Attention Solves your TSP

Paper review & Research progress

Truong Son Hy *

*Department of Computer Science
The University of Chicago

Ryerson Physical Lab

Introduction

Objective

Learn heuristics for combinatorial optimization (**NP-hard problems**):

- Travelling Salesman Problem (TSP)
- Vehicle Routing Problem (VRP)
- Orienteering Problem (OP)
- (Stochastic) Prize Collecting TSP (PCTSP)

Introduction

Objective

Learn heuristics for combinatorial optimization (**NP-hard problems**):

- Travelling Salesman Problem (TSP)
- Vehicle Routing Problem (VRP)
- Orienteering Problem (OP)
- (Stochastic) Prize Collecting TSP (PCTSP)

The paper's proposal

- Model based on attention layers with benefits over the Pointer Network.
- Train the model using **REINFORCE** with a simple baseline based on a deterministic greedy rollout.

NP-hard and NP-complete

NP-hard

- TSP is an NP-hard (non-deterministic polynomial-time hardness) problem.
- If I give you a solution, you cannot check whether or not that solution is optimal by any polynomial-time algorithm.

NP-hard and NP-complete

NP-hard

- TSP is an NP-hard (non-deterministic polynomial-time hardness) problem.
- If I give you a solution, you cannot check whether or not that solution is optimal by any polynomial-time algorithm.

NP-complete

- Deciding if there is a Hamiltonian path/cycle in a graph is an NP-complete problem.
- If I give you a solution, you can check whether or not it is a valid Hamiltonian path/cycle by a simple polynomial-time algorithm.
- Problem of deciding if a graph has a Hamiltonian path/cycle can be **reduced** to finding one.

NP-complete

NP-complete problems

- Boolean satisfiability problem (SAT)
- Hamiltonian path/cycle problem
- Clique problem
- Vertex cover problem

NP-complete

NP-complete problems

- Boolean satisfiability problem (SAT)
- Hamiltonian path/cycle problem
- Clique problem
- Vertex cover problem

P = NP?

- 2-SAT problem has a polynomial-time algorithm, but not for k-SAT with $k > 2$.
- 3-SAT can be **reduced** to Hamiltonian path/cycle problem, and vice versa. **Reduction** is a transformation by a polynomial-time algorithm from one problem to another.
- 3-SAT is called to be **Turing equivalent** to Hamiltonian path/cycle problem.

NP-complete

P = NP?

- If we can offer a polynomial-time algorithm to solve **any** NP-complete problem, then we can solve **all others** (because they are Turing-equivalent) and we can prove that P = NP.

NP-complete

P = NP?

- If we can offer a polynomial-time algorithm to solve **any** NP-complete problem, then we can solve **all others** (because they are Turing-equivalent) and we can prove that P = NP.
- Any neural network finding a solution of the Hamiltonian path/cycle problem that has complexity $O(|V|^c)$ where $|V|$ is the size of the graph and c is a constant independent of $|V|$ is considered as a polynomial-time algorithm.

NP-complete

P = NP?

- If we can offer a polynomial-time algorithm to solve **any** NP-complete problem, then we can solve **all others** (because they are Turing-equivalent) and we can prove that P = NP.
- Any neural network finding a solution of the Hamiltonian path/cycle problem that has complexity $O(|V|^c)$ where $|V|$ is the size of the graph and c is a constant independent of $|V|$ is considered as a polynomial-time algorithm.

My proposal

- Investigate how powerful is graph neural networks?
- How to achieve **global** understanding of the graph?
- How much affect **global** has with respect to **local** empirically?

Attention Model

We define a problem instance s as a graph with n nodes, where node $i \in \{1, 2, .., n\}$ is represented by features x_i that is the coordinates. The graph is fully connected. We define a solution (tour) $\boldsymbol{\pi} = (\pi_1, .., \pi_n)$ as a permutation of the nodes.

Our attention based encoder-decoder model defines a stochastic policy $p(\boldsymbol{\pi}|s)$ for selecting a solution given a problem instance s (factorized and parameterized by θ):

$$p_\theta(\boldsymbol{\pi}|s) = \prod_{t=1}^{n} p_\theta(\pi_t|s, \boldsymbol{\pi}_{1:t-1})$$

The encoder produces embeddings of all input nodes. The decoder produces the sequence $\boldsymbol{\pi}$ of input nodes, one node at a time.

Encoder (1)

From the d_x-dimensional input features x_i (for TSP $d_x = 2$), the encoder compute initial d_h-dimensional node embeddings $h_i^{(0)}$ (in experiment, $d_h = 128$):

$$h_i^{(0)} = W^x x_i + b^x$$

Denote with $h_i^{(\ell)}$ the node embeddings produced by layer $\ell \in \{1, .., N\}$.

Encoder (1)

From the d_x-dimensional input features x_i (for TSP $d_x = 2$), the encoder compute initial d_h-dimensional node embeddings $h_i^{(0)}$ (in experiment, $d_h = 128$):

$$h_i^{(0)} = W^x x_i + b^x$$

Denote with $h_i^{(\ell)}$ the node embeddings produced by layer $\ell \in \{1, .., N\}$.

Encoder architecture

Transformer architecture in which each attention layer consist of two sublayers:

- a **multi-head attention** (MHA) layer that executes message passing between the nodes.
- a node-wise **fully connected feed-forward** (FF) layer.
- each sublayer has **skip-connection** and **batch normalization** (BN) that works better than layer normalization.

Encoder (2)

$$\hat{h}_i = BN^{\ell}\left(h_i^{(\ell-1)} + MHA_i^{\ell}\left(h_1^{(\ell-1)}, .., h_n^{(\ell-1)}\right)\right)$$

$$h_i^{(\ell)} = BN^{\ell}\left(\hat{h}_i + FF^{\ell}(\hat{h}_i)\right)$$

- MHA = multi-head attention
- FF = fully connected feed-forward
- BN = batch normalization

The encoder computes an aggregated embedding $\bar{h}^{(N)}$ of the input graph as the mean of the final node embeddings $h_i^{(N)}$:

$$\bar{h}^{(N)} = \frac{1}{n}\sum_{i=1}^{n} h_i^{(N)}$$

Context embedding (1)

The context of the decoder at time t comes from the encoder and the output up to time t consisting of the embedding of: (i) the graph, (ii) the previous (last) node π_{t-1}, and (iii) the first node π_1.

- $t = 1$: $h_{(c)}^{(N)} = [h^{(N)}, v^1, v^f]$ where v^1 and v^f are input placeholders.
- $t > 1$: $h_{(c)}^{(N)} = [\bar{h}^{(N)}, h_{\pi_{t-1}}^{(N)}, h_{\pi_1}^{(N)}]$

A new context node embedding $h_{(c)}^{(N+1)}$ using the (M-head) attention mechanism. Compute a single query $q_{(c)}$ (per head) from the context node:

$$q_{(c)} = W^Q h_{(c)} \qquad k_i = W^K h_i$$

Context embedding (2)

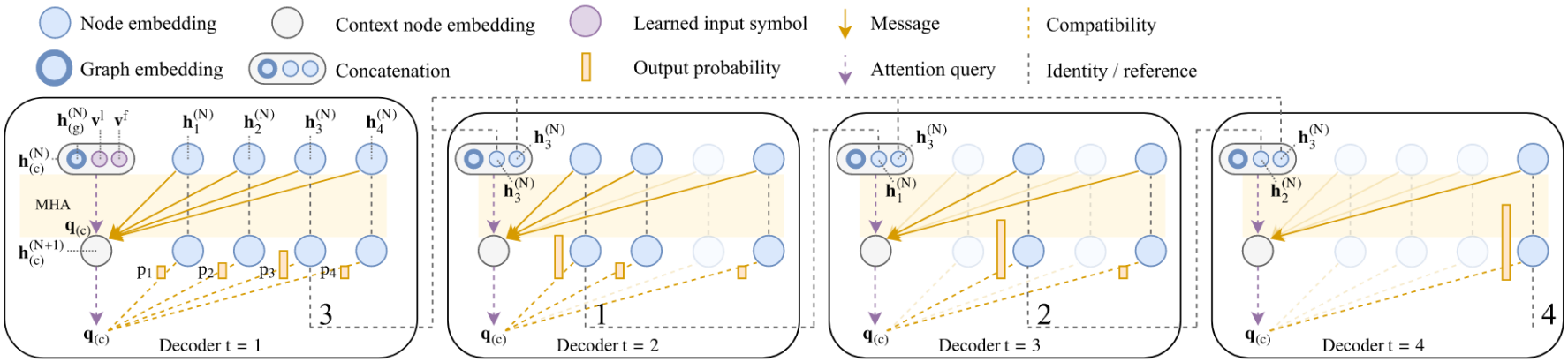
Mask (set $u_{(c)j} = -\infty$ nodes which cannot be visited at time t, otherwise set:

$$u_{(c)j} = \frac{q_{(c)}^T k_j}{\sqrt{d_k}}$$

where $d_k = \frac{d_h}{M}$ is the query/key dimensionality. We output probability vector p using a softmax:

$$p_i = p_\theta(\pi_t = i | s, \boldsymbol{\pi}_{1:t-1}) = \frac{e^{u_{(c)i}}}{\sum_j e^{u_{(c)j}}}$$

Context embedding (3)



Reinforce with greedy rollout baseline (1)

We define the loss $\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[L(\pi)]$ that is the expectation of the cost $L(\pi)$ (tour length for TSP). We optimize $\mathcal{L}$ by gradient descent, using the REINFORCE gradient estimator with baseline $b(s)$:

$$\bigtriangledown\mathcal{L}(\theta|s) = \mathbb{E}_{p_\theta(\pi|s)}[(L(\pi) - b(s)) \bigtriangledown \log p_\theta(\pi|s)]$$

where $b(s)$ is the cost of a solution from a **deterministic greedy rollout** of the policy defined by the best model so far. **Self-critical training** with an exponential moving average:

$$b(s) \leftarrow \beta b(s) + (1 - \beta)L(\pi)$$

Reinforce with greedy rollout baseline (2)

Algorithm 1 REINFORCE with Rollout Baseline

Input: number of epochs E, steps per epoch T, batch size B, significance α
Init $\boldsymbol{\theta}$, $\boldsymbol{\theta}^{\mathrm{BL}} \leftarrow \boldsymbol{\theta}$
for epoch $= 1, \ldots, E$ **do**
  for step $= 1, \ldots, T$ **do**
    $s_i \leftarrow \mathrm{RandomInstance}()\ \forall i \in \{1, \ldots, B\}$
    $\boldsymbol{\pi}_i \leftarrow \mathrm{SampleRollout}(s_i, p_{\boldsymbol{\theta}})\ \forall i \in \{1, \ldots, B\}$
    $\boldsymbol{\pi}_i^{\mathrm{BL}} \leftarrow \mathrm{GreedyRollout}(s_i, p_{\boldsymbol{\theta}^{\mathrm{BL}}})\ \forall i \in \{1, \ldots, B\}$
    $\nabla \mathcal{L} \leftarrow \sum_{i=1}^{B} \left(L(\boldsymbol{\pi}_i) - L(\boldsymbol{\pi}_i^{\mathrm{BL}})\right) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\pi}_i)$
    $\boldsymbol{\theta} \leftarrow \mathrm{Adam}(\boldsymbol{\theta}, \nabla \mathcal{L})$
  end for
  if $\mathrm{OneSidedPairedTTest}(p_{\boldsymbol{\theta}}, p_{\boldsymbol{\theta}^{\mathrm{BL}}}) < \alpha$ **then**
    $\boldsymbol{\theta}^{\mathrm{BL}} \leftarrow \boldsymbol{\theta}$
  end if
end for

Reinforce with greedy rollout baseline (2)

Algorithm 1 REINFORCE with Rollout Baseline

Input: number of epochs E, steps per epoch T, batch size B, significance α
Init $\boldsymbol{\theta}$, $\boldsymbol{\theta}^{\mathrm{BL}} \leftarrow \boldsymbol{\theta}$
for epoch $= 1, \ldots, E$ **do**
  for step $= 1, \ldots, T$ **do**
    $s_i \leftarrow \mathrm{RandomInstance}()\ \forall i \in \{1, \ldots, B\}$
    $\boldsymbol{\pi}_i \leftarrow \mathrm{SampleRollout}(s_i, p_{\boldsymbol{\theta}})\ \forall i \in \{1, \ldots, B\}$
    $\boldsymbol{\pi}_i^{\mathrm{BL}} \leftarrow \mathrm{GreedyRollout}(s_i, p_{\boldsymbol{\theta}^{\mathrm{BL}}})\ \forall i \in \{1, \ldots, B\}$
    $\nabla \mathcal{L} \leftarrow \sum_{i=1}^{B} \left(L(\boldsymbol{\pi}_i) - L(\boldsymbol{\pi}_i^{\mathrm{BL}})\right) \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\boldsymbol{\pi}_i)$
    $\boldsymbol{\theta} \leftarrow \mathrm{Adam}(\boldsymbol{\theta}, \nabla \mathcal{L})$
  end for
  if $\mathrm{OneSidedPairedTTest}(p_{\boldsymbol{\theta}}, p_{\boldsymbol{\theta}^{\mathrm{BL}}}) < \alpha$ **then**
    $\boldsymbol{\theta}^{\mathrm{BL}} \leftarrow \boldsymbol{\theta}$
  end if
end for

Alternative to the greedy rollout

Why don't we choose Genetic Algorithm or Ant Colony as the baseline $b(s)$ and apply **Q-Learning** instead?

Drawbacks

Local vs. Global

- Usually, people only apply few layers of message passing that can only capture **local** structure.

Drawbacks

Local vs. Global

- Usually, people only apply few layers of message passing that can only capture **local** structure.
- The solution of Traveling Saleman Problem (TSP) requires an understanding of the **global** structure.

Drawbacks

Local vs. Global

- Usually, people only apply few layers of message passing that can only capture **local** structure.
- The solution of Traveling Saleman Problem (TSP) requires an understanding of the **global** structure.
- For two furthest vertices to exchange messages, it would require $|V| - 1$ iterations of message passing.

Drawbacks

Local vs. Global

- Usually, people only apply few layers of message passing that can only capture **local** structure.
- The solution of Traveling Saleman Problem (TSP) requires an understanding of the **global** structure.
- For two furthest vertices to exchange messages, it would require $|V| - 1$ iterations of message passing.
- Therefore, we need $|V| - 1$ layers of message passing neural networks.

Drawbacks

Local vs. Global

- Usually, people only apply few layers of message passing that can only capture **local** structure.
- The solution of Traveling Saleman Problem (TSP) requires an understanding of the **global** structure.
- For two furthest vertices to exchange messages, it would require $|V| - 1$ iterations of message passing.
- Therefore, we need $|V| - 1$ layers of message passing neural networks.
- However, in theory, it is proven that the Weisfeiler-Lehman isomorphism test only needs $O(\log |V|)$ iterations. What would be the optimal number of layers? – Open question!

Drawbacks

Local vs. Global

- Usually, people only apply few layers of message passing that can only capture **local** structure.
- The solution of Traveling Saleman Problem (TSP) requires an understanding of the **global** structure.
- For two furthest vertices to exchange messages, it would require $|V| - 1$ iterations of message passing.
- Therefore, we need $|V| - 1$ layers of message passing neural networks.
- However, in theory, it is proven that the Weisfeiler-Lehman isomorphism test only needs $O(\log |V|)$ iterations. What would be the optimal number of layers? – Open question!
- To train a deep network of many layers, we need **skip connection** like to train with Recurrent Neural Networks.

Graph Transformer Networks

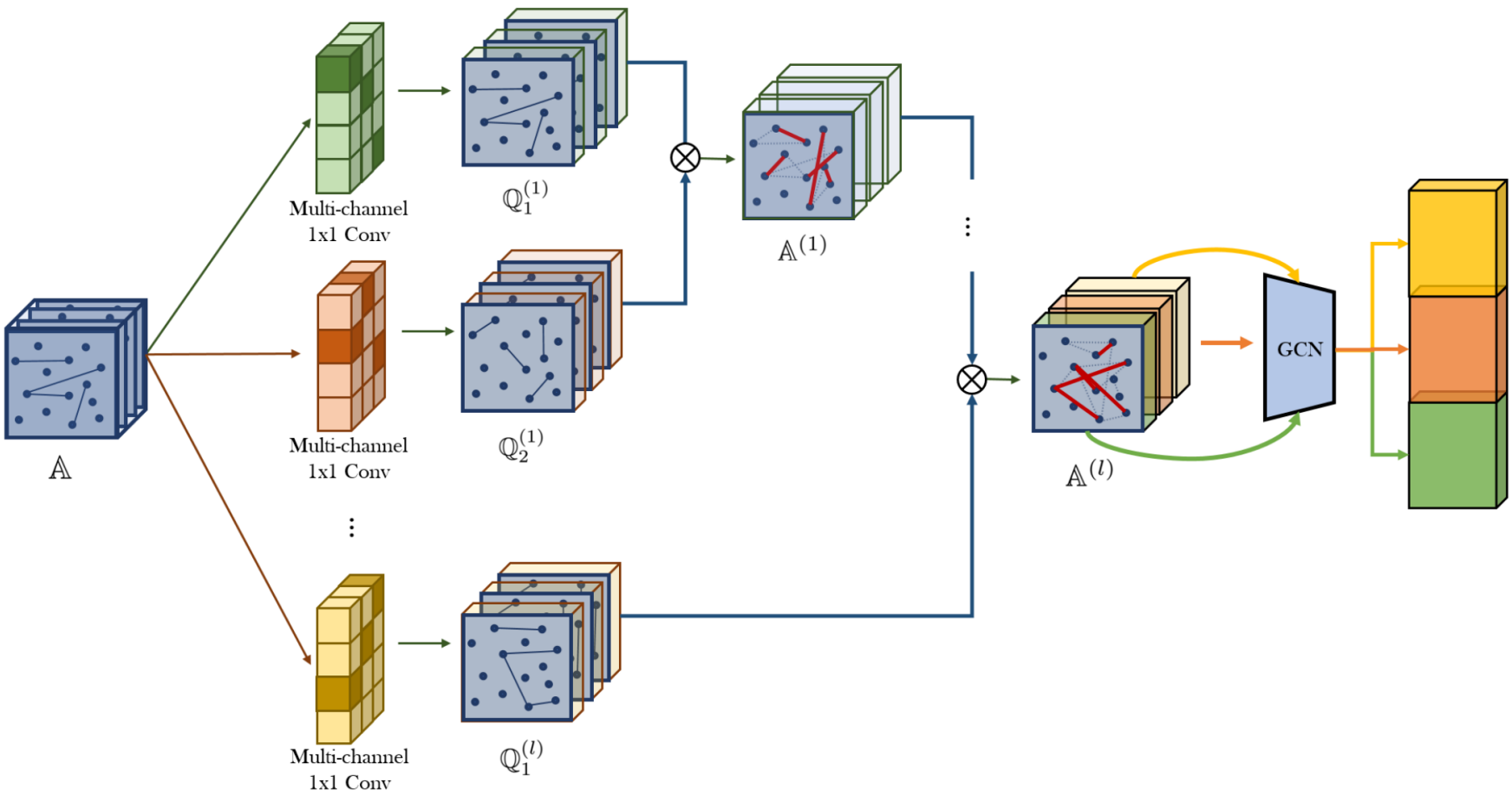


Figure 2: Graph Transformer Networks (GTNs) learn to generate a set of new meta-path adjacency matrices $\mathbb{A}^{(l)}$ using GT layers and perform graph convolution as in GCNs on the new graph structures. Multiple node representations from the same GCNs on multiple meta-path graphs are integrated by concatenation and improve the performance of node classification. $\mathbb{Q}_1^{(l)}$ and $\mathbb{Q}_2^{(l)} \in \mathbf{R}^{N \times N \times C}$ are intermediate adjacency tensors to compute meta-paths at the lth layer.

Proposal architectures

Encoder - Proposal 1

MPNN of $|V| - 1$ layers and skip connection.

Proposal architectures

Encoder - Proposal 1

MPNN of $|V| - 1$ layers and skip connection.

Encoder - Proposal 2

(Transformer idea) $|V|$ MPNNs of $|V| - 1$ layers in which k-th MPNN has input as A^k. The outputs of all MPNNs are concatenated.

Proposal architectures

Encoder - Proposal 1

MPNN of $|V| - 1$ layers and skip connection.

Encoder - Proposal 2

(Transformer idea) $|V|$ MPNNs of $|V| - 1$ layers in which k-th MPNN has input as A^k. The outputs of all MPNNs are concatenated.

Decoder

- The solution of TSP and Hamiltonian problem is represented by an adjacency matrix.
- (From Variational Graph Autoencoder - VGAE) Given the output from the encoder (or latent representation) Z. The decoder gives the output adjacency as:

$$\hat{A} = \sigma(ZZ^T)$$

where σ is the Sigmoid function.

TSP - data generation (1)

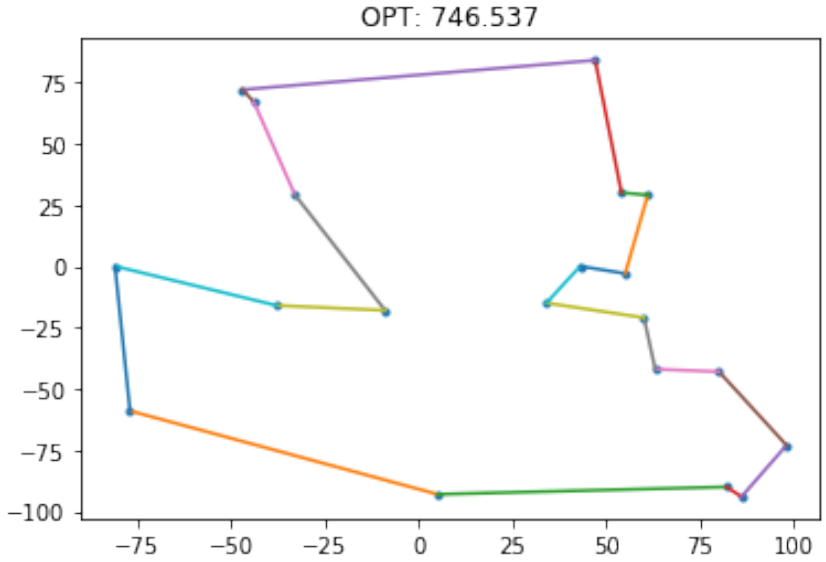
Input

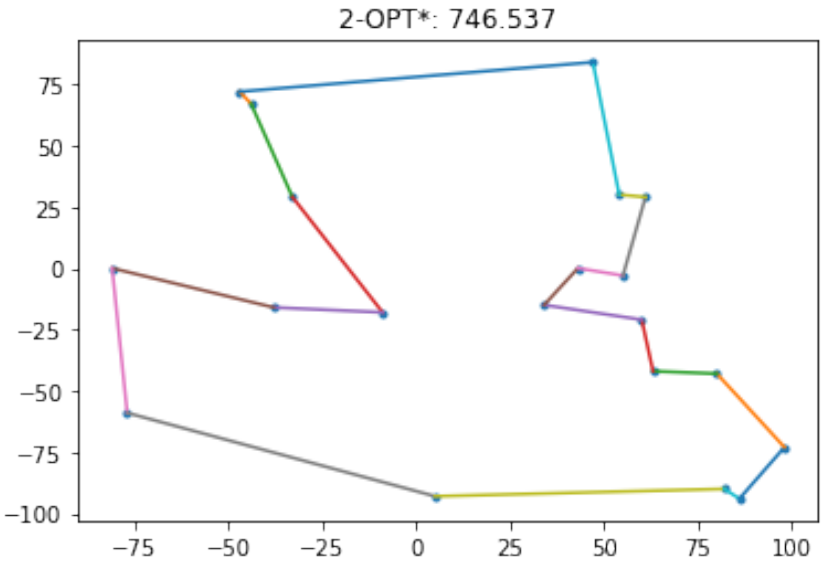
- Random N points on the 2D plane (coordinates range from -100 to 100). In our experiments, $N \in \{5, 10, 20\}$.
- The input adjacency matrix is the normalized **Euclidean-distance matrix**.

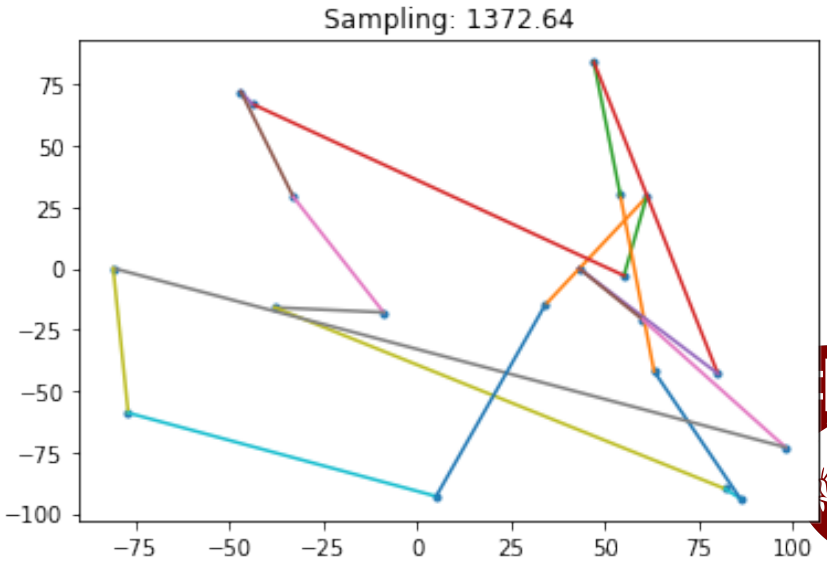
Baselines

- **OPT**: Optimal solution found by Dynamic Programming $O(N^2 \times 2^N)$.
- **Sampling**: Random a solution (permutation) 10,000 times and select the best.
- **2-OPT**: Random a solution (permutation) and iterative fix that solution until there is no **crossing**. We can prove that the factor is less than 2.
- **2-OPT***: Repeat 2-OPT 10,000 times and select the best.

TSP - data generation (2)







TSP - result

It does **not** work! Sorry!

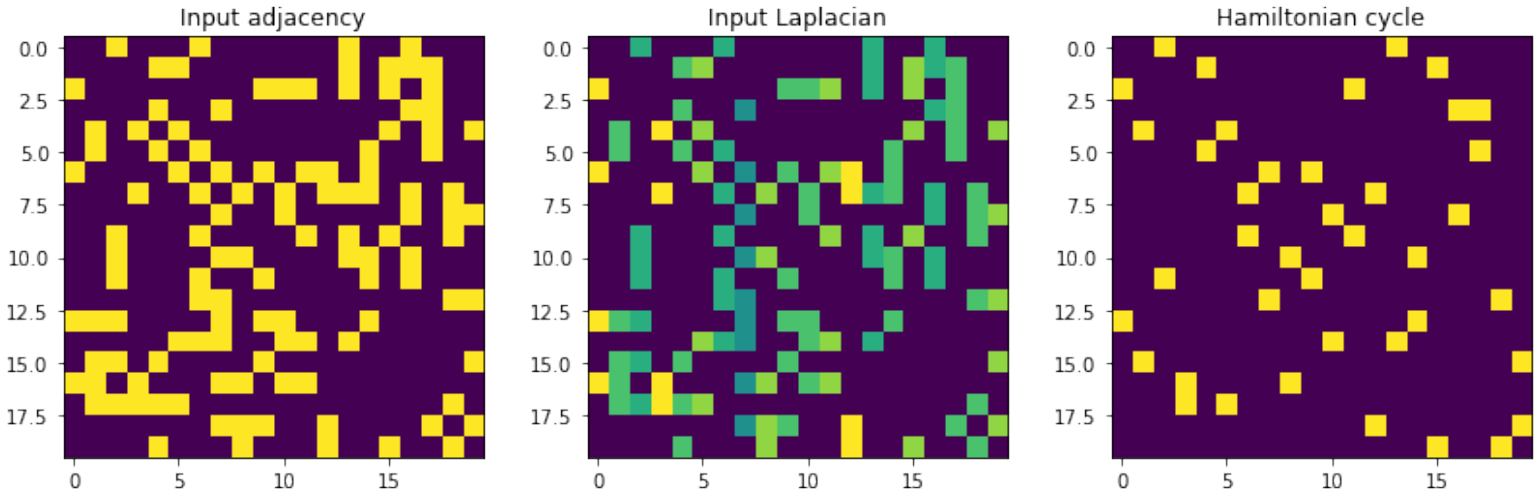
Method	N = 5	N = 10	N = 20
Sampling/OPT	1.0	1.07	1.78
2-OPT*/OPT	1.0	1.0	1.0
Proposal 1/OPT	?	?	?
Proposal 2/OPT	?	?	?

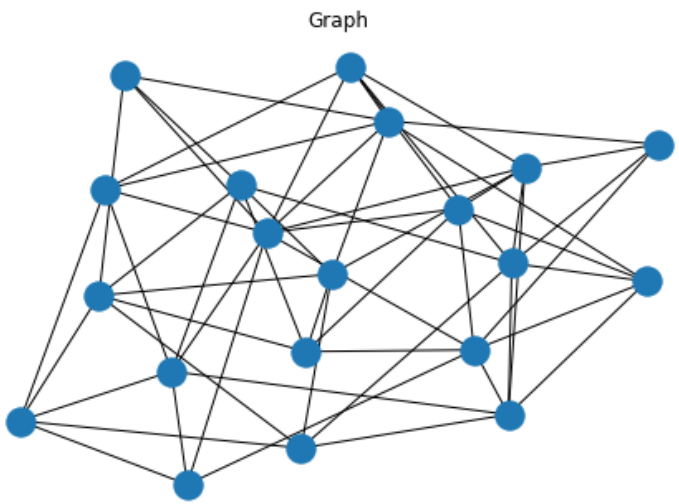
Hamiltonian cycle finding - data generation (1)

Data generation

1. Random a permutation p that determines a Hamiltonian cycle
2. Initialize the graph as that cycle
3. Random a number $m \in [0, N(N-1)/2]$
4. If $m > N$, add $m - N$ random edges to the graph, otherwise do nothing.

Hamiltonian cycle finding - data generation (2)





Hamiltonian cycle finding - result

Training, validation and testing sets contain 10,000 graphs each. The metrics is percentage of the number of graphs the networks can correctly output one of its Hamiltonian cycle. The baseline is **local** MPNN with only 3 iterations of message passing.

Method	N = 5	N = 10	N = 20
(**Local**) MPNN	86.97%	3.52%	7.4%
Proposal 1	86.97%	37.51%	20.52%
Proposal 2	**86.97%**	**41.16%**	**20.54%**
$2^{N(N-1)/2}$	1,024	3.5184372e+13	1.5692754e+57

What next?

Theoretical question

The message passing neural network is zero-th order. What is the limit/power of higher-order message passing? Covariant Compositional Networks?

Practical question

- Attention?
- What are the applications?
- What are the competing methods? For publication.

Q & A

Thank you very much for your **attention**!