeurIPS 2015
\(^3\) Bello, Pham, Le, Norouzi, Bengio, Neural combinatorial optimization with reinforcement learning, ICLR 2017
End-to-end pipeline for OR problems

1. Data
   Define the problem using graphs

2. Embed
   Obtain dense representations of nodes and edges using GNN model

3. Predict
   Compute probability of nodes/edges belonging to the solution

4. Search
   Enforce feasibility and constraints through graph search

5. Train
   Learn prediction policy through imitation (SL) or experience (RL)

This generic pipeline has been used to tackle TSP, MVC, MaxCut, MIS, VRPs, SAT, etc.
Graph Embedding: features
Graph Embedding: message passing

Layer 1

Layer 2
Graph Embedding: aggregation
Vanilla Graph ConvNets \[1,2\]

\[
h_{i}^{\ell+1} = f_{G-VCNN}^{\ell} ( h_{i}^{\ell} , \{ h_{j}^{\ell} : j \to i \} ) = \text{ReLU} \left( U^{\ell} h_{i}^{\ell} + V^{\ell} \sum_{j \to i} h_{j}^{\ell} \right)
\]

\[1\] Sukhbaatar, Szlam, Fergus, Learning multiagent communication with backpropagation, NeurIPS 2016

\[2\] Kipf, Welling, Semi-supervised classification with graph convolutional networks, ICLR 2017
Residual Gated Graph ConvNets $^{[1,2]}$

\[ x_i^{\ell+1} = x_i^\ell + \text{ReLU}(\text{BN}(W_1^\ell x_i^\ell + \sum_{j \sim i} \eta_{ij}^\ell \circ W_2^\ell x_j^\ell)) \]

\[ \eta_{ij}^\ell = \frac{\sigma(e_{ij}^\ell)}{\sum_{j \sim i} \sigma(e_{ij'}^\ell) + \varepsilon} \]

\[ e_{ij}^{\ell+1} = e_{ij}^\ell + \text{ReLU}(\text{BN}(V_1^\ell e_{ij}^\ell + V_2^\ell x_i^\ell + V_3^\ell x_j^\ell)) \]

$^{[1]}$ Bresson, Laurent, Residual gated graph convnets, ICLR 2018

$^{[2]}$ Joshi, Laurent, Bresson, An efficient graph convolutional network technique for the travelling salesman problem, arXiv 2019
Prediction: does an edge belong to the optimal tour?
Prediction: probability distribution over edges
Prediction: Non-autoregressive approach [1]

Predictions for all edges are
- produced in **one shot**
- independent of each other

Search for feasible solutions
Search for feasible solutions

- We can use any search algorithm for graphs + enforce problem constraints:
  - Greedy search
  - Beam search
  - Monte Carlo tree search

- Analogous to search in machine translation\textsuperscript{[1]} or game playing\textsuperscript{[2]}

\textsuperscript{[1]} Wu et al., Google’s neural machine translation system, arXiv 2016
\textsuperscript{[2]} Silver et al., Mastering the game of Go with deep neural networks and tree search, Nature 2016
Alternate: **Autoregressive decoding**\(^1\) with Attention\(^{2,3}\)

---

\(^1\) Khalil, Dai, Zhang, Dilkina, Song, Learning combinatorial optimization algorithms over graphs, NeurIPS 2017

\(^2\) Deudon, Cournut, Lacoste, Adulyasak, Rousseau, Learning heuristics for the tsp by policy gradient, 2018

\(^3\) Kool, van Hoof, Welling, Attention, learn to solve routing problems!, ICLR 2019
Training the policy

Learning by **Imitation (SL)**

- Minimize the loss between optimal solutions (Concorde) and model’s prediction
- Binary classification problem on edges

Learning by **Exploration (RL)**

- Use REINFORCE (policy gradient) to minimize the length of the tour predicted by the model at the end of decoding

And there are trade-offs for both...
Experiments

Current paradigm: Models are trained and evaluated on TSP instances of fixed sizes: 20, 50 and 100 nodes
Performance on fixed TSP

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Concorde</td>
<td>Specific</td>
<td>3.84</td>
<td>0.00%</td>
<td>(1m)</td>
<td>5.70</td>
<td>0.00%</td>
<td>(2m)</td>
<td>7.76</td>
</tr>
<tr>
<td>Gurobi</td>
<td>Generic</td>
<td>3.84</td>
<td>0.00%</td>
<td>(7s)</td>
<td>5.70</td>
<td>0.00%</td>
<td>(2m)</td>
<td>7.76</td>
</tr>
<tr>
<td>Bello et al. (2016)</td>
<td>AR, RNN, RL</td>
<td>3.84</td>
<td>0.10%</td>
<td>5.75</td>
<td>95%</td>
<td>8.00</td>
<td>3.03%</td>
<td>--</td>
</tr>
<tr>
<td>Deudon et al. (2018)</td>
<td>AR, GAT, RL</td>
<td>3.84</td>
<td>0.11%</td>
<td>(5m)</td>
<td>5.77</td>
<td>1.28%</td>
<td>(17m)</td>
<td>8.75</td>
</tr>
<tr>
<td>Deudon et al. (2018)</td>
<td>(+ 2OPT)</td>
<td>3.84</td>
<td>0.09%</td>
<td>(6m)</td>
<td>5.75</td>
<td>1.00%</td>
<td>(32m)</td>
<td>8.12</td>
</tr>
<tr>
<td>Kool et al. (2019)</td>
<td>AR, GAT, RL</td>
<td>3.84</td>
<td>0.08%</td>
<td>(5m)</td>
<td>5.73</td>
<td>0.52%</td>
<td>(24m)</td>
<td>7.94</td>
</tr>
<tr>
<td>Ours (arXiv ’19)</td>
<td>NAR, GCN, SL</td>
<td>3.84</td>
<td>0.10%</td>
<td>(20s)</td>
<td>5.71</td>
<td>0.26%</td>
<td>(2m)</td>
<td>7.92</td>
</tr>
<tr>
<td>Ours (arXiv ’19)</td>
<td>(+ shortest h.)</td>
<td>3.84</td>
<td>0.01%</td>
<td>(12m)</td>
<td>5.70</td>
<td>0.01%</td>
<td>(18m)</td>
<td>7.87</td>
</tr>
</tbody>
</table>

End-to-end solvers can’t compete with OR solvers yet, but...

**Performance:** Supervised learning?  
**Speed:** Non-autoregressive?
## Performance on fixed TSP

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Concorde</td>
<td>Specific</td>
<td>3.84</td>
<td>0.00%</td>
<td>(1m)</td>
<td>5.70</td>
<td>0.00%</td>
<td>(2m)</td>
<td>7.76</td>
</tr>
<tr>
<td>Gurobi</td>
<td>Generic</td>
<td>3.84</td>
<td>0.00%</td>
<td>(7s)</td>
<td>5.70</td>
<td>0.00%</td>
<td>(2m)</td>
<td>7.76</td>
</tr>
<tr>
<td>Bello et al. (2016)</td>
<td>AR, RNN, RL</td>
<td>3.84</td>
<td>0.10%</td>
<td>–</td>
<td>5.75</td>
<td>0.95%</td>
<td>–</td>
<td>8.00</td>
</tr>
<tr>
<td>Deudon et al. (2018)</td>
<td>AR, GAT, RL</td>
<td>3.84</td>
<td>0.11%</td>
<td>(5m)</td>
<td>5.77</td>
<td>1.28%</td>
<td>(17m)</td>
<td>8.75</td>
</tr>
<tr>
<td>Deudon et al. (2018)</td>
<td>(+ 2OPT)</td>
<td>3.84</td>
<td>0.09%</td>
<td>(6m)</td>
<td>5.75</td>
<td>1.00%</td>
<td>(32m)</td>
<td>8.12</td>
</tr>
<tr>
<td>Kool et al. (2019)</td>
<td>AR, GAT, RL</td>
<td>3.84</td>
<td>0.08%</td>
<td>(5m)</td>
<td>5.73</td>
<td>0.52%</td>
<td>(24m)</td>
<td>7.94</td>
</tr>
<tr>
<td><strong>Ours (arXiv ’19)</strong></td>
<td>NAR, GCN, SL</td>
<td><strong>3.84</strong></td>
<td><strong>0.10%</strong></td>
<td>(20s)</td>
<td><strong>5.71</strong></td>
<td><strong>0.26%</strong></td>
<td>(2m)</td>
<td>7.92</td>
</tr>
<tr>
<td><strong>Ours (arXiv ’19)</strong></td>
<td>(+ shortest h.)</td>
<td>3.84</td>
<td>0.01%</td>
<td>(12m)</td>
<td>5.70</td>
<td>0.01%</td>
<td>(18m)</td>
<td>7.87</td>
</tr>
<tr>
<td>Ours (NeurIPS ’19)</td>
<td>AR, GAT, SL</td>
<td>3.84</td>
<td>0.01%</td>
<td>(5m)</td>
<td>5.70</td>
<td>0.04%</td>
<td>(24m)</td>
<td>7.86</td>
</tr>
<tr>
<td>Ours (NeurIPS ’19)</td>
<td>AR, GAT, RL</td>
<td>3.84</td>
<td>0.09%</td>
<td>(5m)</td>
<td>5.72</td>
<td>0.59%</td>
<td>(24m)</td>
<td>7.97</td>
</tr>
<tr>
<td>Ours (NeurIPS ’19)</td>
<td>AR, GCN, SL</td>
<td>3.84</td>
<td>0.01%</td>
<td>(5m)</td>
<td>5.72</td>
<td>0.27%</td>
<td>(24m)</td>
<td>7.96</td>
</tr>
<tr>
<td>Ours (NeurIPS ’19)</td>
<td>AR, GCN, RL</td>
<td>3.84</td>
<td>0.10%</td>
<td>(5m)</td>
<td>5.73</td>
<td>0.76%</td>
<td>(24m)</td>
<td>7.97</td>
</tr>
</tbody>
</table>

**Performance:** Supervised learning

**Speed:** Non-autoregressive
For small instances, the model is able to confidently identify most of the tour edges in the edge probability distribution without beam search.
As instance size increases, edge probability distributions reflect the combinatorial explosion in TSP
Beam search is essential for finding the optimal tour for more complex instances.
Beam search is essential for finding the optimal tour for more complex instances. But what about Generalization?

“...whether or not a learned policy performs decently on a different problem distribution” (in terms of structure or size)

- Bengio, Lodi, Prouvost, 2018
Generalization to variable TSP sizes

Optimality gap vs. TSP size, for NAR models when using beam search (with width = 1,280)
Generalization: impact of architecture

Optimality gap vs. TSP size, for **NAR and AR models** (both trained with SL)
Generalization to variable TSP sizes

Speed: Non-autoregressive

Generalization: Autoregressive

Optimality gap vs. TSP size for **NAR and AR models** (both trained with SL)
Generalization: impact of learning paradigm

Optimality gap vs. TSP size, for AR models trained with RL and SL
Generalization: large-scale instances

Optimality gap vs. TSP size, for AR models trained with RL and SL
Generalization: large-scale instances

Performance: Supervised learning

Generalization: Reinforcement learning

(for AR architecture)

Optimality gap vs. TSP size, for AR models trained with RL and SL
End-to-end pipeline for OR problems

1. Data
   Use variable problem sizes for training?

2. Embed
   How to design scale-invariant GNN models?

3. Predict
   Trade-offs between AR and NAR decoders

4. Search
   Do we need more powerful search algorithms?

5. Train
   How to do curriculum learning and finetuning?

Next steps:
Where can we innovate for better scale-invariant generalization?
Questions?

Get the slides:

Chaitanya Joshi
Research Assistant
Nanyang Technological University

chaitanya.joshi@ntu.edu.sg
chaitjo.github.io
twitter.com/chaitjo
github.com/chaitjo