The Transformer Network for the Traveling Salesman Problem

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)

IPAM-UCLA Workshop on Deep Learning and Combinatorial Optimization

February 22th 2021

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)

Outline

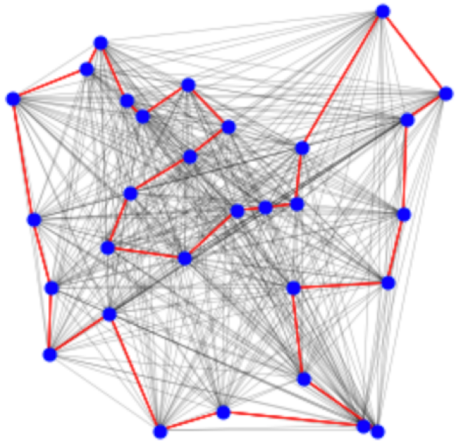
- History of TSP
- Traditional Solvers
- Neural Network Solvers
- Proposed Architecture
- Decoding Technique
- Numerical Results
- Discussion
- Conclusion

Outline

- History of TSP
- Traditional Solvers
- Neural Network Solvers
- Proposed Architecture
- Decoding Technique
- 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 problem, 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
- Decoding Technique
- Numerical Results
- Discussion
- Conclusion

Traditional TSP Solvers

- Two traditional approaches to tackle combinatorial problems :
  - Exact algorithms : Exhaustive search, Dynamic or Integer Programming. These algorithms are guaranteed to find optimal solutions, but they become intractable when n grows.
  - 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 :

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

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

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

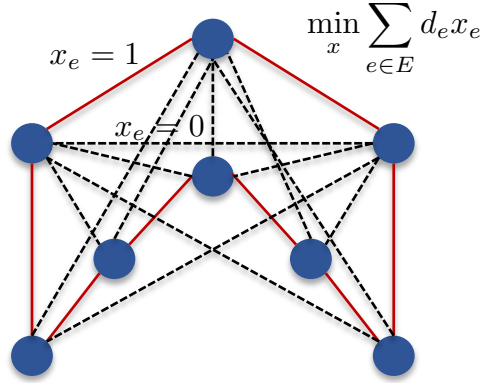
$$\text{where } \text{Cut}(A, A^c) = \{e_{vv'} \text{ s.t. } v \in A, v' \in A^c\}$$

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

[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 :**



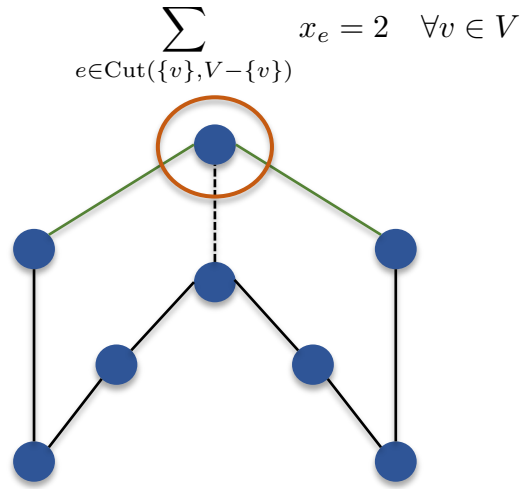
$$\min_x \sum_{e \in E} d_e x_e \quad \text{s.t.}$$

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

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

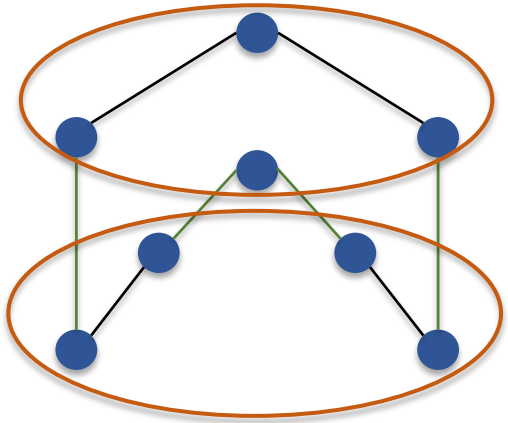
where $\text{Cut}(A, A^c) = \{e_{vv'} \text{ s.t. } v \in A, v' \in A^c\}$

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

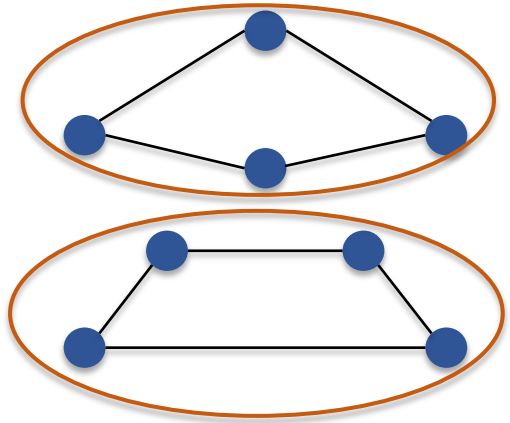


Sub-contour constraints that guarantee a feasible tour.

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



Satisfied

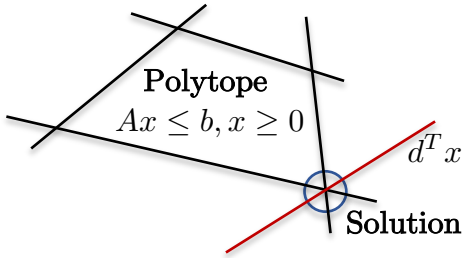


Not satisfied

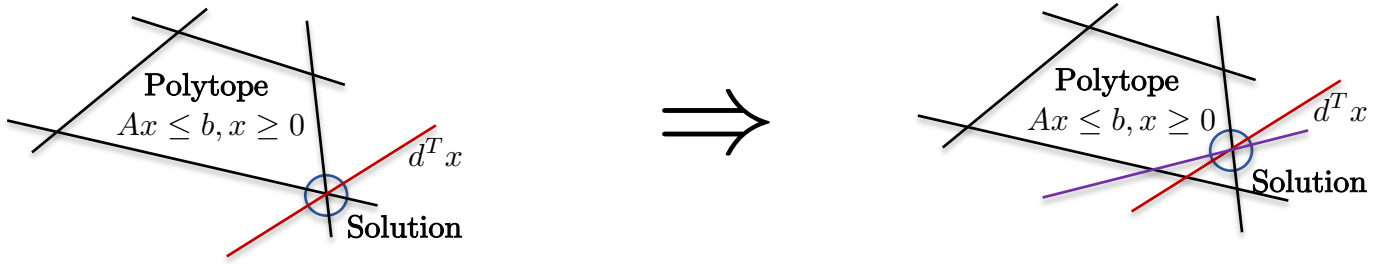
IP Solvers

- 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 :

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



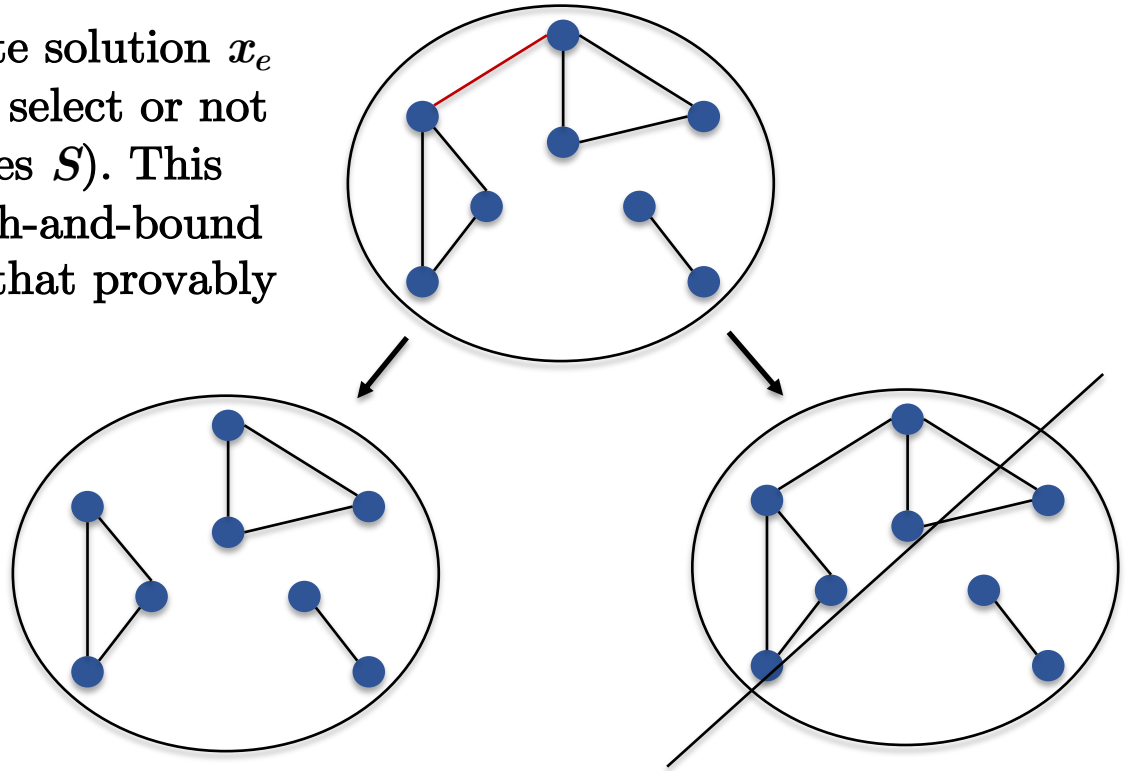
- LP problems can be solved in $O(n^{2.5})$ but all possible sets S makes the problem intractable. Candidates S that violate the sub-group constraint must be identified and added to the LP problem to get a valid tour.
- This leads to the Cutting Planes technique[1], which iteratively solves LP problems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



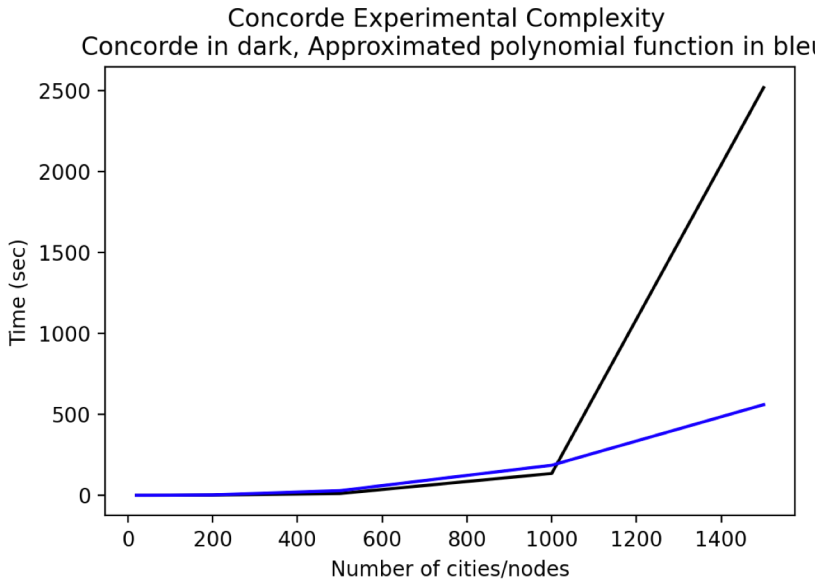
Discarded solution with a upper bound solution (for discrete x) and lower bound (for continuous x).

- 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.

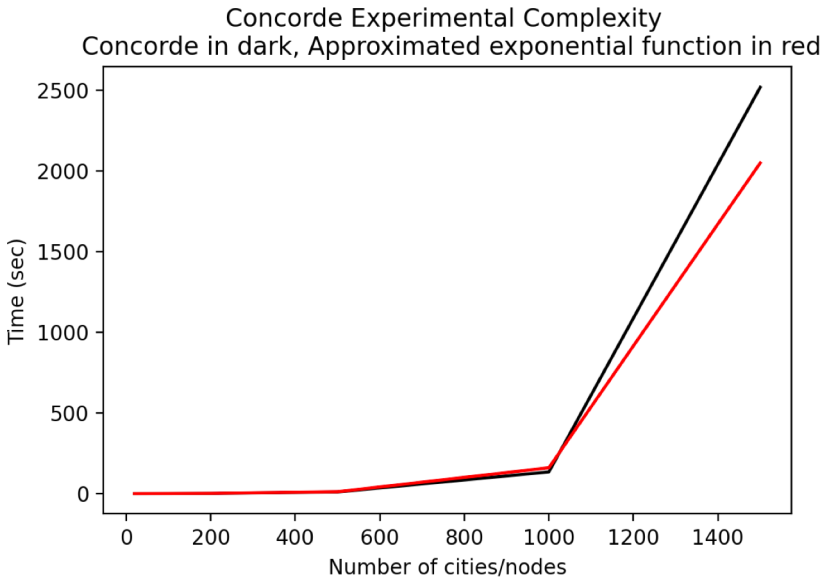
[2] Bellman, Held, Karp, 1962

Concorde[1] Complexity

- Experimental complexity :



Polynomial complexity
$O(n^{2.72})$

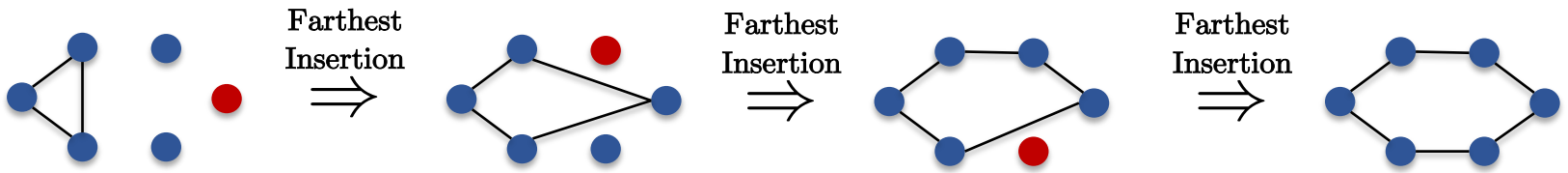


Exponential complexity
$O(e^{0.005n})$

[1] Applegate, Bixby, Chvatal, Cook, The Traveling Salesman Problem, 2006

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.
  - Farthest/nearest/greedy insertion algorithm[2] : Complexity is $O(n^2)$ and the approximation ratio is 2.43 for farthest insertion (best insertion in practice).
  - Google OR-Tools[3] : 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, Local Optimization and the Traveling Salesman Problem, 1990
[3] Google OR-tools: Google's Operations Research tools, 2015

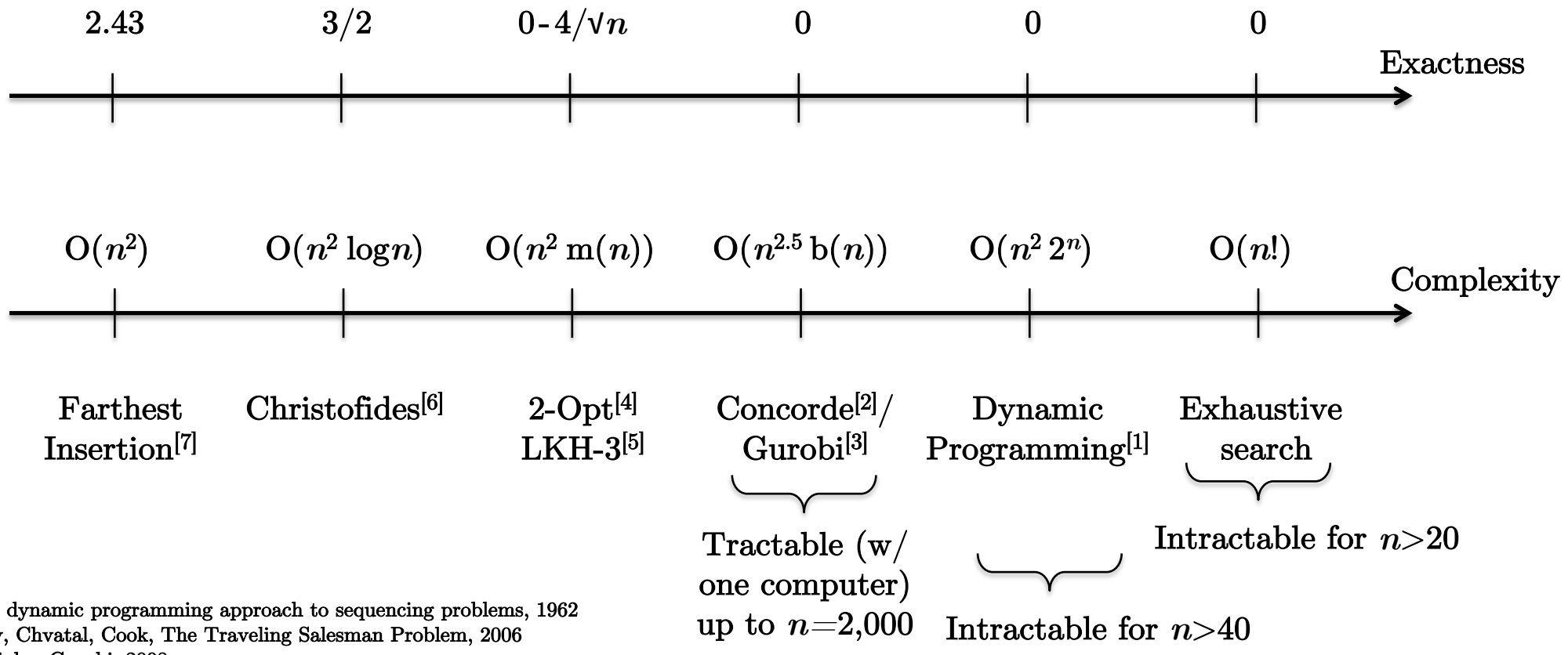
Heuristic Solvers

- Approximate/heuristic algorithms :
  - 2-Opt algorithm[1,2] : 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 $4/\sqrt{n}$.
  - Extension to 3-Opt move (replacing 3 edges) and more[3].
  - LKH-3 algorithm[4] : Extension of the original LKH[5] and LKH-2[6]. It is the best heuristic for solving TSP. It is an extension of 2-Opt/3-Opt where edge candidates are estimated with a Minimum Spanning Tree[6]. It tackles various TSP-type problems.

[1] Lin, Computer solutions of the traveling salesman problem, 1965
[2] Johnson, McGeoch, The traveling salesman problem: A case study in local optimization, 1995
[3] Blazinskas, Misevicius, Combining 2-Opt, 3-Opt and 4-Opt with K-SWAP-KICK perturbations for the traveling salesman problem, 2011
[4] Helsgaun, An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems, 2017
[5] Lin, Kernighan, An Effective Heuristic Algorithm for the Traveling-Salesman Problem, 1973
[6] Helsgaun, An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic, 2000

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] Helsgaun, An Extension of the Lin-Kernighan-Helsgaun TSP Solver for Constrained Traveling Salesman and Vehicle Routing Problems, 2017
[6] Christofides, Worst-case analysis of a new heuristic for the travelling salesman problem, 1976
[7] Johnson, Local Optimization and the Traveling Salesman Problem, 1990

Outline

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

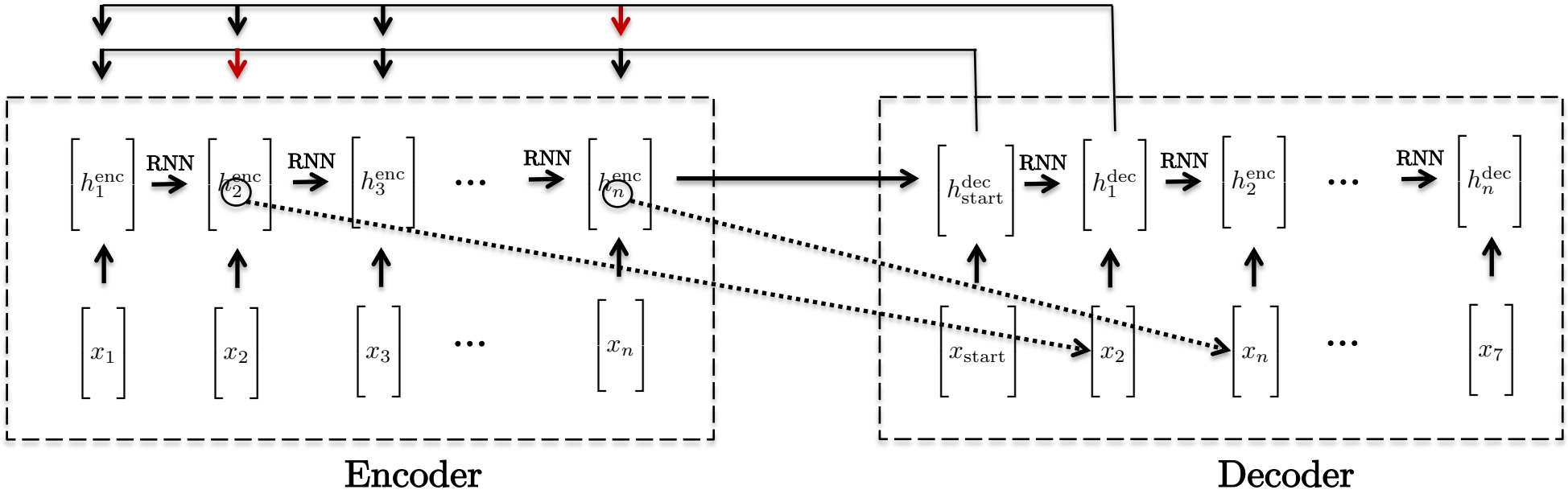
Deep Learning for the TSP Combinatorial Problem

- 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 70 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] Dai, Khalil, 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).

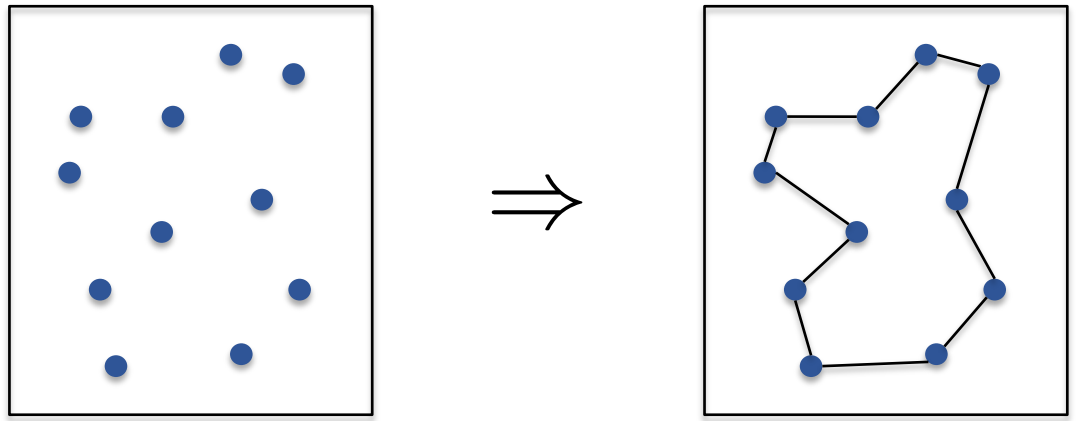
[1] Wu, Song, Cao, Zhang, Lim, Learning Improvement Heuristics for Solving Routing Problems, 2020
[2] Xing, Tu, A Graph Neural Network Assisted Monte Carlo Tree Search Approach to Traveling Salesman Problem, 2020
[3] D. Silver et al, Mastering the game of go with deep neural networks and tree search,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

Outline

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

Our Approach

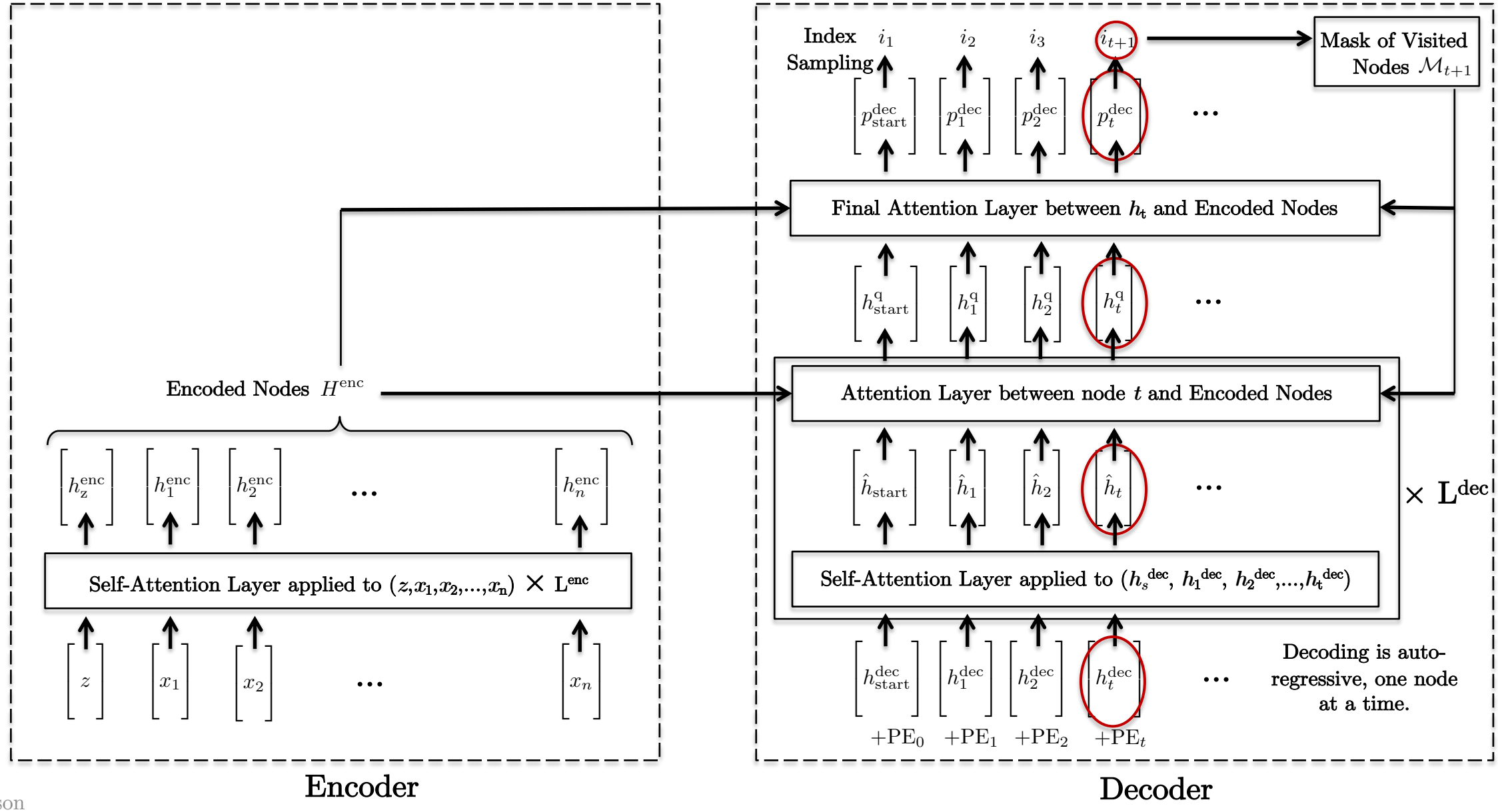
- We cast TSP as a "translation" problem :
  - Source is a set of 2D points.
  - Target is a tour (sequence of indices) with minimal length.
- Motivation : The translation problem has seen significant progress with Transformers[1].
- We train by reinforcement learning, with the same setting as[2] for 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 $O(n^2)$.

$$H^{\text{enc}} = H^{\ell=L^{\text{enc}}} \in \mathbb{R}^{(n+1)\times d}$$

where

$$H^{\ell=0} = \text{Concat}(z, X) \in \mathbb{R}^{(n+1)\times 2}, z \in \mathbb{R}^2, X \in \mathbb{R}^{n\times 2}$$

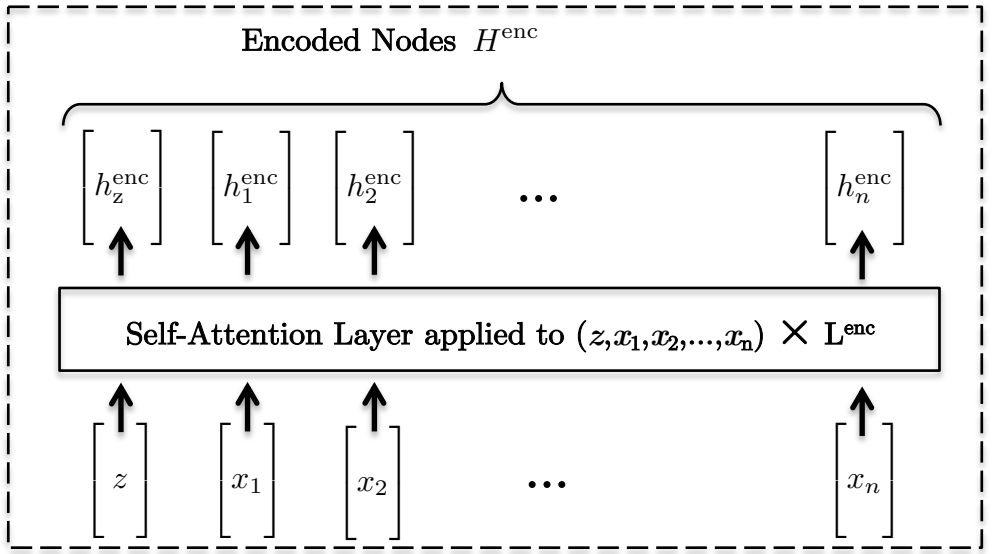
Start token, initialized at random.

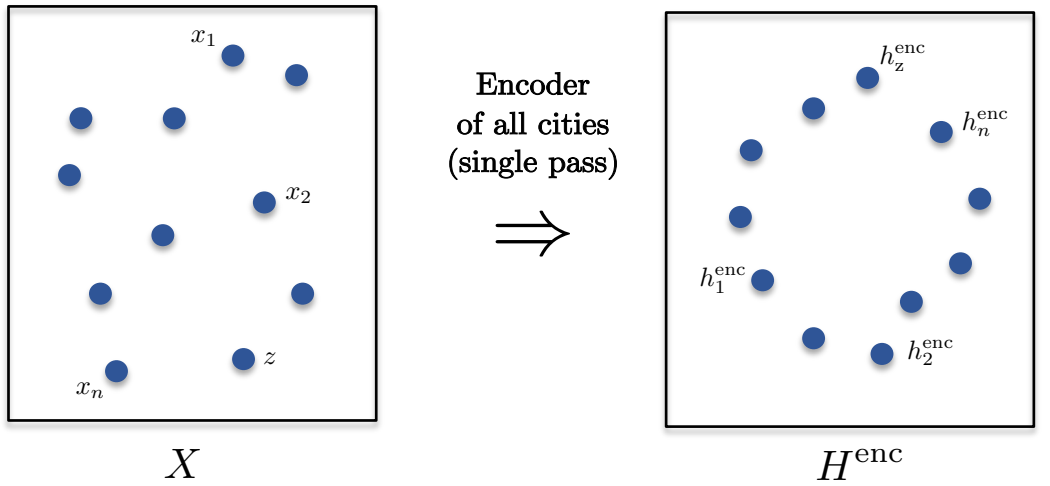
$$H^{\ell+1} = \text{softmax}(\frac{Q^\ell {K^\ell}^T}{\sqrt{d}})V^\ell \in \mathbb{R}^{(n+1)\times d},$$

$$Q^\ell = H^\ell W_Q^\ell \in \mathbb{R}^{(n+1)\times d}$$

$$K^\ell = H^\ell W_K^\ell \in \mathbb{R}^{(n+1)\times d}$$

$$V^\ell = H^\ell W_V^\ell \in \mathbb{R}^{(n+1)\times d}$$

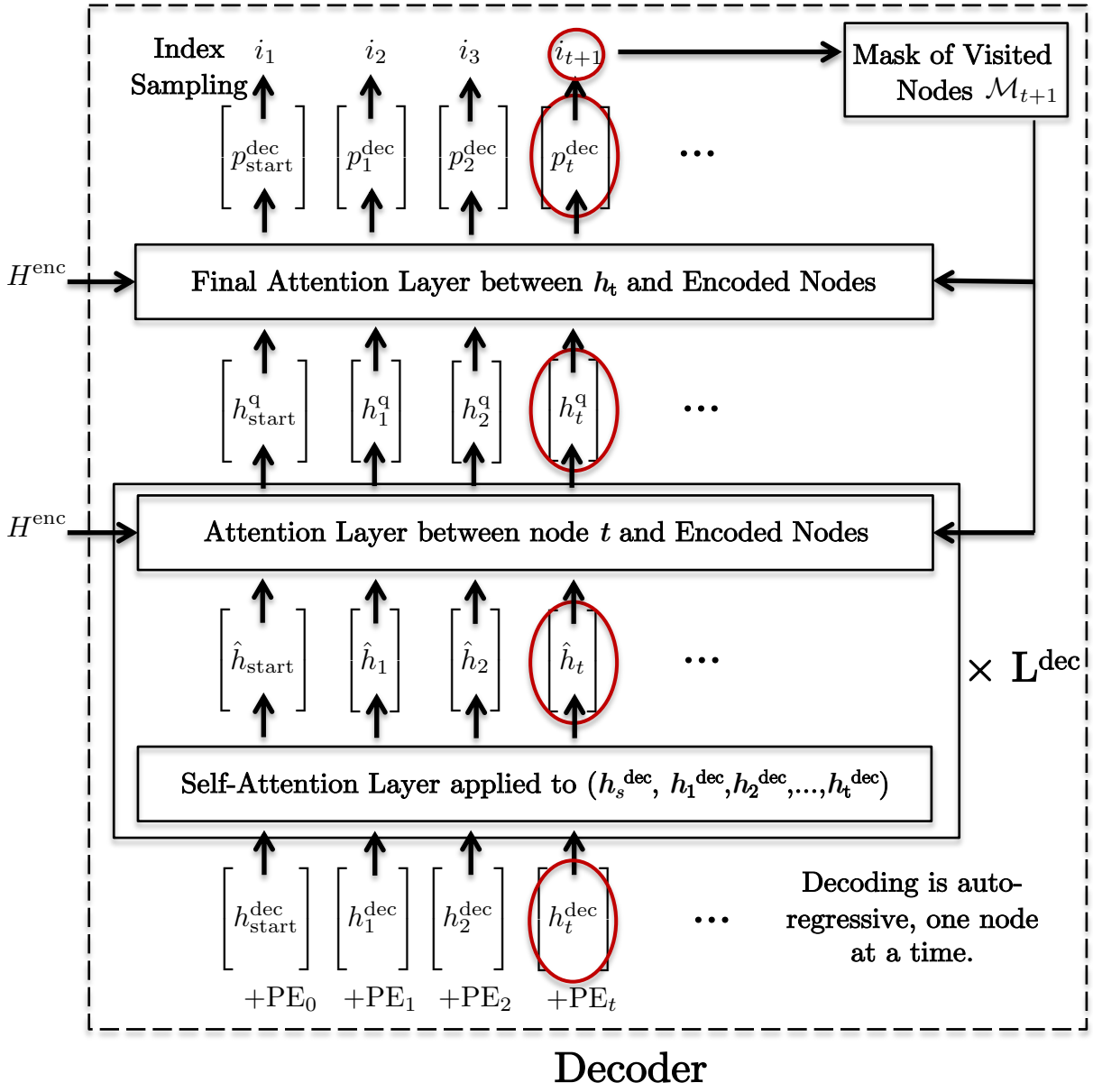




X H^{enc}

Decoder

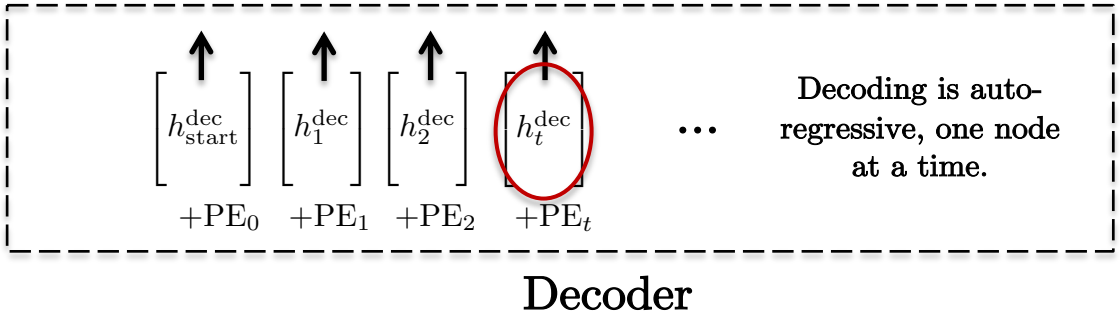
- The decoding is auto-regressive, one city at a time.
- Suppose we have decoded the first t cities in the tour, and we want to predict the next city.
- The decoding process has four steps :
  - Part 1 : Start with encoding of the i_t city + positional encoding.
  - Part 2 : Encode t^{th} city with the 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 – Part 1

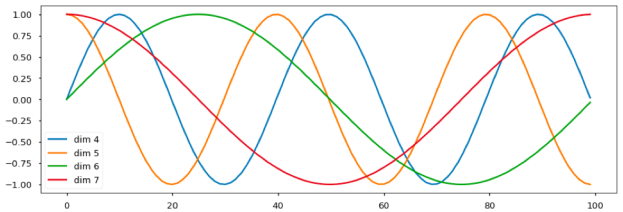
- **Part 1 : Start decoding with the encoding of the previously selected i_t city :**

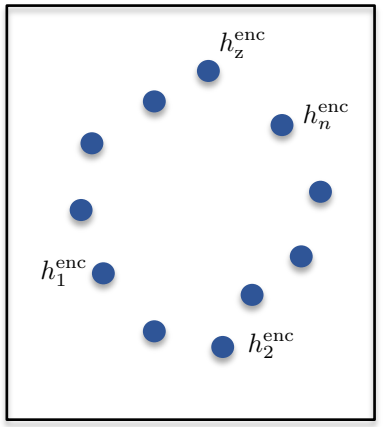
$$h_t^{\text{dec}} = h_{i_t}^{\text{enc}} + \text{PE}_t \in \mathbb{R}^d$$
$$h_{t=0}^{\text{dec}} = h_{\text{start}}^{\text{dec}} + \text{PE}_{t=0} \in \mathbb{R}^d, \text{ with } h_{\text{start}}^{\text{dec}} = h_{\text{z}}^{\text{enc}}$$



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

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

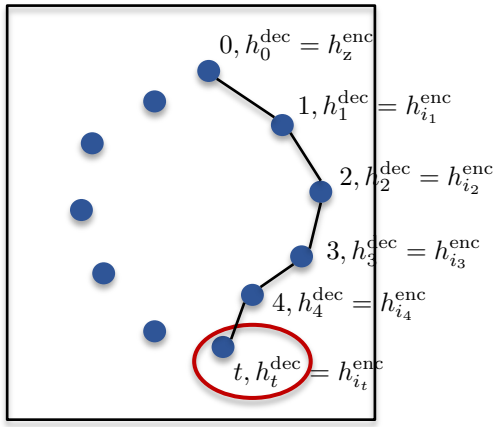




H^{enc}

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.

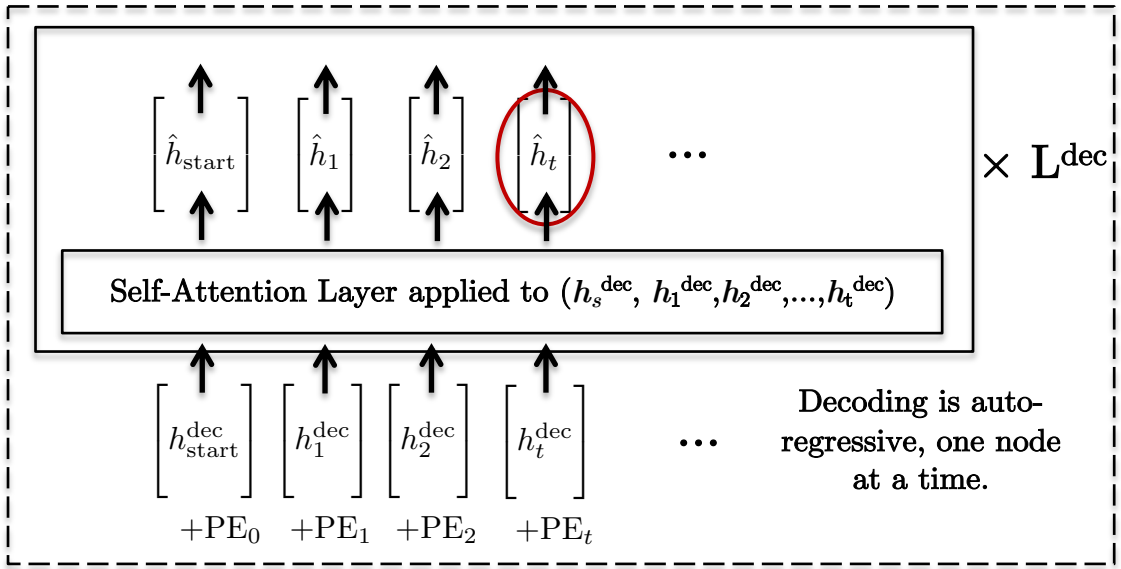
$$\hat{h}_t^{\ell+1} = \text{softmax}(\frac{q^\ell {K^\ell}^T}{\sqrt{d}})V^\ell \in \mathbb{R}^d, \ \ell = 0, ..., L^{\text{dec}} - 1$$

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

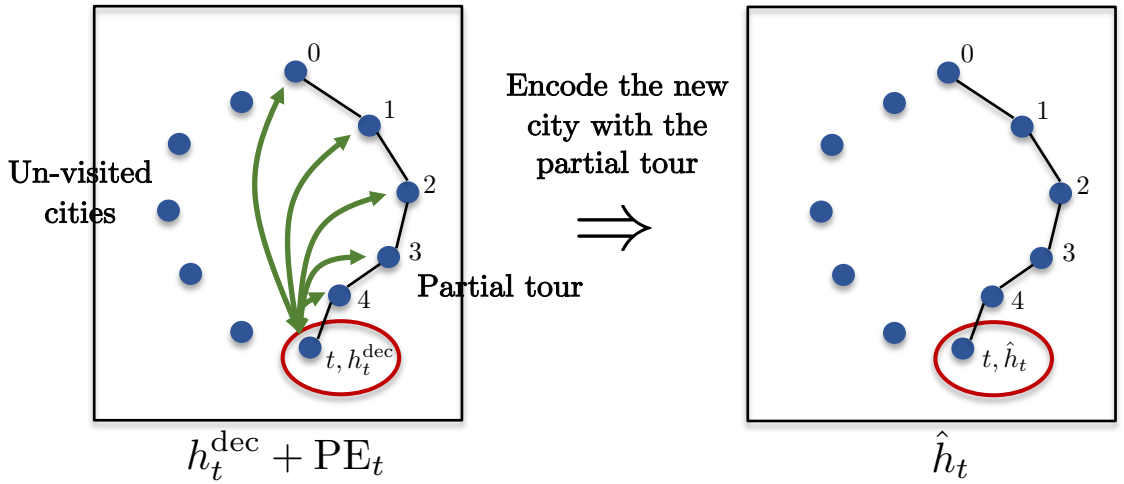
$$K^\ell = \hat{H}_{1,t}^\ell \hat{W}_K^\ell \in \mathbb{R}^{t \times d}$$

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

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



Decoder



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

Decoder – Part 3

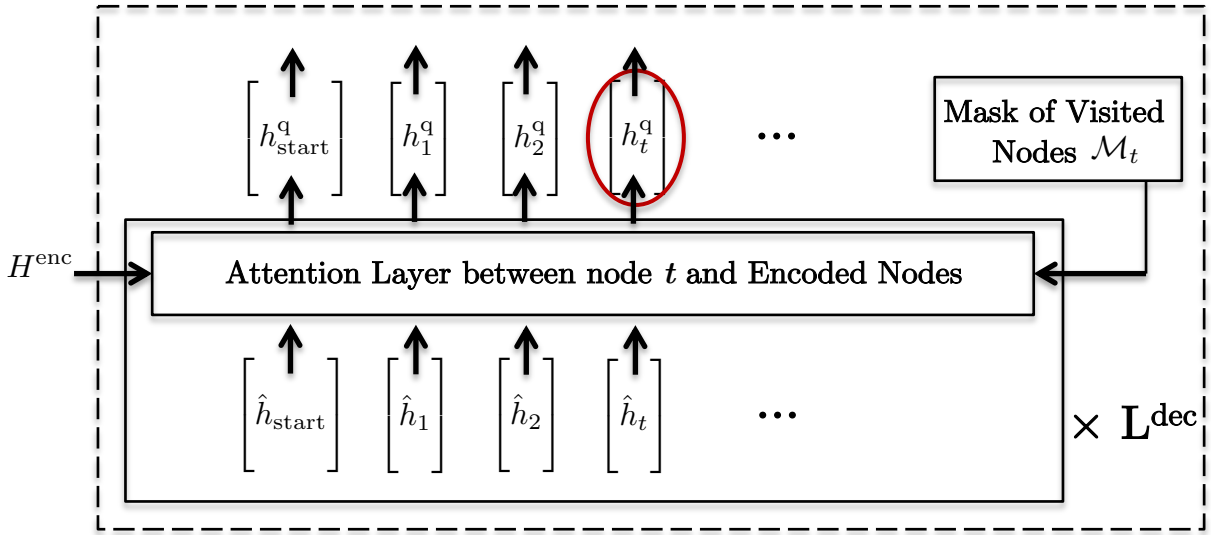
- Part 3 : Query the next city with the non-visited cities using attention.
  - Multi-head attention residual connection, layer normalization are used.
  - Memory/speed complexity is $O(n)$ at each recursive step.

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

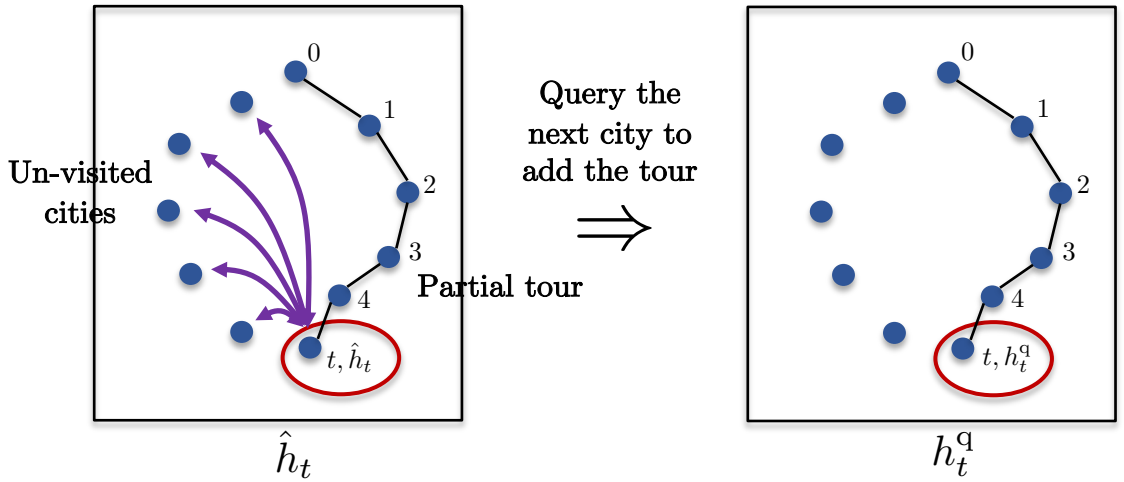
$$q^\ell = \hat{h}_t^{\ell+1} \tilde{W}_q^\ell \in \mathbb{R}^d$$

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

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



Decoder



Decoder – Part 4

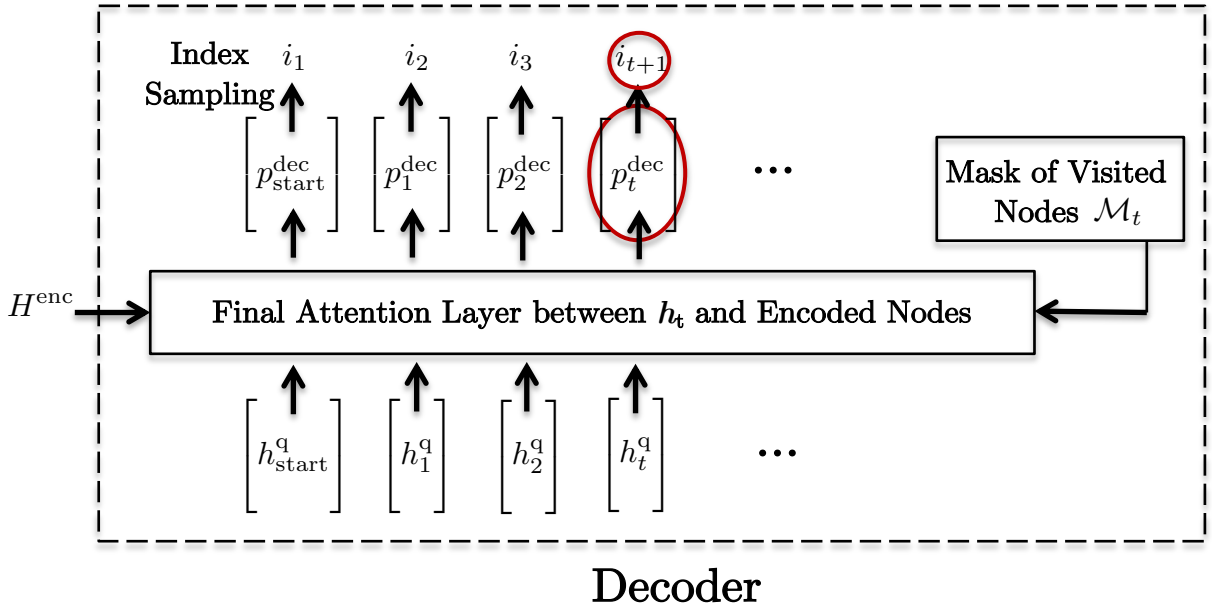
- Part 4 : Final query using attention + index sampling.
  - Single-head attention is used to get a distribution over the non-visited cities.
  - Finally, the next node i_{t+1} is sampled from the distribution using Bernoulli during training and greedy (index with max probability) at inference time.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

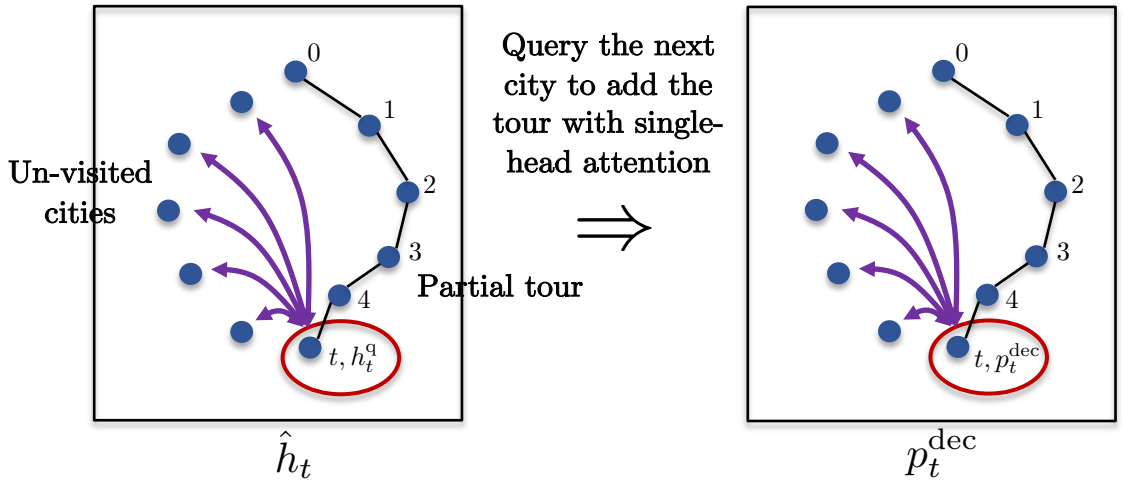
$$p_t^{\text{dec}} = \text{softmax}(C \tanh(\frac{qK^T}{\sqrt{d}} \odot \mathcal{M}_t)) \in \mathbb{R}^n,$$

with

$$q = h_t^{\text{q}} \bar{W}_q \in \mathbb{R}^d$$

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



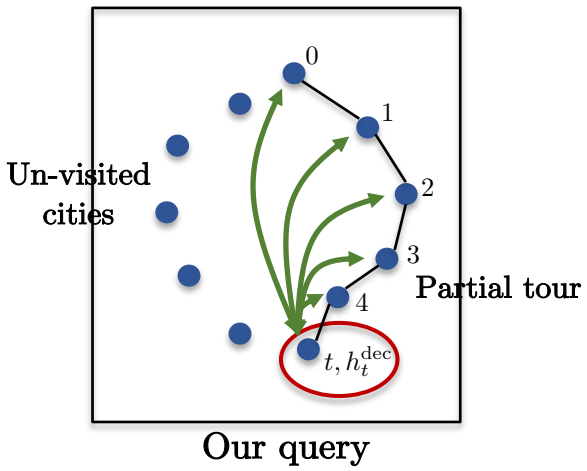


Architecture Comparison

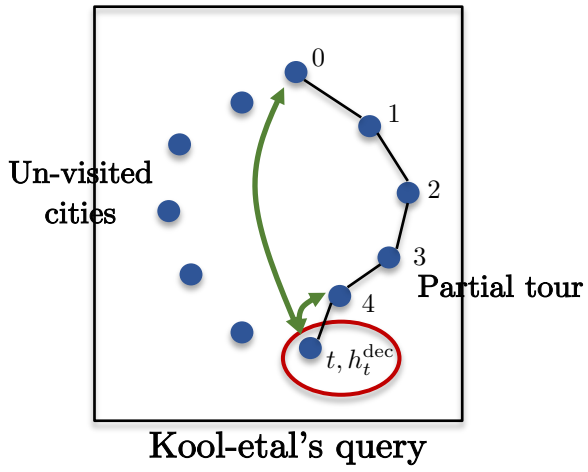
- Transformers for NLP (translation) vs. TSP (combinatorial optimization) :
  - Order of input sequence is not relevant for TSP.
    - Order of output sequence is coded with PEs for both TSP and NLP.
  - TSP-Encoder benefits from Batch Normalization (one-shot encoding of all cities).
    - TSP-Decoder 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

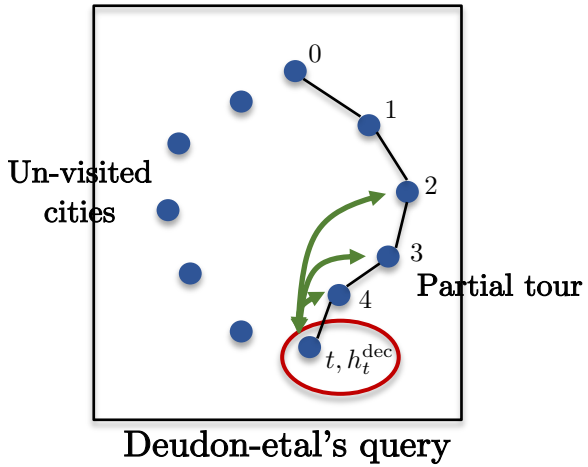
- Models of Kool-etal[1] and Deudon-etal[2] :
  - We use the same transformer-encoder (with BN).
  - Our decoding architecture is different :
    - Our query uses all cities in the partial tour with a self-attention module.
    - Kool-etal use the first and last cities with a global representation of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



Our query



Kool-etal's query

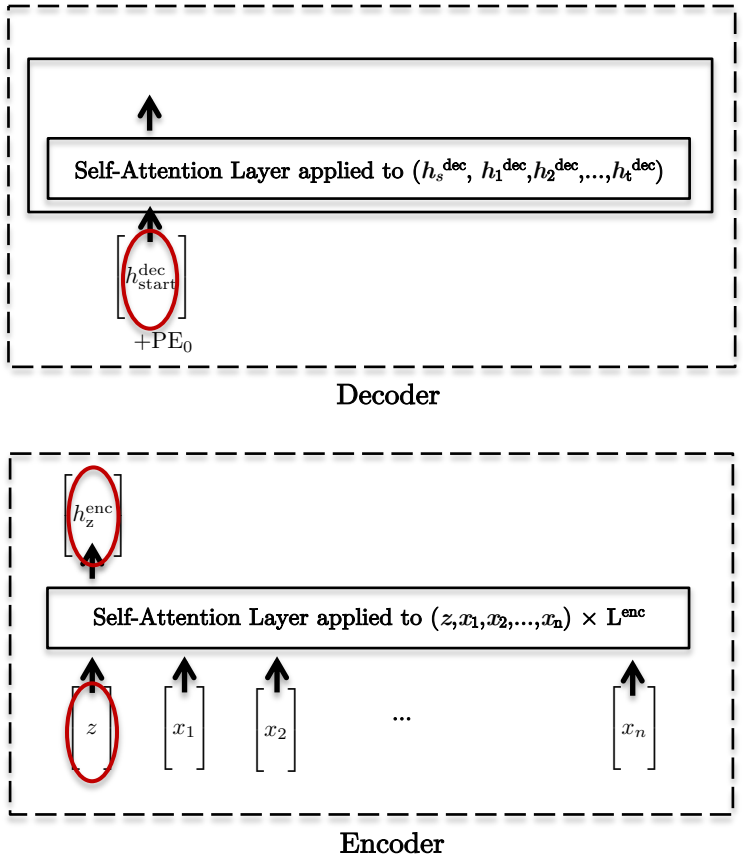


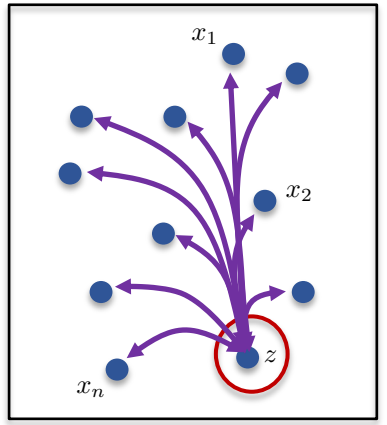
Deudon-etal's query

[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

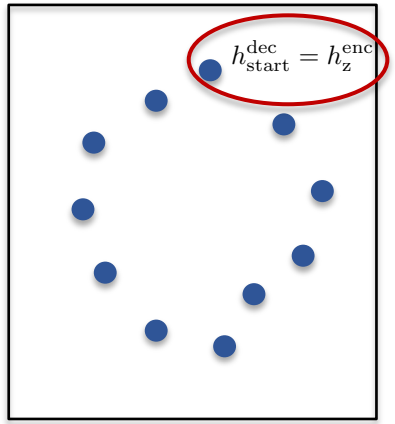
Architecture Comparison

- Our decoding process starts differently :
  - We use a token city z. This city does not exist and aims at starting the decoding at the best possible location by querying all cities with a self-attention module.



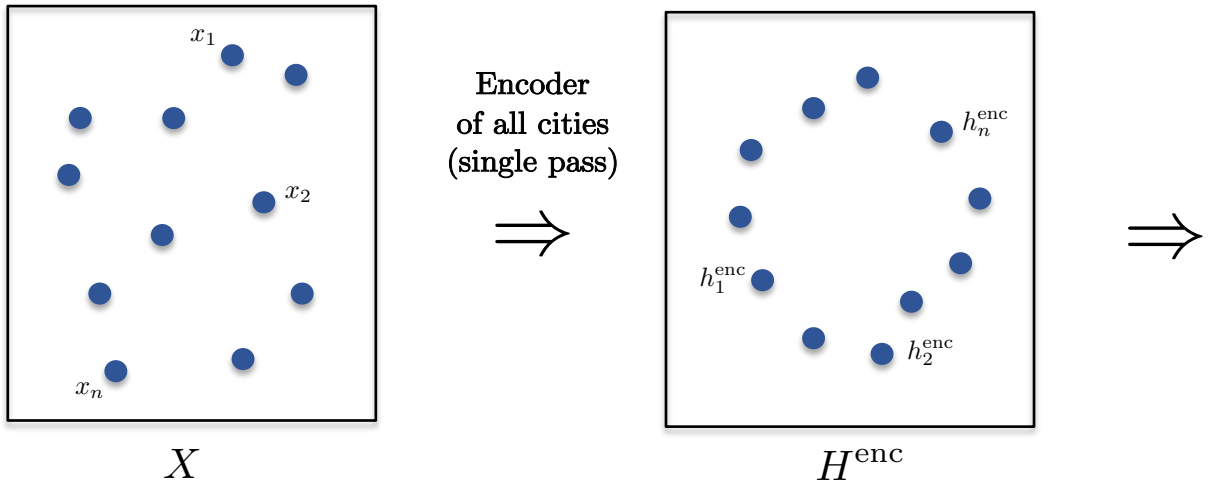


Encoding with all cities



Architecture Comparison

- Our decoding process starts differently :
  - Kool-etal[1] starts the decoding with the mean representation of the encoding cities and a random token of the first and current cities.
  - Deudon-etal[2] starts the decoding with the mean representation of the encoding cities and a random token of the last three cities.



Kool-etal's starting query :

$$h_{\text{start}}^{\text{dec}} = \text{Concat}(z_0, z_{t-1}, g),$$

$$\text{with } z_0, z_{t-1} \in \mathcal{N}_{0,1}^d, \; g = \frac{1}{n}\sum_{i=1}^{n} h_i^{\text{enc}} \in \mathbb{R}^d$$

Deudon-etal's starting query :

$$h_{\text{start}}^{\text{dec}} = \text{Concat}(z_{t-1}, z_{t-2}, z_{t-3}),$$

$$\text{with } z_{t-1}, z_{t-2}, z_{t-3} \in \mathcal{N}_{0,1}^d$$

[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

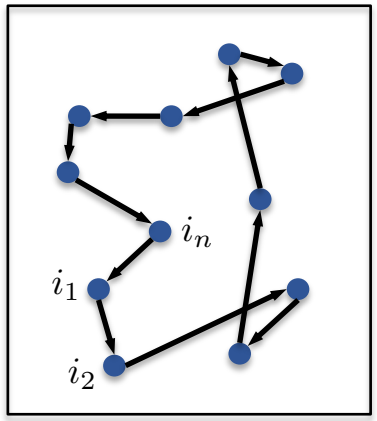
- History of TSP
- Traditional Solvers
- Neural Network Solvers
- Proposed Architecture
- **Decoding Technique**
- Numerical Results
- Discussion
- Conclusion

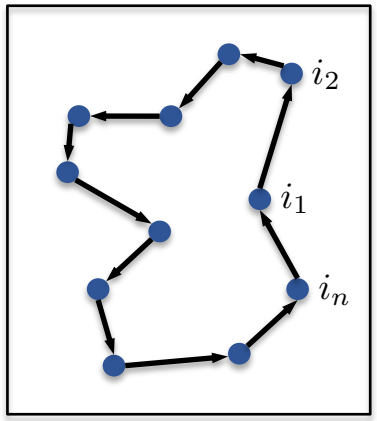
Tour Decoding

- A tour is represented as an ordered sequence of city indices : $\text{seq}_n = \{i_1, i_2, \ldots, i_n\}$
- TSP can be cast as a sequence optimization problem :

$$\max_{\text{seq}_n = \{i_1, \ldots, i_n\}} P^{\text{TSP}}(\text{seq}_n | x) = P^{\text{TSP}}(i_1, \ldots, i_n | x)$$

Probability of the sequence to be a solution of the TSP given a set x of n 2-D points.





$P^{\text{TSP}}(\text{seq}_n^1 | x) \quad < \quad P^{\text{TSP}}(\text{seq}_n^2 | x)$

Tour Decoding

- For auto-regressive decoding, i.e. selecting a city one at a time, P^{TSP} can be factorized with the chain rule :

$$P^{\text{TSP}}(i_1, ..., i_n|x) = \underbrace{P(i_1|x)}_{\substack{\text{Probability of} \\ \text{selecting city } i_1 \\ \text{given } x}} . \underbrace{P(i_2|i_1, x)}_{\substack{\text{Probability of} \\ \text{selecting city } i_2 \\ \text{given } i_1, x}} . \underbrace{P(i_3|i_2, i_1, x)}_{\substack{\text{Probability of} \\ \text{selecting city } i_3 \\ \text{given } i_2, i_1, x}} ... \underbrace{P(i_n|i_{n-1}, i_{n-2}, ..., x)}_{\substack{\text{Probability of} \\ \text{selecting city } i_n \\ \text{given } i_{n-1}, i_{n-2}, ..., x}}$$

- The decoding problem aims at finding the sequence $i_1, i_2, ..., i_n$ that maximizes the objective :

$$\max_{i_1,...,i_n} \Pi_{t=1}^{n} P(i_t|i_{t-1}, i_{t-2}, ...i_1, x)$$

Conditional probability
approximated by a neural network

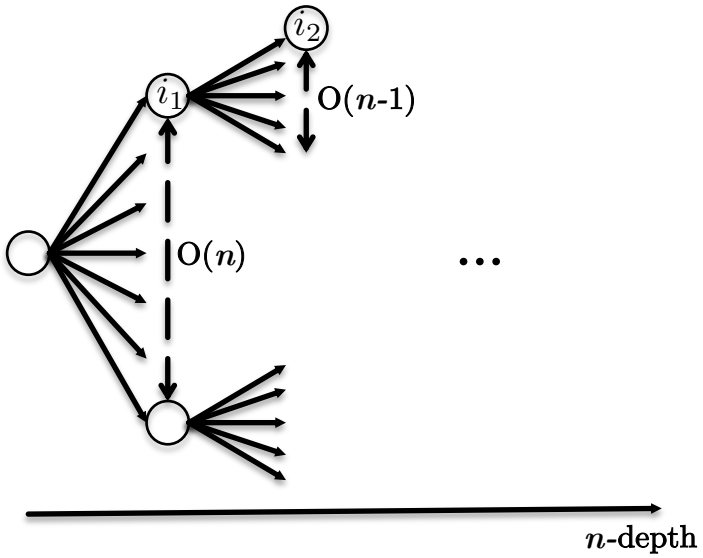
Decoding Techniques

- Exact search : Exhaustive search is intractable with O($n!$).

- Greedy search : At each time step, select the next city with the highest probability :

$$i_t = \arg\max_i \; P(i|i_{t-1}, i_{t-2}, ...i_1, x)$$

Complexity is O(n).



- Sampling techniques such as beam search or Monte Carlo Tree Search (MTCS) are known to improve results in NLP[1] and TSP[2-7].

Complexity is O(Bn).

[1] Tillmann, Ney, Word reordering and a dynamic programming beam search algorithm for statistical machine translation, 2003
[2] Nowak, Villar, Bandeira, Bruna, A Note on Learning Algorithms for Quadratic Assignment with Graph Neural Networks, 2017
[3] Kaempfer, Wolf, Learning the Multiple Traveling Salesmen Problem with Permutation Invariant Pooling Network, 2018
[4] Kool, Van Hoof, Welling, Attention, learn to solve routing problems!, 2018
[5] Joshi, Laurent, Bresson, An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem, 2019
[6] Wu, Song, Cao, Zhang, Lim, Learning Improvement Heuristics for Solving Routing Problems, 2020
[7] Xing, Tu, A Graph Neural Network Assisted Monte Carlo Tree Search Approach to Traveling Salesman Problem, 2020

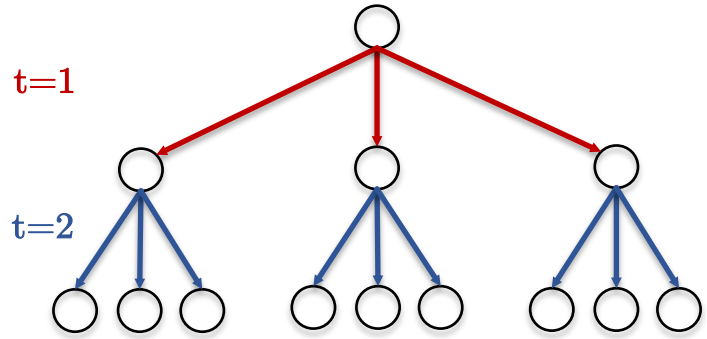
Beam Search[1]

- Beam search is a breadth-first search (BFS) technique where the breath has a limited size B.
- Beam search decoding problem :

$$\max_{\{i_1^b,\dots,i_n^b\}_{b=1}^B} \Pi_{b=1}^B P(i_1^b,...,i_n^b|x) \quad \text{s.t.} \quad \{i_1^b,...,i_n^b\} \neq \{i_1^{b'},...,i_n^{b'}\}, \ \forall b \neq b'$$

B sequences/ beams

Two beams are different.



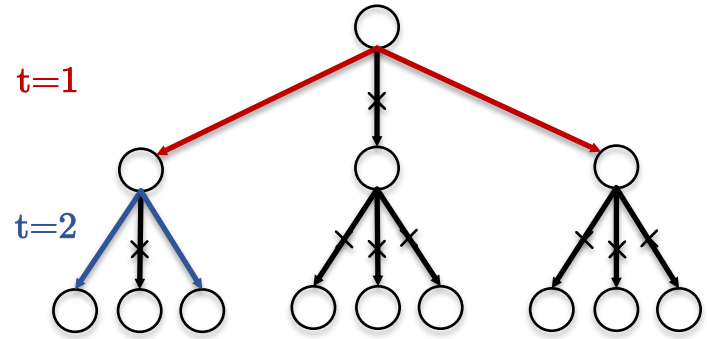
BFS explores the search space by expanding to all children nodes first.

- For B=1, the solution is given by greedy decoding.
- For B>1, the solution at t is determined by considering all possible extensions of B beams, and only keeping the Top-B probabilities :

$$\{i_1^b,...,i_t^b\}_{b=1}^B = \text{Top-B}\left\{\Pi_{k=1}^t P(i_k^b|i_{k-1}^b,i_{k-2}^b,...,i_1^b,x)\right\}_{b=1}^{B.(n-t)}$$

Or equivalently (for better numerical stabilities) :

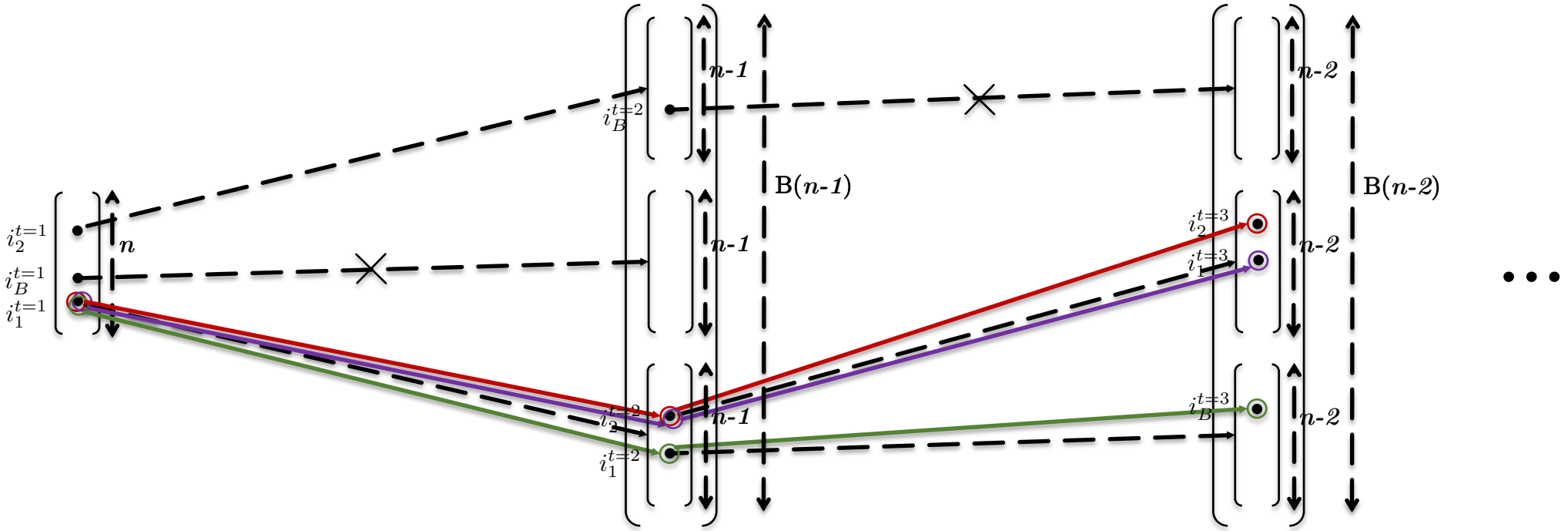
$$\{i_1^b,...,i_t^b\}_{b=1}^B = \text{Top-B}\left\{\sum_{k=1}^t \underbrace{\log P(i_k^b|i_{k-1}^b,i_{k-2}^b,...,i_1^b,x)}_{\text{Score } s(i_k|i_{k-1},i_{k-2},...,i_1,x)}\right\}_{b=1}^{B.(n-t)}$$



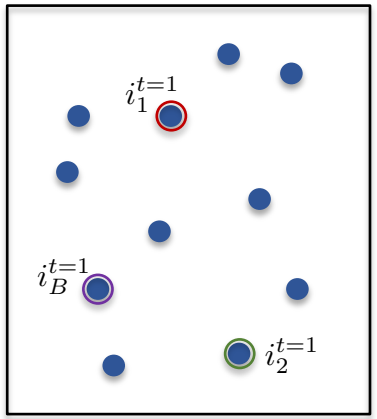
Beam Search explores the search space by expanding to a limited set of children nodes selected by a criteria. Beam size is B=2.

[1] Lowerre, The Harpy Speech Recognition System, 1976

Beam Search

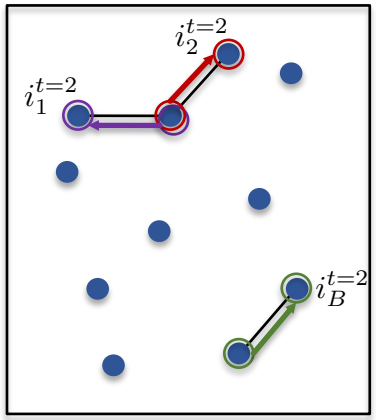


At t=1, run the net and get the starting cities with the top-B scores.



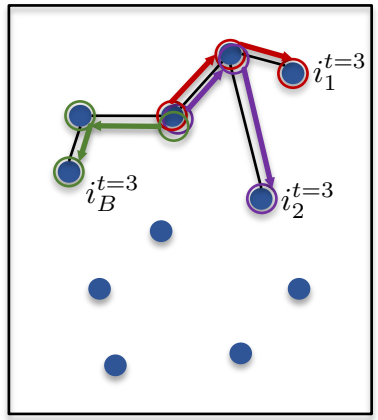
At t=2, run the net for each beam that can expand to B(n-1) possible cities. Keep the beams with top-B scores:

$s(i_2|i_1, x) + s(i_1|x)$



At t=3, run the net for each beam that can expand to B(n-2) possible cities. Keep the beams with top-B scores:

$s(i_3|i_2, i_1, x) +$
$s(i_2|i_1, x) + s(i_1|x)$



Outline

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

Numerical Experiments

- Proposed technique vs. SOTA

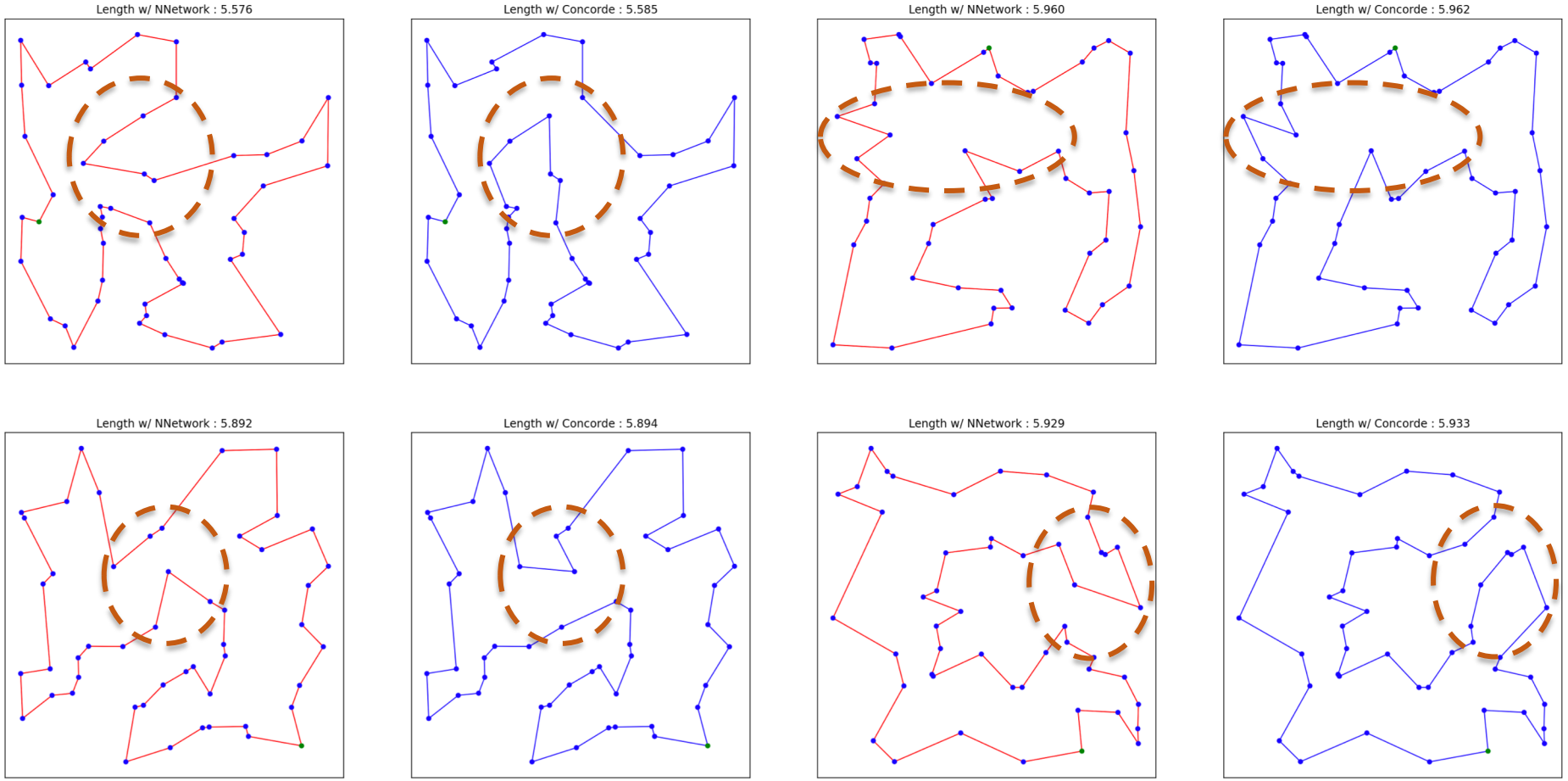
	Method	TSP50 Obj	TSP50 Gap	TSP50 T Time	TSP50 I Time	TSP100 Obj	TSP100 Gap	TSP100 T Time	TSP100 I Time
MIP	Concorde'06	**5.689**	0.00%	2m*	0.05s	**7.765**	0.00%	3m*	0.22s
	Gurobi'08	-	0.00%*	2m*	-	7.765*	0.00%*	17m*	-
Heuristic	Nearest insertion	7.00*	22.94%*	0s*	-	9.68*	24.73%*	0s*	-
	Farthest insertion	6.01*	5.53%*	2s*	-	8.35*	7.59%*	7s*	-
	OR tools'15	5.80*	1.83%*	-	-	7.99*	2.90%*	-	-
	LKH-3'17	-	0.00%*	5m*	-	**7.765***	0.00%*	21m*	-
Neural Network Greedy Sampling	Vinyals et-al'15	7.66*	34.48%*	-	-	-	-	-	-
	Bello et-al'16	5.95*	4.46%*	-	-	8.30*	6.90%*	-	-
	Dai et-al'17	5.99*	5.16%*	-	-	8.31*	7.03%*	-	-
	Deudon et-al'18	5.81*	2.07%*	-	-	8.85*	13.97%*	-	-
	Kool et-al'18	5.80*	1.76%*	2s*	-	8.12*	4.53%*	6s*	-
	Kool et-al'18 (our version)	-	-	-	-	8.092	4.21%	-	-
	Joshi et-al'19	5.87	3.10%	55s	-	8.41	8.38%	6m	-
	Our model	**5.707**	0.31%	13.7s	0.07s	**7.875**	1.42%	4.6s	0.12s
Neural Network Advanced Sampling	Kool et-al'18 (B=1280)	5.73*	0.52%*	24m*	-	7.94*	2.26%*	1h*	-
	Kool et-al'18 (B=5000)	5.72*	0.47%*	2h*	-	7.93*	2.18%*	5.5h*	-
	Joshi et-al'19 (B=1280)	5.70	0.01%	18m	-	7.87	1.39%	40m	-
	Xing et-al'20 (B=1200)	-	0.20%*	-	3.5s*	-	1.04%*	-	27.6s*
	Wu et-al'20 (B=1000)	5.74*	0.83%*	16m*	-	8.01*	3.24%*	25m*	-
	Wu et-al'20 (B=3000)	5.71*	0.34%*	45m*	-	7.91*	1.85%*	1.5h*	-
	Wu et-al'20 (B=5000)	5.70*	0.20%*	1.5h*	-	7.87*	1.42%*	2h*	-
	Our model (B=100)	5.692	0.04%	2.3m	**0.09s**	7.818	0.68%	4m	**0.16s**
	Our model (B=1000)	5.690	0.01%	17.8m	0.15s	7.800	0.46%	35m	0.27s
	Our model (B=2500)	**5.689**	4e-3%	44.8m	0.33s	**7.795**	0.39%	1.5h	0.62s

- Test with 10k TSP50 and TSP100.
- Results* are reported from other papers.
- T Time means total time for 10k TSP (in parallel).
- I Time means inference time to run a single TSP (in serial).
- Concorde[1] run on Intel Xeon Gold 6132 CPU.
  - The Concorde library includes over 700 functions.
- Neural networks run on Nvidia 2080Ti GPU.
  - 300 lines of code.
  - Soon release on GitHub.

[1] https://github.com/jvkersch/pyconcorde

Numerical Experiments

- Proposed technique (B=2500) vs. Concorde for TSP50 :

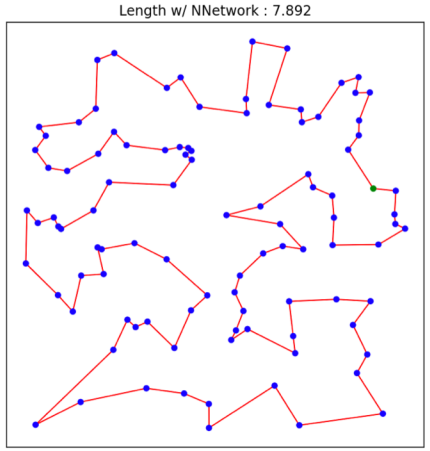


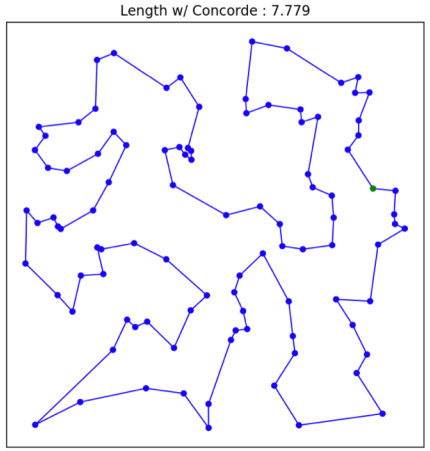
- Concorde is an exact solver when all edge lengths (distances between two cities) can be represented in a fixed-point arithmetic (32 bits before/after the decimal place).
- The Concorde code starts by converting all distances to the fixed-point arithmetic.
- If the distances are initially encoded by floating-point numbers (when using Python) then Concorde rounds the distances to the nearest integer for the fixed-point arithmetic. This means that the computed tour may not necessarily be the optimal tour for floating-point distances.
- See Section 5.4 in[1].

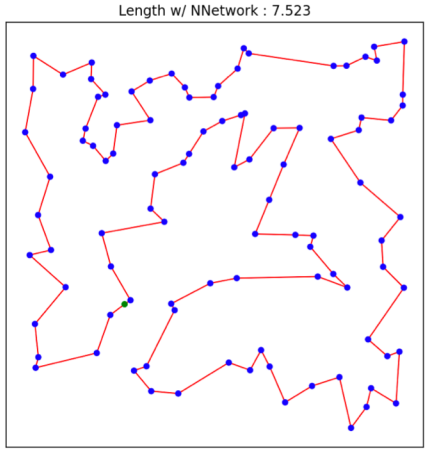
[1] Applegate, Bixby, Chvatal, Cook, The Traveling Salesman Problem: A Computational Study, 2006

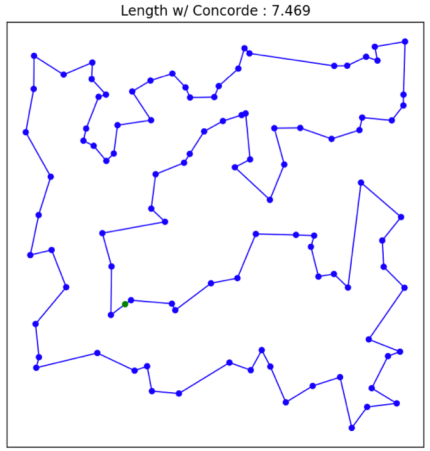
Numerical Experiments

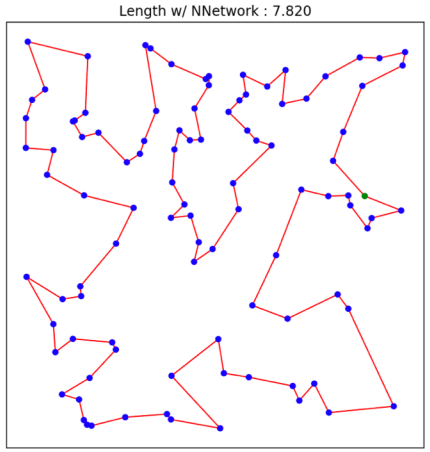
- Proposed technique (B=1000) vs. Concorde for TSP100 :

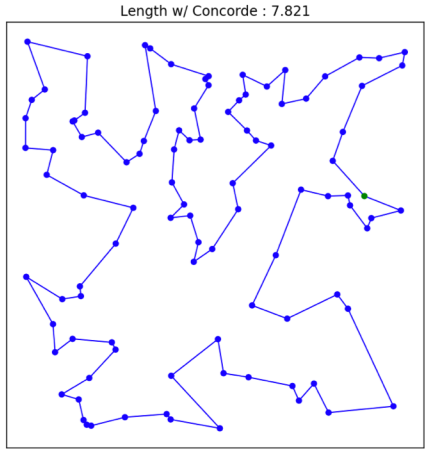


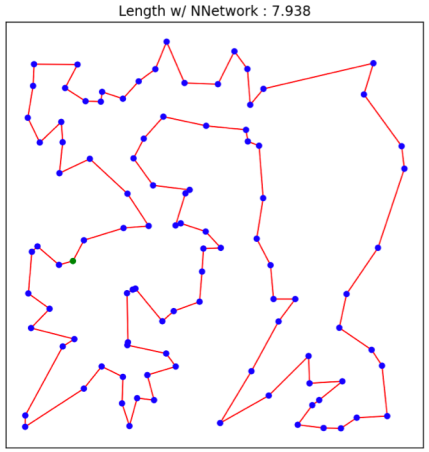


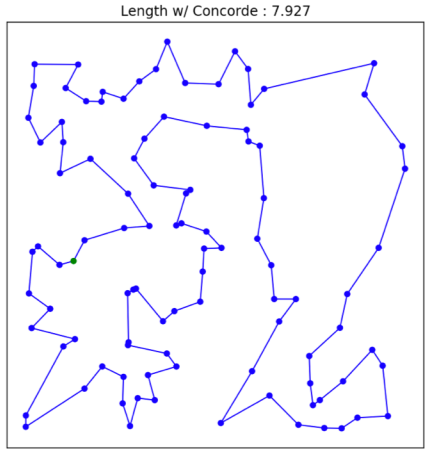








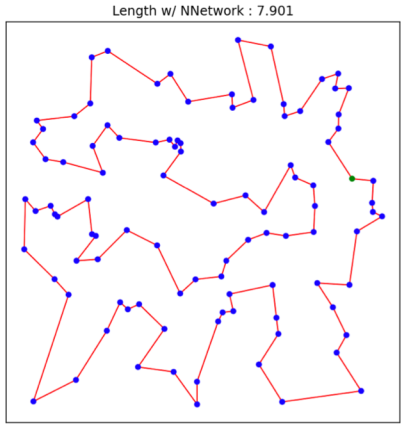


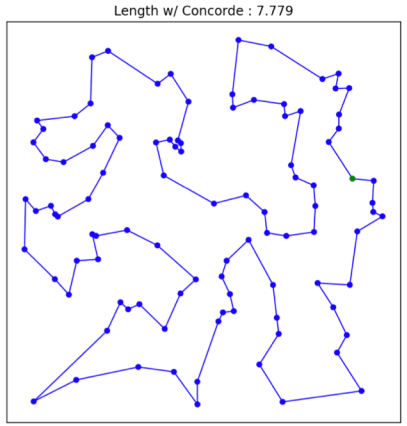


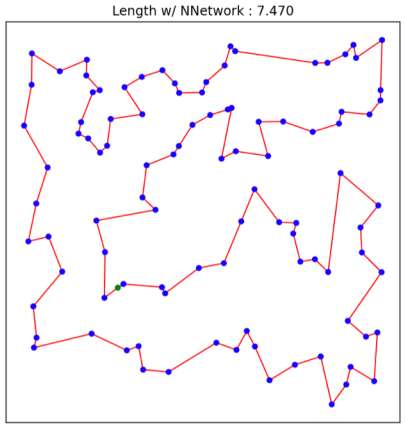
Numerical Experiments

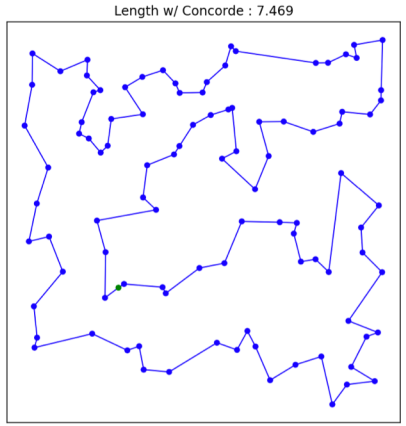
- Proposed technique (B=1000) vs. Concorde for TSP50⇒TSP100 :

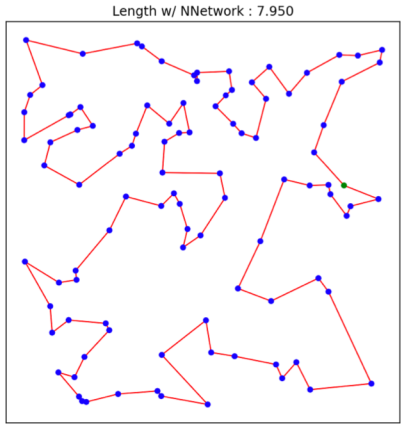
Method	TSP50 → TSP100 Obj	Gap	T Time	I Time
Concorde	7.765	0.00%	3m*	0.22s
Our model (greedy)	8.008	3.12%	4.4sec	0.13s
Our model (B=1000)	7.872	1.37%	1h	0.50s

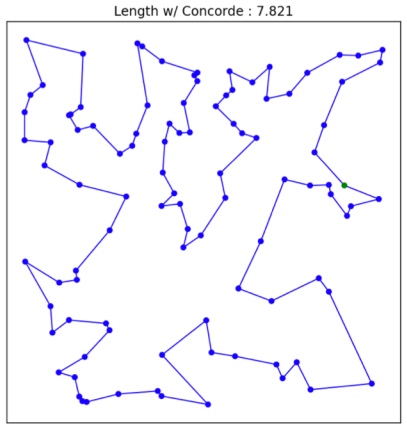


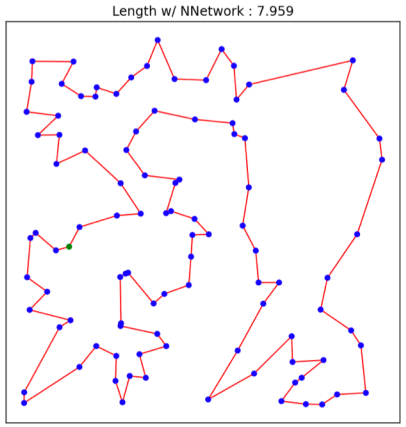


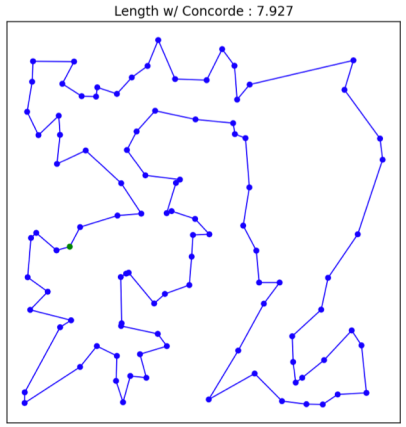








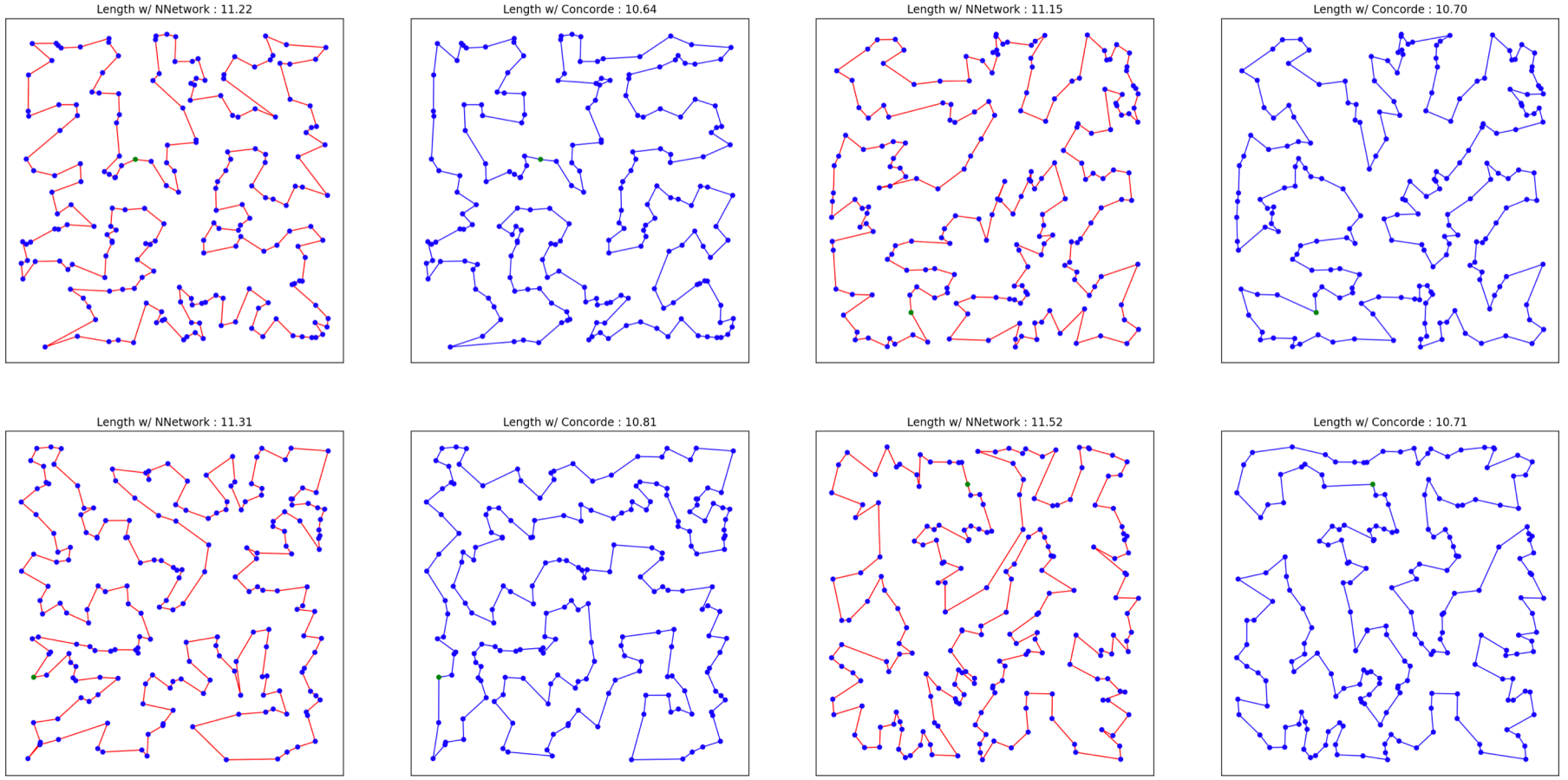




Numerical Experiments

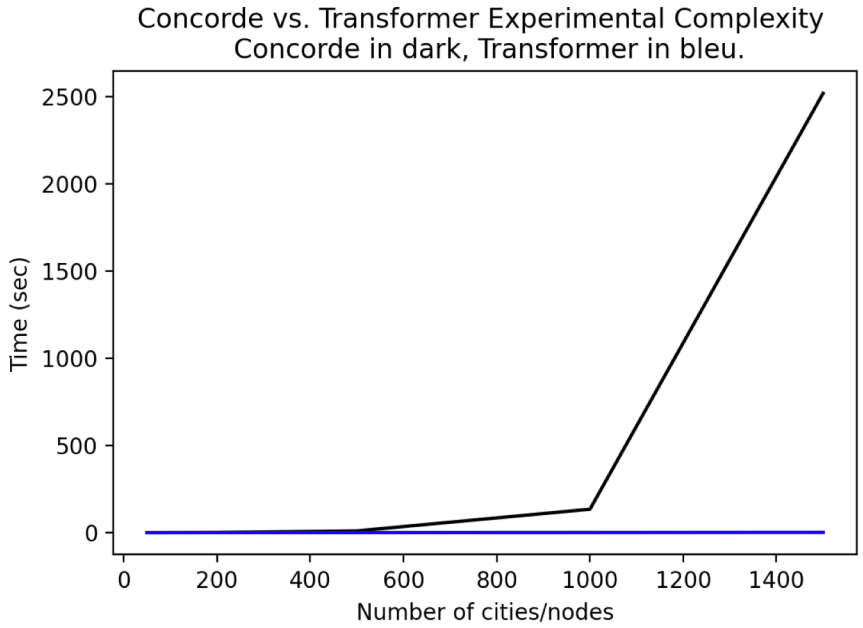
- Proposed technique (B=1000) vs. Concorde for TSP100⇒TSP200 :

Method	TSP100 → TSP200 Obj	Gap	T Time	I Time
Concorde	10.708	0.00%	-	1.29s
Our model (greedy)	11.353	6.02%	10.7s	0.24s
Our model (B=1000)	11.181	4.41%	2.2h	0.96s

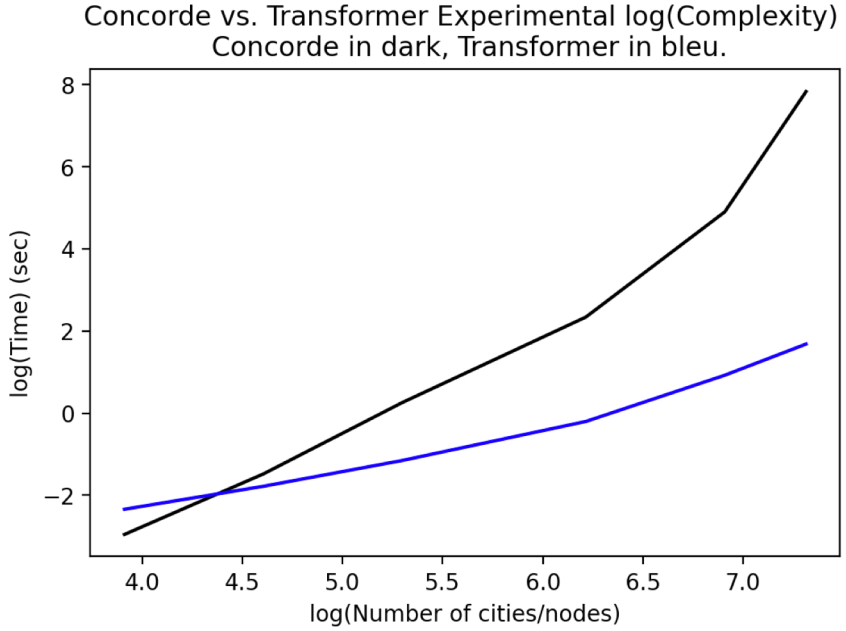


Our Model vs. Concorde[1] Complexity

- Experimental complexity (inference time for a single TSP) :



Complexity



Logarithmic complexity

[1] Applegate, Bixby, Chvatal, Cook, The Traveling Salesman Problem, 2006

Outline

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

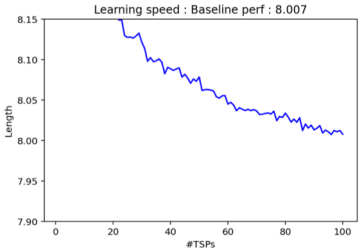
Future Work

- In this work, we essentially focused on the architecture.
  - Like for NLP and CV, the Transformer architecture can be successful to solve Combinatorial Optimization problem(s).
  - It improves recent learned heuristics with an optimal gap of 0.004% for TSP50 and 0.39% for TSP100.
- Further developments :
  - Better sampling techniques such as group beam-search[1,2] or MCTS[3] are known to improve results.
  - Use of heuristics like 2-Opt to get intermediate rewards has also shown improvements[4] (the tour length as global reward requires to wait the end of the tour construction).

[1] Vijayakumar, Cogswell, Selvaraju, Sun, Lee, Crandall, Batra, Diverse beam search: Decoding diverse solutions from neural sequence models, 2016
[2] Meister, Vieira, Cotterell, Best-first beam search, 2020
[3] Xing, Tu, A Graph Neural Network Assisted Monte Carlo Tree Search Approach to Traveling Salesman Problem, 2020
[4] Wu, Song, Cao, Zhang, Lim, Learning Improvement Heuristics for Solving Routing Problems, 2020

Discussion

- Curb your enthusiasm !
  - Traditional solvers like Concorde/LKH-3 outperform learning solvers in terms of (1) performance and (2) generalization.
  - But neural network solvers offer 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/LKH-3 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
- Decoding Technique
- 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.

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