Learning to Solve the Traveling Salesman Problem with Transformers

Xavier Bresson

School of Computer Science and Engineering
Data Science and AI Research Centre
Nanyang Technological University (NTU), Singapore

NATIONAL RESEARCH FOUNDATION
PRIME MINISTER'S OFFICE
SINGAPORE

Joint work with Thomas Laurent (LMU)

Seminar series on "Mathematics, Physics and Machine Learning"
Instituto Superior Técnico
Jan 27th 2021

Organizer : Prof. Mário Figueiredo

Outline

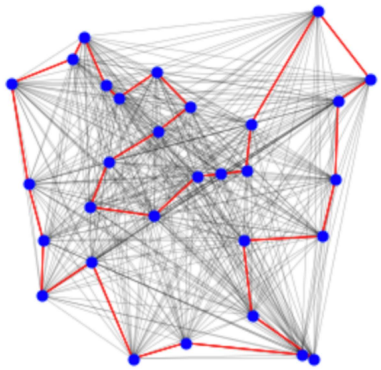
- History of TSP
- Traditional Solvers
- Neural Network Solvers
- Proposed Architecture
- Numerical Results
- Discussion
- Conclusion

Outline

- History of TSP
- Traditional Solvers
- Neural Network Solvers
- Proposed Architecture
- Numerical Results
- Discussion
- Conclusion

Traveling Salesman Problem (TSP)

- TSP : "Given a list of cities and the distances between each pair of cities, what is the shortest possible path that visits each city exactly once and returns to the origin city?" – First mathematical formulation by W. Hamilton 1805-1865.
- TSP belongs to the class of routing problems which are used every day in industry (warehouse, transportation, supply chain, hardware design, manufacturing, etc).
- TSP is NP-hard, exhaustive search has a complexity in $O(n!)$.
- TSP is the most popular and most studied combinatorial problems, starting with von Neumann (1951).
- TSP has driven the discovery of several important optimization techniques : Cutting Planes[1], Branch-and-Bound[2], Local Search[3], Lagrangian Relaxation[4], Simulated Annealing[5].



[1] Dantzig, Fulkerson, Johnson, 1954
[2] Bellman, Held, Karp, 1962
[3] Croes, A method for solving traveling-salesman problems, 1958
[4] Fisher, The Lagrangian Relaxation Method for Solving Integer Programming Problems, 1981
[5] Kirkpatrick, Gelatt, Vecchi, Optimization by Simulated Annealing, 1983

Outline

- History of TSP
- **Traditional Solvers**
- Neural Network Solvers
- Proposed Architecture
- Numerical Results
- Discussion
- Conclusion

Traditional TSP Solvers

- Two traditional approaches to tackle combinatorial problems :
  - Exact algorithms : Exhaustive search or Integer Programming. These algorithms are guaranteed to find optimal solutions, but they become intractable for n>20-40.
  - Approximate/heuristic algorithms : These algorithms trade optimality for computational efficiency. They are problem-specific, often designed by iteratively applying a simple man-crafted rule, known as heuristic. Their complexity is polynomial and their quality depends on an approximate ratio that characterizes the (worst/average-case) error w.r.t the optimal solution.

Exact TSP Solvers

- Exact algorithms :
  - Dynamic programming[1] : O($n^2\,2^n$), intractable for n>40.
  - Gurobi[2] : General purpose Integer Programming (IP) solver with Cutting Planes (CP) and Branch-and-Bound (BB).
  - Concorde[3] : Highly specialized linear IP+CP+BB for TSP. Concorde is widely regarded as the fastest exact TSP solver, for large instances, currently in existence.
  - TSP can be formulated as Integer Linear Programming (ILP) problem :

$$\min_x \sum_{e \in E} d_e x_e \quad \text{s.t.}$$

$$\sum_{e \in \text{Cut}(\{v\}, V-\{v\})} x_e = 2 \quad \forall v \in V$$

$$\sum_{e \in \text{Cut}(S, S^c)} x_e \geq 2 \quad \forall S \subset V, S \neq \emptyset, S \neq V$$

$$\text{where } \text{Cut}(A, A^c) = \{e_{vv'} \text{ s.t. } v \in A, v' \in A^c\}$$

$$x_e \in \{0, 1\} \quad \forall e \in E$$

[1] Held, Richard, A dynamic programming approach to sequencing problems, 1962
[2] Gu, Rothberg, Bixby, Gurobi, 2008
[3] Applegate, Bixby, Chvatal, Cook, The Traveling Salesman Problem: A Computational Study, 2006

IP Solvers

- **Interpretation :**

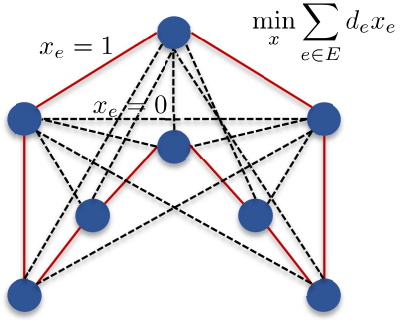
$$\min_x \sum_{e \in E} d_e x_e \quad \text{s.t.}$$

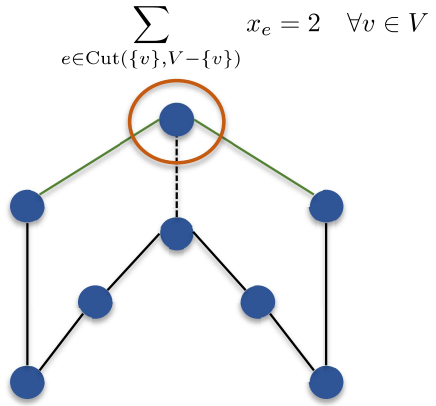
$$\sum_{e \in \text{Cut}(\{v\}, V-\{v\})} x_e = 2 \quad \forall v \in V$$

$$\sum_{e \in \text{Cut}(S, S^c)} x_e \geq 2 \quad \forall S \subset V, S \neq \emptyset, S \neq V$$

where $\text{Cut}(A, A^c) = \{e_{vv'} \text{ s.t. } v \in A, v' \in A^c\}$

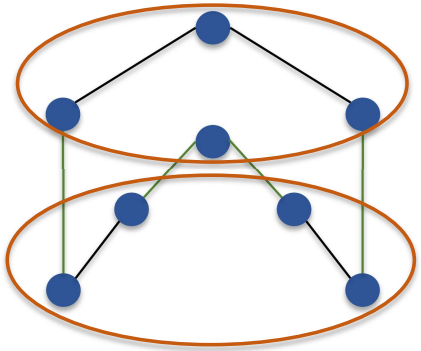
$x_e \in \{0, 1\} \quad \forall e \in E$



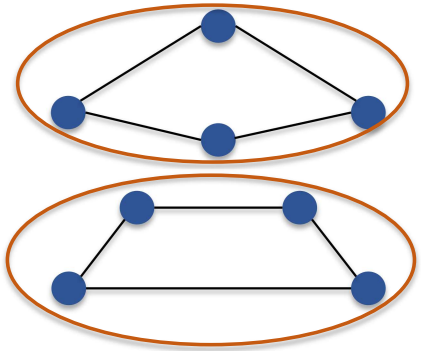


Sub-contour constraints that guarantee a feasible tour.

$$\sum_{e \in \text{Cut}(S, S^c)} x_e \geq 2 \quad \forall S \subset V$$



Satisfied

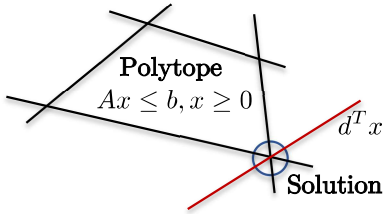


Not satisfied

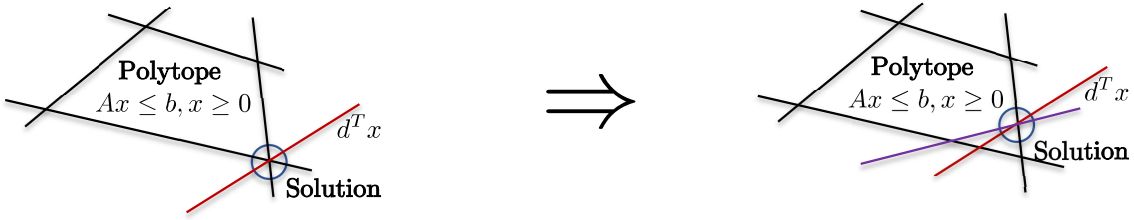
IP Solvers

- ILP problems are NP-hard because the space of optimization is binary $x_e \in \{0,1\}$.
- ILP can be relaxed as a Linear Programming (LP) problem[1] with $x_e \in [0,1]$ of the standard form :

$$\min_x \; d^T x \;\; \text{s.t.} \;\; Ax \leq b, x \geq 0$$



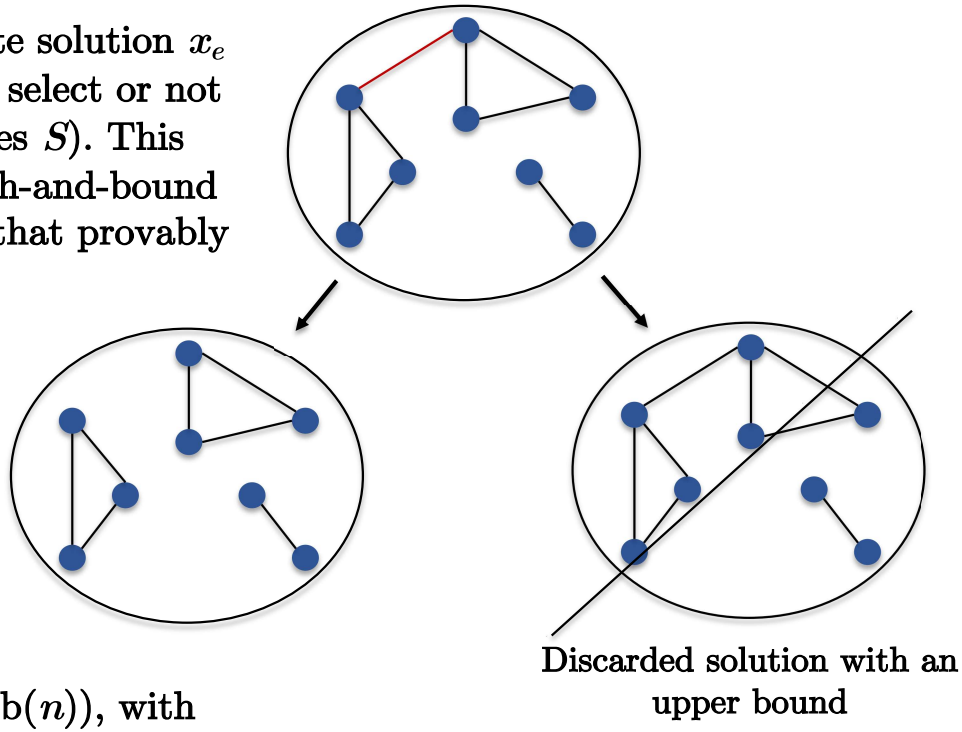
- LP problems can be solved in $O(n^{2.5})$ but all possible sets S makes the problem intractable. Candidates S that violate the sub-group constraint must be identified and added to the LP problem to get a valid tour.
- This leads to the Cutting Planes technique[1], which iteratively solves LP problems by adding linear inequality constraints.



[1] Dantzig, Fulkerson, Johnson, 1954

IP Solvers

- Solving a LP problem does not guarantee a discrete solution $x_e \in \{0,1\}$, and continuous values lead to a choice to select or not the edge in the tour (hence changing the candidates S). This leads to a tree of possible solutions, and the branch-and-bound technique[1] discards branches of the search space that provably do not contain an optimal solution.

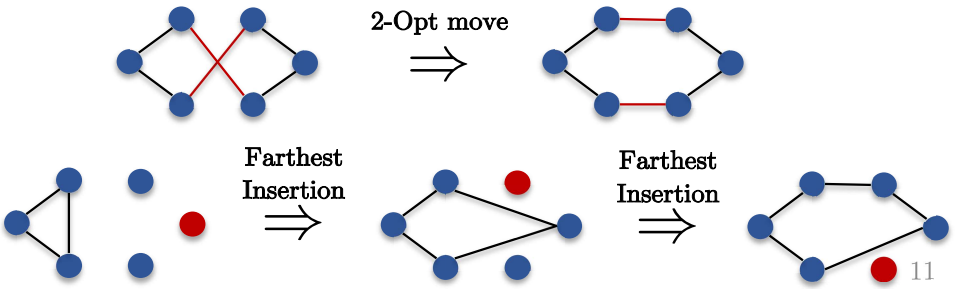


- Overall complexity for Concorde/Gurobi is $O(n^{2.5}\, b(n))$, with $O(n^{2.5})$ for LP and $O(b(n))$ for the number of branches to explore.

[2] Bellman, Held, Karp, 1962

Heuristic Solvers

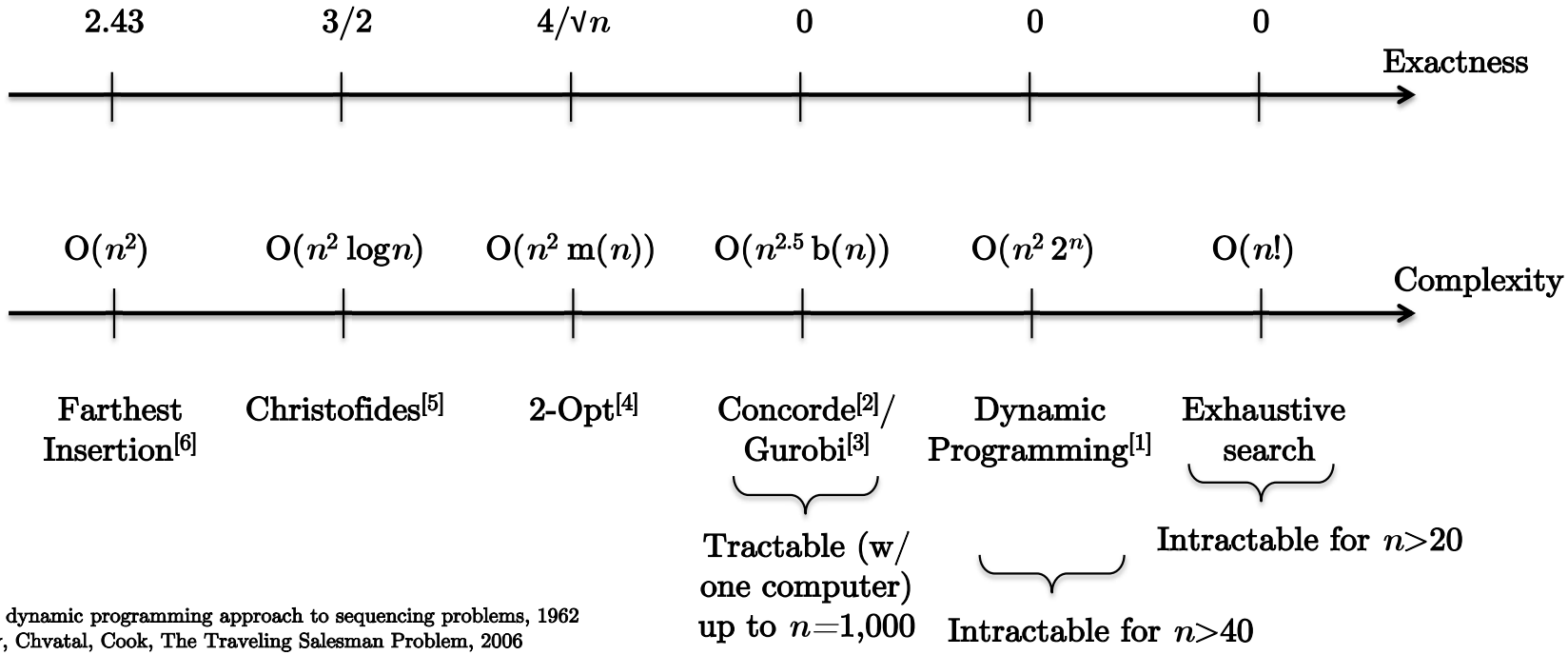
- Approximate/heuristic algorithms :
  - Christofides algorithm[1] : Approximation based on minimum spanning tree. Polynomial-time algorithm with $O(n^2 \log n)$ that is guaranteed to find a solution within a factor 3/2 of the optimal solution.
  - 2-Opt algorithm[2] : Heuristic is based on a move that replaces two edges to reduce the tour length. Complexity is $O(n^2\, m(n))$, where n^2 is the number of node pairs and $m(n)$ is the number of times all pairs must be tested to reach a local minimum with worst-case being $O(2^{n/2})$. The approximation ratio is then $4/\sqrt{n}$.
  - Farthest/nearest/greedy insertion algorithm[3] : Complexity is $O(n^2)$ and the approximation ratio is 2.43 for farthest insertion (best insertion in practice).
  - Google OR-Tools[4] : Highly optimized program that solves TSP and a larger set of vehicle routing problems. This program applies different heuristics s.a. Simulated Annealing, Greedy Descent, Tabu Search, to navigate in the search space, and refines the solution by Local Search techniques.



[1] Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, 1976
[2] Johnson, McGeoch, The traveling salesman problem: A case study in local optimization, 1995
[3] Johnson, Local Optimization and the Traveling Salesman Problem, 1990
[4] Google OR-tools: Google's Operations Research tools, 2015

Traditional Solvers

- Hierarchy of traditional TSP algorithms :



[1] Held, Richard, A dynamic programming approach to sequencing problems, 1962
[2] Applegate, Bixby, Chvatal, Cook, The Traveling Salesman Problem, 2006
[3] Gu, Rothberg, Bixby, Gurobi, 2008
[4] Johnson, McGeoch, The traveling salesman problem: A case study in local optimization, 1995
[5] Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, 1976
[6] Johnson, Local Optimization and the Traveling Salesman Problem, 1990

Outline

- History of TSP
- Traditional Solvers
- **Neural Network Solvers**
- Proposed Architecture
- Numerical Results
- Discussion
- Conclusion

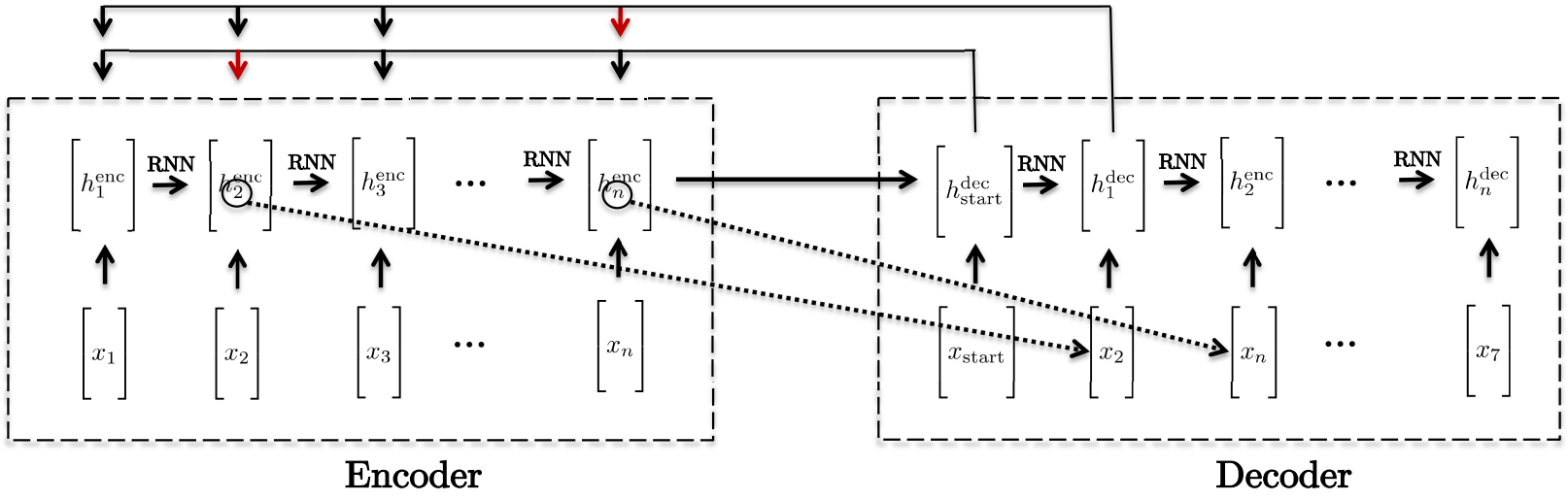
Deep Learning for the TSP Combinatorial Problem

- DL has significantly improved Computer Vision, Natural Language Processing and Speech Recognition in the last decade by replacing hand-crafted visual/text/speech features by features learned from data[1,2].
- For combinatorial problems, the main question is whether DL can learn better heuristics from data, i.e. replacing human-engineered heuristics?
  - This is appealing because developing algorithms to tackle efficiently NP-hard problems may require years of research (TSP has been actively studied for 60 years). Besides, many industry problems are combinatorial by nature.
- The last five years have seen the emergence of promising techniques where (graph) neural networks have been capable to learn new combinatorial algorithms with supervised or reinforcement learning.

[1] LeCun, Bottou, Bengio, Haffner, Gradient Based Learning Applied to Document Recognition, 1998
[2] LeCun, Bengio, Hinton, Deep learning, 2015

Neural Network Solvers

- Hopfield Nets[1] : First NN to solve (small) TSPs.
- PointerNets[2] : Pioneer paper using modern DL to tackle TSP and combinatorial optimization problems. Combine recurrent networks to encode the cities and decode the node sequence of the tour (auto-regressive decoding), and attention mechanism[3] (similar idea than[4] that was applied to NLP with great success). Supervised learning with approximate TSP solutions.



[1] Hopfield, Tank, Neural computation of decisions in optimization problems, 1985
[2] Vinyals, Fortunato, Jaitly, Pointer networks, 2015
[3] Bahdanau, Cho, Bengio, Neural machine translation by jointly learning to align and translate, 2014

Neural Network Solvers

- PointerNets+RL[1] : Improve[2] with Reinforcement Learning (no TSP solutions required). Two RL approaches : A standard unbiased reinforce algorithm[3] and an active search algorithm that can explore more candidates. The tour length is used as the reward.
- Order-invariant PointerNets+RL[4] : Improve[2] which is not invariant by permutations of the order of the input cities (which is important for NLP but not for TSP). That requires[2] to randomly permute the input order to make the network learn this invariance.
- S2V-DQN[5] : A graph network that takes a graph and a partial tour as input, and outputs a state-valued function Q to estimate the next node in the tour. Training is done by RL and memory replay[6], which allows intermediate rewards that encourage farthest node insertion heuristic.
- Quadratic Assignment Problem[7] : TSP can be formulated as a QAP, which is NP-hard and hard to approximate. A graph network based on the powers of adjacency matrix of node distances is trained in supervised manner. The loss is the KL distance between the adjacency matrix of the ground truth cycle and its network prediction. A feasible tour is computed with beam-search.

[1] Bello, Pham, Le, Norouzi, Bengio, Neural combinatorial optimization with reinforcement learning, 2016
[2] Vinyals, Fortunato, Jaitly, Pointer networks, 2015
[3] Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, 1992
[4] Nazari, Oroojlooy, Takac, Snyder, Reinforcement Learning for Solving the Vehicle Routing Problem, 2018
[5] Khalil, Dai, Zhang, Dilkina, Song, Learning combinatorial optimization algorithms over graphs, 2017
[6] Mnih et al, Playing Atari with Deep Reinforcement Learning, 2013
[7] Nowak, Villar, Bandeira, Bruna, A Note on Learning Algorithms for Quadratic Assignment with Graph Neural Networks, 2017

Neural Network Solvers

- Permutation-invariant Pooling Network[1] : Solve a variant of TSP with multiple salesmen. The network is trained by supervised learning and outputs a fractional solution, which is transformed into a feasible integer solution by beam-search. Non-autoregressive approach.
- Tranformer-encoder+2-Opt heuristic[2] : Use standard transformer to encode the cities and decode sequentially with a query composed of the last three cities (the cities before are ignored). Train with Actor-Critic RL, and solution is refined with a standard 2-Opt heuristic.
- Tranformer-encoder+Attention-decoder[3] : Also use standard transformer to encode the cities and decode sequentially with a query composed of the first city, the last city in the partial tour and a global representation of all cities. Train with reinforce and a deterministic baseline.
- GraphConvNet[4] : Learn a deep graph network by supervision to predict the probabilities of an edge to be in the TSP tour. A feasible tour is generated by beam-search. Non-autoregressive approach.

[1] Kaempfer, Wolf, Learning the Multiple Traveling Salesmen Problem with Permutation Invariant Pooling Network, 2018
[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!, 2018
[4] Joshi, Laurent, Bresson, An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem, 2019

Neural Network Solvers

- 2-Opt Learning[1] : Design transformer-based network to learn to select nodes for the 2-Opt heuristics (original 2-Opt may require $O(2^{n/2})$ moves before stopping). Learn by RL and actor-critic.
- GNNs with Monte Carlo Tree Search[2] : Inspired by AlphaGo[3], augment graph network with MCTS to improve the search exploration of tours by evaluating multiple next node candidates in the tour (auto-regressive methods cannot go back to the selection of the nodes).

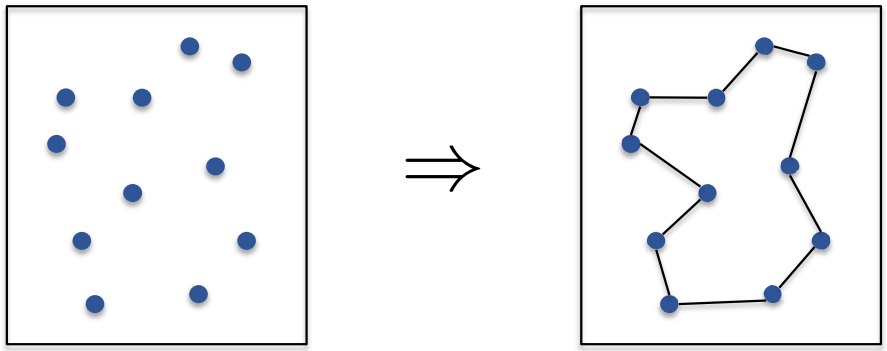
[1] Wu, Song, Cao, Zhang, Lim, Learning Improvement Heuristics for Solving Routing Problems, 2020
[2] Xing, Tu, A Graph Neural Network Assisted Monte Carlo Tree Search Approach to Traveling Salesman Problem, 2020
[3] D. Silver et al, Mastering the game of go with deep neural networks and tree search, 2016

Outline

- History of TSP
- Traditional Solvers
- Neural Network Solvers
- **Proposed Architecture**
- Numerical Results
- Discussion
- Conclusion

Our Approach

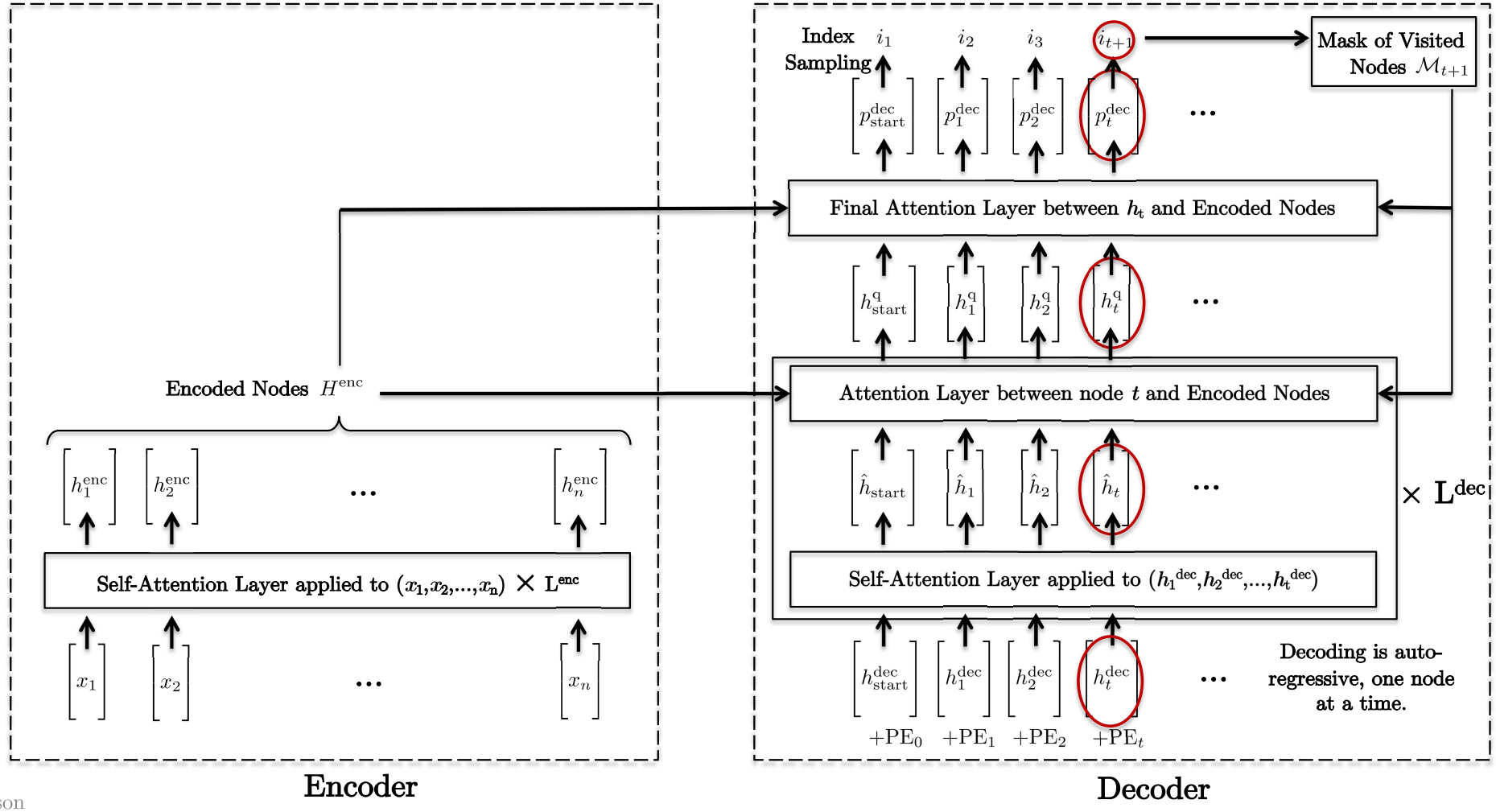
- We cast TSP as a "translation" problem :
  - Source is a set of 2D points.
  - Target is a tour (sequence of indices) with minimal length.
- Motivation : The translation problem has seen significant progress with Transformers[1].
- We train by reinforcement learning, with the same setting as[2] for comparison.
  - The reward is the tour length and the baseline is simply updated if the train network improves it on a set of TSPs.



Input/source Output/target

[1] Vaswani et al, Attention is all you need, 2017
[2] Kool, Van Hoof, Welling, Attention, learn to solve routing problems!, 2018

Proposed Architecture



Encoder

- **Standard Transformer encoder :**
  - **Multi-head attention, residual connection, batch normalization are used.**
  - **Memory/speed complexity is O(n^2).**

$$H^{\text{enc}} = H^{\ell=L^{\text{enc}}} \in \mathbb{R}^{n \times d}$$

where

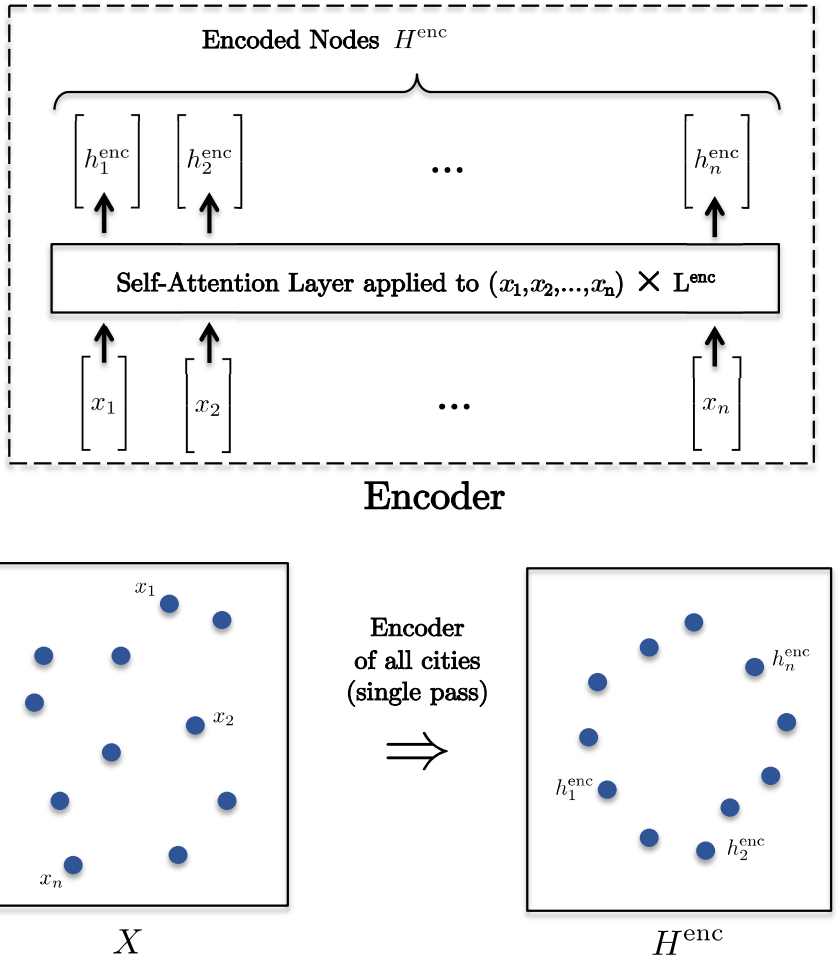
$$H^{\ell=0} = X \in \mathbb{R}^{n \times 2}$$

$$H^{\ell+1} = \text{softmax}(\frac{Q^\ell {K^\ell}^T}{\sqrt{d}})V^\ell \in \mathbb{R}^{n \times d},$$

$$Q^\ell = H^\ell W_Q^\ell \in \mathbb{R}^{n \times d}$$

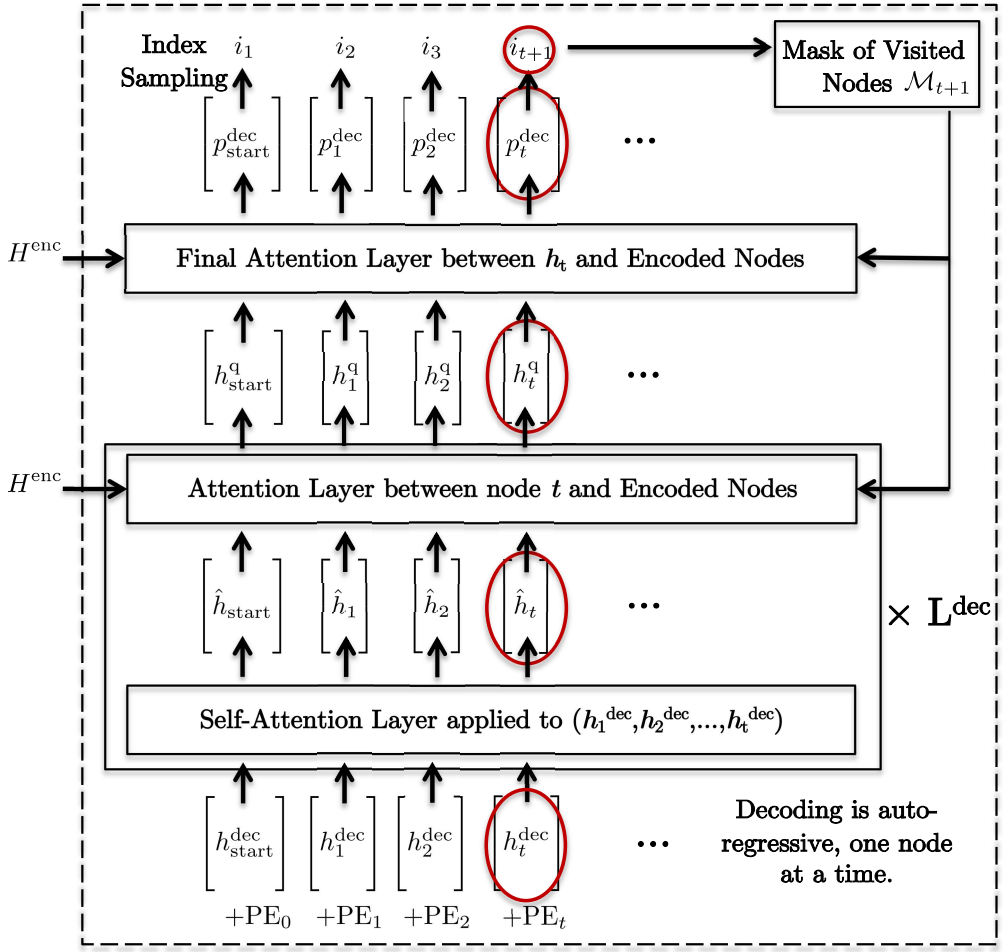
$$K^\ell = H^\ell W_K^\ell \in \mathbb{R}^{n \times d}$$

$$V^\ell = H^\ell W_V^\ell \in \mathbb{R}^{n \times d}$$



Decoder

- The decoding is auto-regressive, one city at a time.
- Suppose we have decoded the first t-1 cities in the tour, and we want to predict the next city.
- The decoding process has four steps :
  - Part 1 : Start with encoding of the i_t city + positional encoding.
  - Part 2 : Encode t^{th} city with the partial tour using self-attention.
  - Part 3 : Query the next city with the non-visited cities using multi-head attention.
  - Part 4 : Final query using a single-head attention + index sampling.



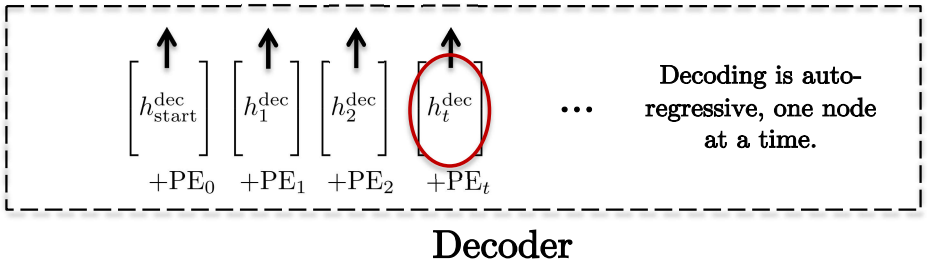
Decoder

Decoder – Part 1

- **Part 1 : Start decoding with the encoding of the previously selected i_t city :**

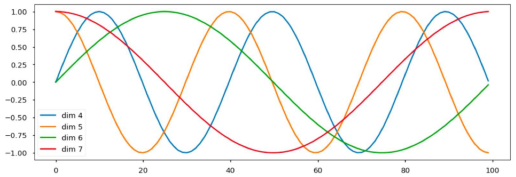
$$h_t^{\text{dec}} = h_{i_t}^{\text{enc}} + \text{PE}_t \in \mathbb{R}^d$$
$$h_{t=0}^{\text{dec}} = h_{\text{start}}^{\text{dec}} = z + \text{PE}_{t=0} \in \mathbb{R}^d$$

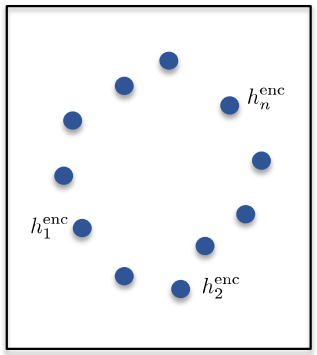
Learnable start vector, initialized at random.



- **Add positional encoding (PE) to order the nodes in the tour :**

$$\text{PE}_t \in \mathbb{R}^d$$
$$\text{PE}_{t,i} = \begin{cases} \sin(2\pi f_i t) \text{ if } i \text{ is even,} \\ \cos(2\pi f_i t) \text{ if } i \text{ is odd,} \end{cases}$$
$$\text{with } f_i = \frac{10,000^{\frac{d}{\lfloor 2i \rfloor}}}{2\pi}.$$

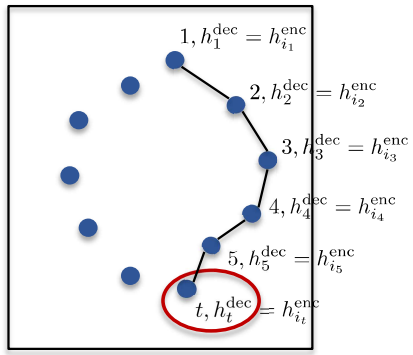




H^{enc}

Start with previously selected city + add PE

$\Rightarrow$



$h_t^{\text{dec}} + \text{PE}_t$

Decoder – Part 2

- Part 2 : Encode t^{th} city with the partial tour using self-attention.
  - Multi-head attention, residual connection, layer normalization are used.
  - Memory/speed complexity is $O(t)$ at each recursive step.

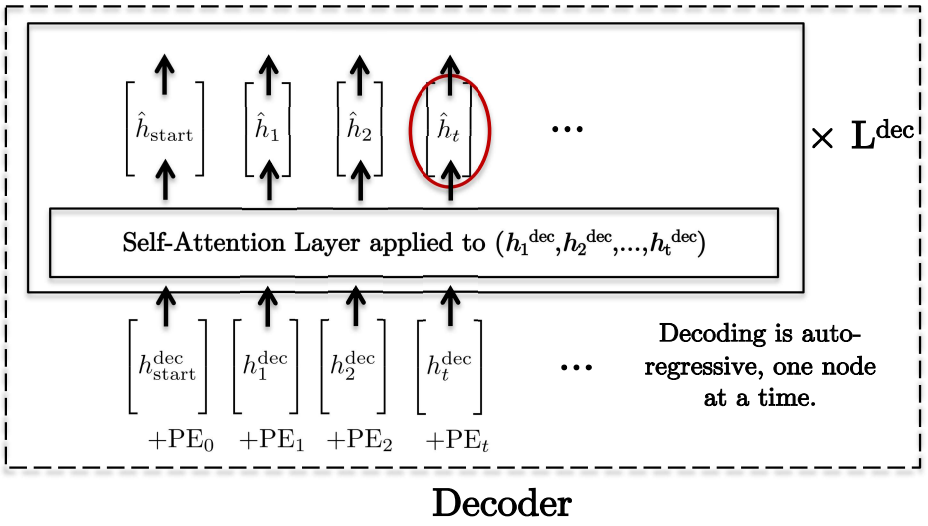
$$\hat{h}_t^{\ell+1} = \text{softmax}(\frac{q^\ell {K^\ell}^T}{\sqrt{d}})V^\ell \in \mathbb{R}^d, \ \ell = 0, ..., L^{\text{dec}} - 1$$

$$q^\ell = \hat{h}_t^\ell \hat{W}_q^\ell \in \mathbb{R}^d$$

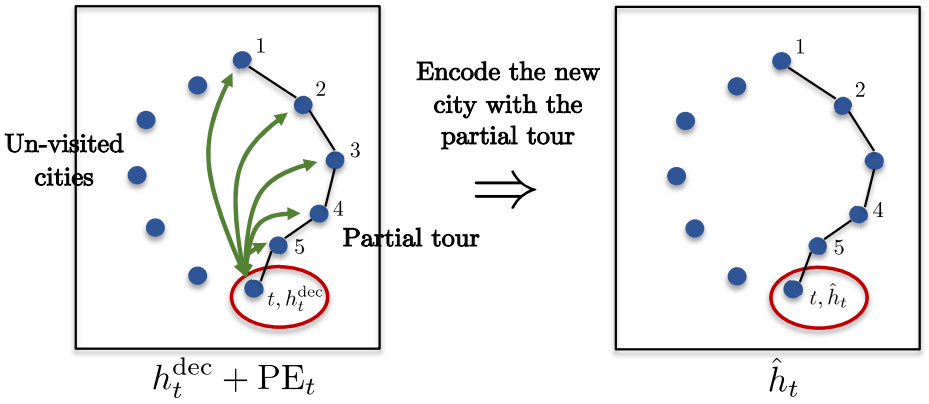
$$K^\ell = \hat{H}_{1,t}^\ell \hat{W}_K^\ell \in \mathbb{R}^{t \times d}$$

$$V^\ell = \hat{H}_{1,t}^\ell \hat{W}_V^\ell \in \mathbb{R}^{t \times d}$$

$$\hat{H}_{1,t}^\ell = [\hat{h}_1^\ell, .., \hat{h}_t^\ell], \ \hat{h}_t^\ell = \begin{cases} h_t^{\text{dec}} \text{ if } \ell = 0 \\ h_t^{\text{q},\ell} \text{ if } \ell > 0 \end{cases}$$



Decoder



$h_t^{\text{dec}} + \text{PE}_t$ $\hat{h}_t$

Decoder – Part 3

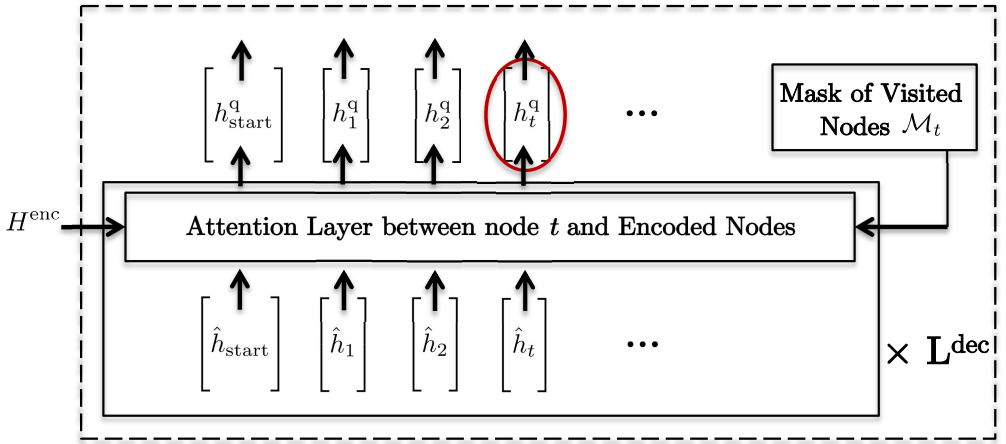
- Part 3 : Query the next city with the non-visited cities using attention.
  - Multi-head attention residual connection, layer normalization are used.
  - Memory/speed complexity is O(n) at each recursive step.

$$h_t^{\mathrm{q},\ell+1} = \mathrm{softmax}(\frac{q^\ell {K^\ell}^T}{\sqrt{d}} \odot \mathcal{M}_t) V^\ell \in \mathbb{R}^d, \ \ell = 0, ..., L^{\mathrm{dec}} - 1$$

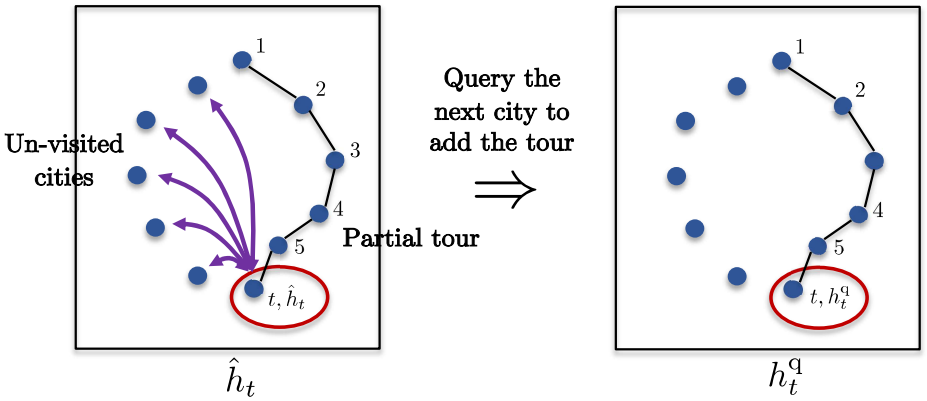
$$q^\ell = \hat{h}_t^{\ell+1} \tilde{W}_q^\ell \in \mathbb{R}^d$$

$$K^\ell = H^{\mathrm{enc}} \tilde{W}_K^\ell \in \mathbb{R}^{t \times d}$$

$$V^\ell = H^{\mathrm{enc}} \tilde{W}_V^\ell \in \mathbb{R}^{t \times d}$$



Decoder



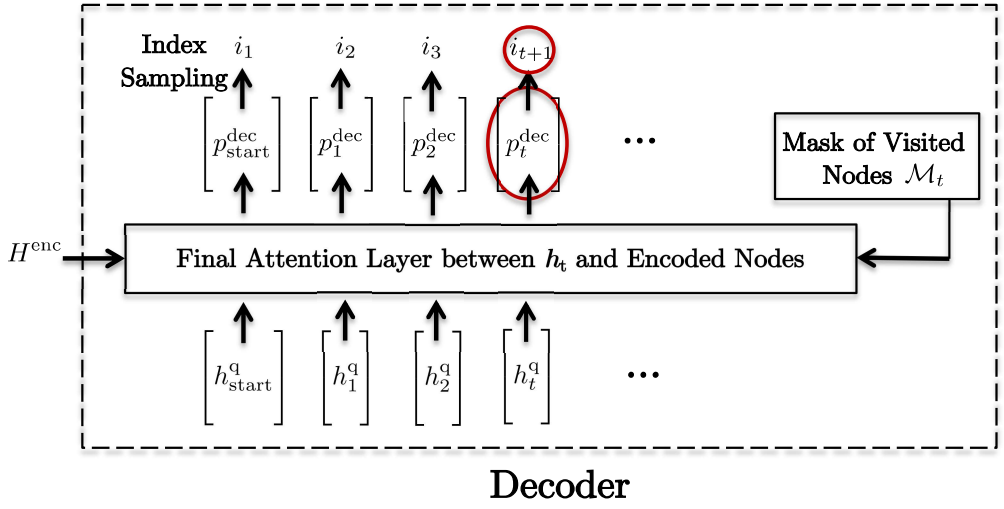
Decoder – Part 4

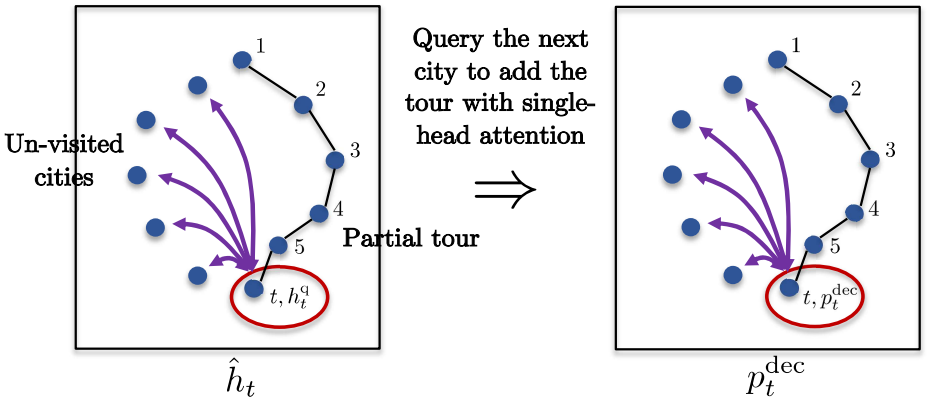
- Part 4 : Final query using attention + index sampling.
  - Single-head attention is used to get a distribution over the non-visited cities.
  - Finally, the next node i_{t+1} is sampled from the distribution using Bernoulli during training and greedy (index with max probability) at inference time.
  - Memory/speed complexity is O(n).

$$p_t^{\text{dec}} = \text{softmax}(C \tanh(\frac{qK^T}{\sqrt{d}} \odot \mathcal{M}_t)) \in \mathbb{R}^n,$$

with

$$q = h_t^{\text{q}} \bar{W}_q \in \mathbb{R}^d$$
$$K = H^{\text{enc}} \bar{W}_K \in \mathbb{R}^{n \times d}$$



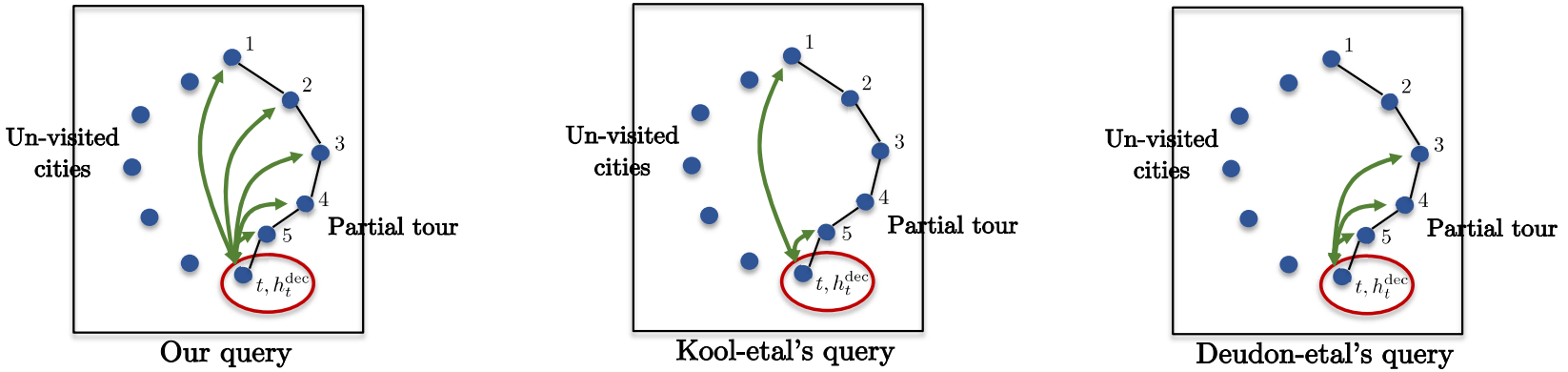


Architecture Comparison

- Transformers for NLP (translation) vs. TSP (combinatorial optimization) :
  - Order of input sequence is unknown for TSP.
    - Order of output sequence is coded with PEs for both TSP and NLP.
  - TSP-Encoder benefits from Batch Normalization (one-shot encoding of all cities).
    - TSP-Decoder uses Layer Normalization as with NLP (auto-regressive decoding).
  - TSP transformer is learned by Reinforcement Learning (no TSP solutions/approximations required).
  - Both transformers for NLP and TSP have quadratic complexity $\mathrm{O}(n^2\,\mathrm{L})$.

Architecture Comparison

- Models of Kool-etal[1] and Deudon-etal[2] :
  - We use the same transformer-encoder (with BN).
  - Our decoding architecture is different :
    - Our query uses all cities in the partial tour with a self-attention module.
    - Kool-etal use the first and current cities with a global representation of all cities as the query for the next city.
    - Deudon-etal define the query with the last three cities in the partial tour.



[1] Kool, Van Hoof, Welling, Attention, learn to solve routing problems!, 2018
[2] Deudon, Cournut, Lacoste, Adulyasak, Rousseau, Learning Heuristics for the TSP by Policy Gradient, 2018

Outline

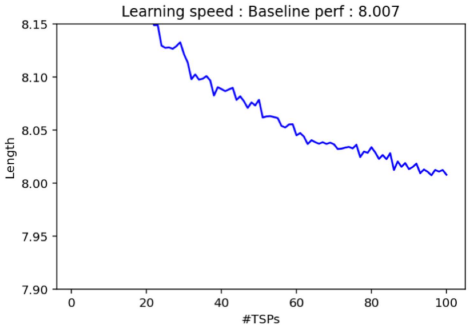
- History of TSP
- Traditional Solvers
- Neural Network Solvers
- Proposed Architecture
- **Numerical Results**
- Discussion
- Conclusion

Numerical Experiments

- For comparison, the same setting is used as in[1].

Method	Predicted Tour Length
Applegate, Bixby, Chvatal, Cook (Concorde)'06	7.764
Kool, Van Hoof, Welling'18	8.12
Our implementation of Kool, Van Hoof, Welling'18	8.092
Proposed technique	8.007

Table 1: TSP100

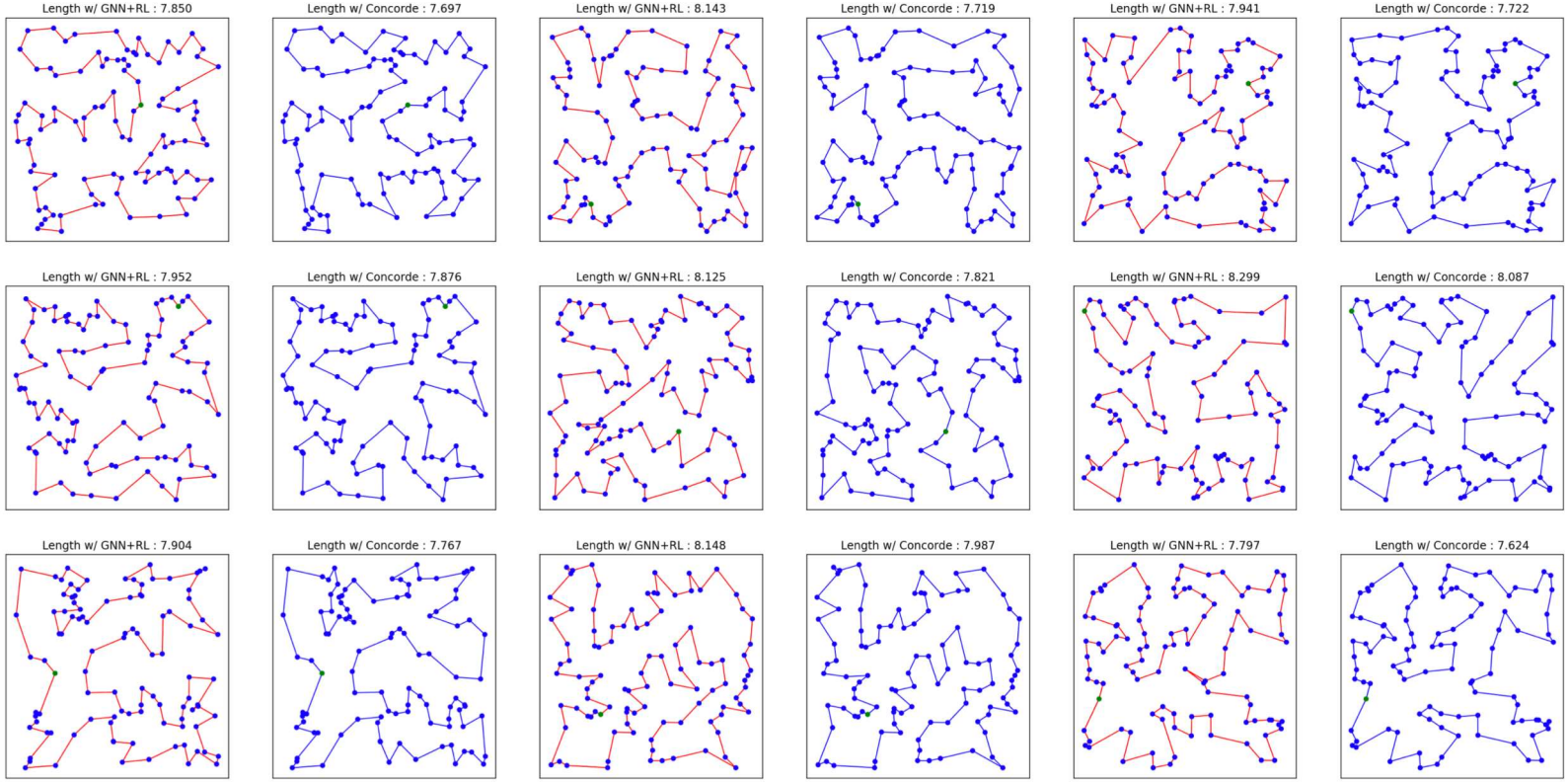


- Ablation study : Performance degrades if
  - PEs are removed from the decoder.
  - LN is used for the encoder and BN is used for the decoder.

[1] Kool, Van Hoof, Welling, Attention, learn to solve routing problems!, 2018

Numerical Experiments

- **Proposed technique vs. Concorde :**



Outline

- History of TSP
- Traditional Solvers
- Neural Network Solvers
- Proposed Architecture
- Numerical Results
- **Discussion**
- Conclusion

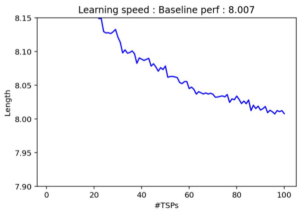
Future Work

- In this work, we essentially focused on the architecture.
- Further developments :
  - Sampling techniques such as beam-search are known to improve the results[1,2,3,4].
  - Use of human-made heuristics like 2-Opt to get intermediate rewards has also shown improvements[5] (the tour length as reward requires to wait the end of the tour construction before backpropagation).
  - Look-ahead strategies like MCTS has shown great promise[6].

[1] Nowak, Villar, Bandeira, Bruna, A Note on Learning Algorithms for Quadratic Assignment with Graph Neural Networks, 2017
[2] Kaempfer, Wolf, Learning the Multiple Traveling Salesmen Problem with Permutation Invariant Pooling Network, 2018
[3] Kool, Van Hoof, Welling, Attention, learn to solve routing problems!, 2018
[4] Joshi, Laurent, Bresson, An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem, 2019
[5] Wu, Song, Cao, Zhang, Lim, Learning Improvement Heuristics for Solving Routing Problems, 2020
[6] Xing, Tu, A Graph Neural Network Assisted Monte Carlo Tree Search Approach to Traveling Salesman Problem, 2020

Discussion

- Curb your enthusiasm !
  - Traditional solvers like Concorde/Gurobi (LP+CP+BB) outperform learning solvers in terms of (1) performance and (2) generalization.
  - But neural network solvers have faster inference time, $O(n^2\,L)$ vs. $O(n^{2.5}\,b(n))$.



- What's next ?
  - The Bitter Lesson, R. Sutton, 2019 : Learn longer as we can generate an infinite number of training data in CO.
  - Can we improve further the architecture? The learning paradigm?
  - Scaling to larger TSP sizes, n>1000, n>1M cities?
    - GPU memory is limited with $O(n^2)$ (Transformer architectures and auto-regressive decoding are in $O(n^2)$).
  - Consider "harder" TSP problems where traditional solvers like Gurobi can only provide weaker solutions or would take very long to solve.

Discussion

- What's next ?
  - Consider "harder" combinatorial problems where traditional solvers s.a. Gurobi cannot be used :

$$\min_{x \in \{0,1\}^n} c^T x \ \text{ s.t. } \ Ax \leq b, \ x \geq 0 \quad \Rightarrow \quad \min_{x \in \{0,1\}^n} f(x) \ \text{ s.t. } \ g(x) \leq 0, \ h(x) = 0$$

Linear/convex objective — Convex polytope — Non-convex objective — Non-linear constraints

  - Leverage learning techniques to improve traditional solvers :
    - Traditional solvers leverage Branch-and-Bound technique[1]. Selecting the variables to branch is critical for search efficiency, and relies on human-engineered heuristics s.a. Strong Branching[2] which is a high-quality but expensive branching rule. Recent works[3,4] have shown that neural networks can be successfully used to imitate expert heuristics and speed-up the BB computational time.
    - Future work may focus on going beyond imitation of human-based heuristics, and learning novel heuristics for faster Branch-and-Bound technique.

[1] Bellman, Held, Karp, 1962
[2] Achterberg, Koch, Martin, Branching rules revisited, 2005
[3] Gasse, Chetelat, Ferroni, Charlin, Lodi, Exact Combinatorial Optimization with Graph Convolutional Neural Networks, 2019
[4] Nair et al, Solving Mixed Integer Programs Using Neural Networks, 2020

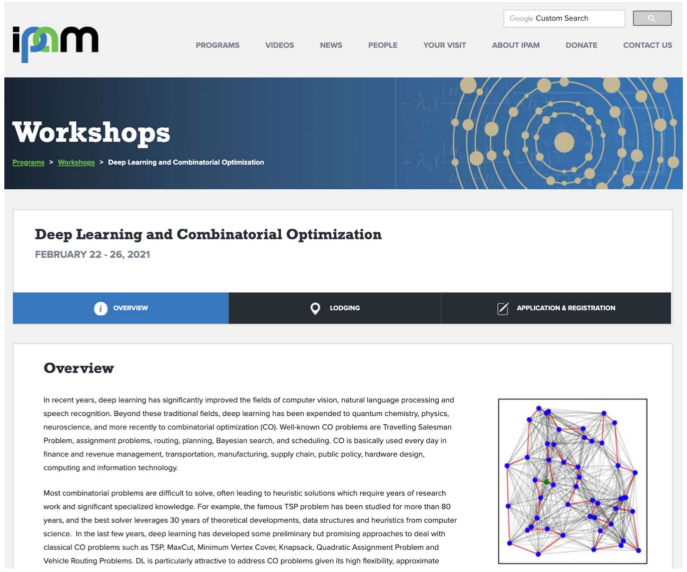
Outline

- History of TSP
- Traditional Solvers
- Neural Network Solvers
- Proposed Architecture
- Numerical Results
- Discussion
- **Conclusion**

Conclusion

- Combinatorial optimization is pushing the limit of deep learning.
  - Traditional solvers still provide better solutions than learning models.
  - Traditional solvers have been studied since the 1950s and the interest of applying deep learning to combinatorial optimization has just started.
  - This topic of research will naturally expend in the coming years as combinatorial problems problems s.a. assignment, routing, planning, scheduling are used every day by companies.
  - Novel software will be developed that combine continuous, discrete optimization and learning techniques.

Workshop Announcement



https://www.ipam.ucla.edu/programs/workshops/deep-learning-and-combinatorial-optimization

ORGANIZING COMMITTEE

Peter Battaglia (DeepMind Technologies)
Xavier Bresson (Nanyang Technological University, Singapore)
Stefanie Jegelka (Massachusetts Institute of Technology)
Yann LeCun (New York University, Canadian Institute for Advanced Research)
Andrea Lodi (École Polytechnique de Montréal)
Stanley Osher (University of California, Los Angeles (UCLA), Mathematics)
Oriol Vinyals (DeepMind Technologies)
Max Welling (University of Amsterdam)

Speaker List

Shipra Agrawal (Columbia University, Computer Science)
Sanjeev Arora (Princeton University)
Peter Battaglia (DeepMind Technologies)
Xavier Bresson (Nanyang Technological University, Singapore)
Joan Bruna (New York University)
Laurent Charlin (HEC Montréal)
Kyle Cranmer (New York University)
Sanjeeb Dash (IBM Watson Research Center)
Santanu Dey (Georgia Institute of Technology, School of Industrial and Systems Engineering)
Bistra Dilkina (University of Southern California (USC))
Tina Eliassi-Rad (Northeastern University, Computer Science & Network Science)
Emma Frejinger (University of Montreal)
Maxime Gasse (École Polytechnique de Montréal)
Stefano Gualandi (Università di Pavia)
Oktay Gunluk (Cornell University)
Joey Huchette (Rice University)
Stefanie Jegelka (Massachusetts Institute of Technology)
Ron Kimmel (Technion - Israel Institute of Technology, Intel Perceptual Computing)
Zico Kolter (Carnegie Mellon University)
Vladlen Koltun (Intel Corporation)
Wouter Kool (University of Amsterdam)
Andrea Lodi (École Polytechnique de Montréal)
Azalia Mirhoseini (Google Inc.)
Stanley Osher (University of California, Los Angeles (UCLA), Mathematics)
Sebastian Pokutta (Konrad-Zuse-Zentrum für Informationstechnik (ZIB), Department of Mathematics)
Louis-Martin Rousseau (École Polytechnique de Montréal)
Thiago Serra (Bucknell University)
Le Song (Georgia Institute of Technology)
Petar Velickovic (DeepMind Technologies)
Oriol Vinyals (DeepMind Technologies)
Ellen Vitercik (Carnegie Mellon University)

Thank you

Xavier Bresson

xbresson@ntu.edu.sg

http://www.ntu.edu.sg/home/xbresson

https://github.com/xbresson

https://twitter.com/xbresson

https://www.facebook.com/xavier.bresson.1

https://www.linkedin.com/in/xavier-bresson-738585b