Neural Network Solvers for Combinatorial Optimization (CO)

Graph 9. Auto-regressive CO Solvers

1

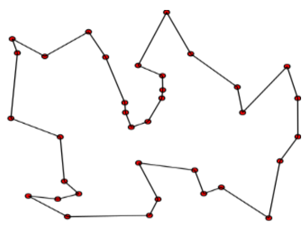
Lectures on Neural-network CO Solvers

- Graph 9: Autoregressive (AR) CO Solvers
- Graph 10: Non-autoregressive (NAR) CO Solvers
- Graph 11: Pre-trained Large Language Models for CO
- Graph 12: Neural Solvers for Mixed Integer Programming

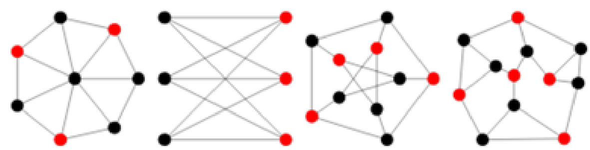
2

Combinatorial Optimization (CO)

Combinatorial Optimization (CO) is a subfield of mathematical optimization that consists of finding an optimal object from a finite set of objects,[1] where the set of feasible solutions is discrete or can be reduced to a discrete set.

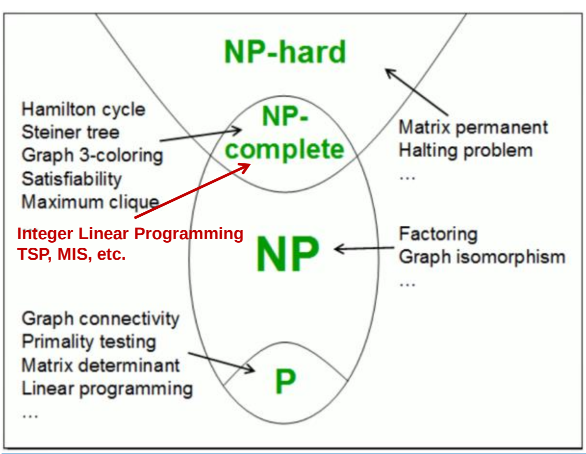


Traveling Salesmen Problem (TSP)



Maximal Independent Set (MIS)

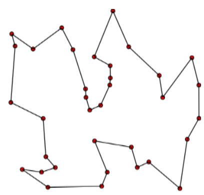
P, NP and NP Complete (NPC)



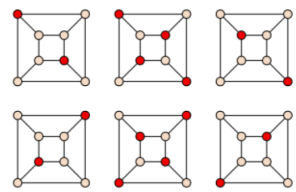
- **P**: The class of problems which can be solved by algorithms in polynomial time..
- **NP:** The class of problems for which a deterministic way to find a solution in polynomial time is not known; however, a guessed solution can be verified in polynomial time.
- **NP Complete (NPC):** If a problem is in NP and all other NP problems can be reduced to it in polynomial time, the problem is NP-complete.
- **We want**: $NPC \xrightarrow{approximate} P$

Why graphs are related?

Traveling Salesmen Problem (TSP)



Maximal Independent Set (MIS)

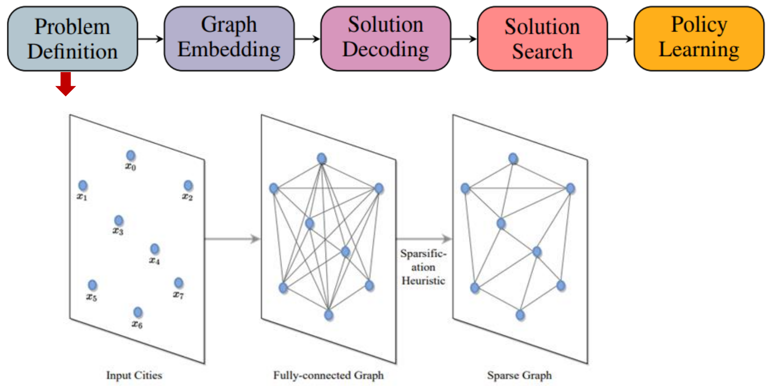


- Both can be defined as a search problem over a graph (for an optimal solution).
- A candidate solution can be generated by sequentially selecting a variable (node or edge) in each step under certain constraints.
- This process reminds us about Language Modeling for generating a word sequence.
- Can we use Transformer or ChartGPT to solve CO problems and beat existing methods?

TSP Solvers

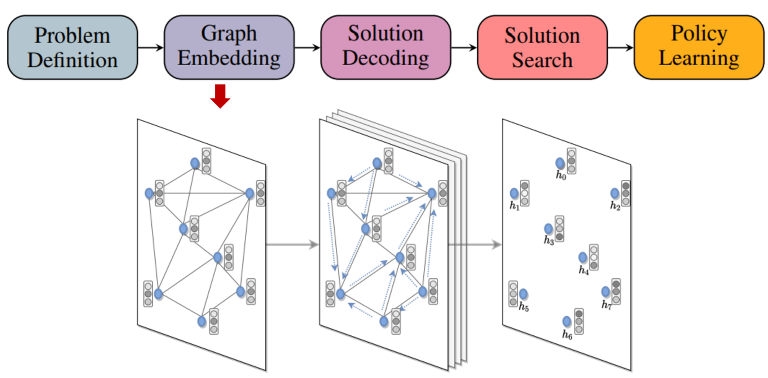
- **Traditional Algorithms**
  - Exhaustive search: $O(n!)$
  - Dynamic Programming: $O(n^2 2^n)$
  - Linear Programming: scaled to graphs with $n = 200$ nodes
  - Heuristic and approximate solvers: heavily depend on hand-craft heuristics
- **Neural Network Solvers (Recent ML models)**
  - DRL (deep reinforcement learning) solvers proposed recently (ICLR 2017, ICLR 2019), scaled to $n = 100$ nodes until 2022
  - DIMES (our DRL model): scaled $n = 10{,}000$ nodes (R Qui, Z Sun & Y Yang, NearIPS'2022)
  - DIFUSCO (our graph-based diffusion model): outperforming DIMES both in scalability and accuracy (Z Sun and Y Yang, ICML 2023)

Neural CO Pipeline [CK Joshi et al, CP 2021]



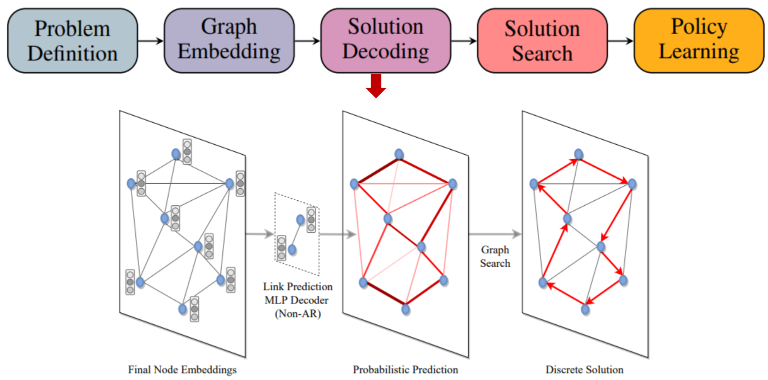
(b) **Problem Definition:** TSP is formulated via a fully-connected graph of cities/nodes. The graph can be sparsified via heuristics such as k-nearest neighbors.

Neural CO Pipeline



(c) **Graph Embedding:** Embeddings for each graph node are obtained using a Graph Neural Network encoder, which builds local structural features.

Neural CO Pipeline

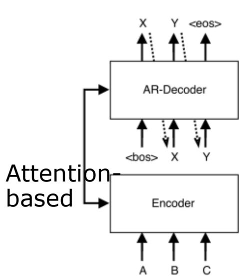


(d) **Solution Decoding & Search:** Probabilities are assigned to each node for belonging to the solution set, either independent of one-another (*i.e.* Non-autoregressive decoding) or conditionally through graph traversal (*i.e.* Autoregressive decoding). The predicted probabilities are converted into discrete decisions through classical graph search techniques such as greedy search or beam search.

9

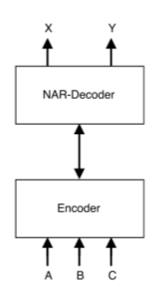
Decoding Strategies

Autoregressive (AR) Solvers



Encoder-Decoder Neural Networks

Non-autoregressive (NAR) Solvers



Encoder + Active Search (not Neural Networks)

10

Neural CO Pipeline

Problem Definition → Graph Embedding → Solution Decoding → Solution Search → Policy Learning

3.4 Solution Search

Greedy Search For AR decoding, the predicted probabilities at node i are used to select the edge to travel along at the current step via sampling from the probability distribution p_i or greedily selecting the most probable edge p_{ij}, *i.e.* greedy search. Since NAR decoders directly output probabilities over all edges independent of one-another, we can obtain valid TSP tours using greedy search to traverse the graph starting from a random node and masking previously visited nodes. Thus, the probability of a partial tour π' can be formulated as $p(\pi') = \prod_{j' \sim i' \in \pi'} p_{i'j'}$, where each node j' follows node i'.

Beam Search and Sampling During inference, we can increase the capacity of greedy search via limited width breadth-first beam search, which maintains the b most probable tours during decoding. Similarly, we can sample b solutions from the learnt policy and select the shortest tour among them. Naturally, searching longer, with more sophisticated techniques, or sampling more solutions allows trading off run time for solution quality. However, it has been noted that using large b for search/sampling or local search during inference may overshadow an architecture's inability to generalize [20]. To better understand generalization, we focus on using greedy search and beam search/sampling with small $b = 128$.

Neural CO Pipeline

Problem Definition → Graph Embedding → Solution Decoding → Solution Search → Policy Learning

- **Supervised** Learning from Labeled Training Data
  - $\mathcal{D} = \{(s_i, \boldsymbol{\pi}_i)\}$ for s_i as an instance graph and $\boldsymbol{\pi}_i$ as a ground-truth optimal solution for s_i.
  - Train a GNN AR model (RNN or Transformer based) or an MLP classifier over $\mathcal{D}$.
- **Unsupervised** Reinforcement Learning from a Cost Function
  1) $S = \{s_i\}$ is a large set of graphs without ground-truth optimal solution per graph.
  2) For each $s \in S$, generate feasible solutions ($\pi' s$) based on the current heatmap $P_\theta(\pi|s)$.
  3) Evaluate each feasible solution with a cost function, producing $cost(\pi_1), cost(\pi_2), cost(\pi_3), \ldots$;
  4) Use the costs of the explored solutions to update the heatmap with policy gradient;
  5) Repeat steps 2-4 for a prespecified number of iterations.

AR decoding v.s. NAR decoding

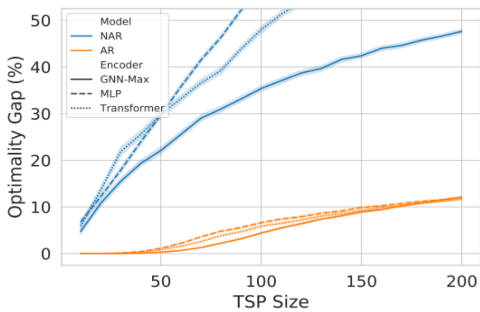


Figure 7: **Comparing AR and NAR decoders.** Sequential AR decoding is a powerful inductive bias for TSP as it enables significantly better generalization, even in the absence of graph structure (MLP encoders).

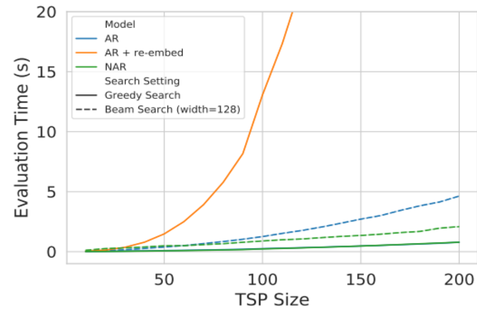


Figure 8: **Inference time for various decoders.** One-shot NAR decoding is significantly faster than sequential AR, especially when re-embedding the graph at each decoding step [39].

Lectures on Neural-network CO Solvers

- ✓ Introduction to ML for Combinatorial Optimization (CO)
- Graph 9: Autoregressive (AR) CO Solvers
  - Reinforcement learning with RNN-based networks [Bello*, Pham*, et al., **ICLR 2017**]
  - Reinforcement learning with Transformer-based networks [W Kool et al., **ICLR 2019**]
- Graph 10: Non-autoregressive (NAR) CO Solvers
- Graph 11: Pre-trained Large Language Models for CO
- Graph 12: Neural Solvers for Mixed Integer Programming

Outline of the Lectures (Graph 7, 8 and 9)

- Introduction to ML for Combinatorial Optimization (CO)
- Autoregressive (AR) CO Solvers
  - Reinforcement learning with RNN-based networks [Bello*, Pham*, et al., **ICLR 2017**]
  - Reinforcement learning with Transformer-based networks [W Kool et al., **ICLR 2019**]
- Non-autoregressive (NAR) CO Solvers
- Pre-trained Large Language Models [C Yang et al., **ICLR 2024**]

3/28/2024 @Yiming Yang, 11-741 S24 Graph9 AR CO Solvers 15

15

Recap: RNN-based Seq2seq Model

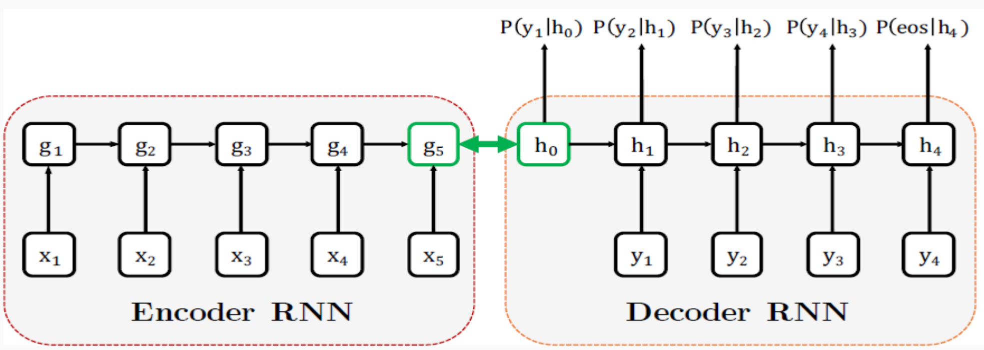


Figure 1: One-layer Seq2Seq Model.

Summarizing the whole sentence in single vector g_s

11/14/2024 @Yiming Yang, 11-741 Lecture on DL Attention 16

16

Autoregressive Factorization in Decoding

Denoting $x = (x_1, x_2, \cdots, x_S)$ and $y = (y_1, y_2, \cdots, y_T)$, we have

$$P_\theta(\mathrm{y}|\mathrm{x}) = \prod_{j=1}^{T} P_\theta\left(y_j|\mathrm{x}, y_{<j}\right) = \prod_{j=1}^{T} P_\theta\left(y_j|h_{i-1}\right) = \frac{exp\left(y_j^T h_{j-1}\right)}{\sum_{j'=1}^{M} exp\left(y_{j'}^T h_{j-1}\right)}$$

where h_{j-1} encodes the information of x, $y_{<j}$.

RNN-based AR Model for CO [Bello*, Pham*, et al., ICLR 2017]

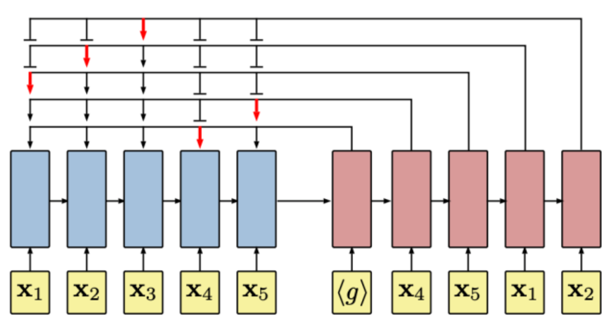
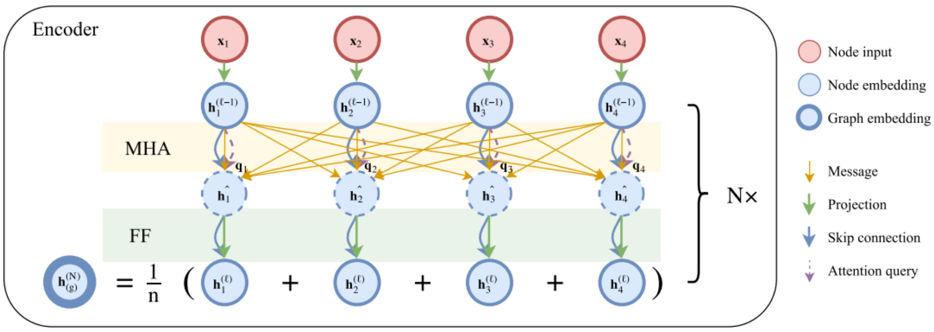


Figure 1: A pointer network architecture introduced by (Vinyals et al., 2015b).

Transformer-based AR Model with RL [W Kool et al., ICLR 2019]

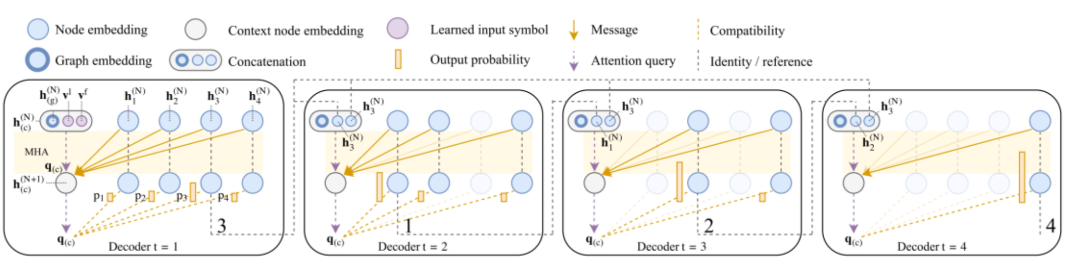
Transformer Encoder



19

Transformer-based AR Model with RL [W Kool et al., ICLR 2019]

Transformer Decoder



20

Notation in Model Optimization

- $s \in D$ is an instance graph in collection D.
- $\boldsymbol{\pi} = (\pi_1, \pi_2, \dots, \pi_n)$ is a feasible solution (valid TSP tour) in s, and π_i is an edge in the tour.
- $\boldsymbol{c_s}(\pi_i)$ is the (non-negative) cost of edge π_i and $\mathcal{L}(\boldsymbol{\pi}) = \sum_{i=1}^{n} \boldsymbol{c_s}(\pi_i)$ is the total cost of $\boldsymbol{\pi}$.
- $p_\theta(\boldsymbol{\pi}|s)$ is the probabilistic distribution learnable by a neural network
  - Ideally, we want $p_\theta(\boldsymbol{\pi}^*|s) = 1$ for any optimal solution and $p_\theta(\boldsymbol{\pi}|s) = 0$ otherwise.
  - Practically, we train a transformer model which assigns high $p_\theta(\boldsymbol{\pi}|s)$ values to near-optimal solutions.
- AR factorization: $p_\theta(\boldsymbol{\pi}|s) = \prod_{i=1}^{n} p_\theta(\pi_i|\pi_{<i}, s) = \prod_{i=1}^{n} P_\theta(\pi_i|h_{i-1})$

21

Optimization: RL with Policy Gradient

- Loosss function

$$\mathcal{L}_\theta(s) = E_{\pi\sim p_\theta(.|s)}\mathcal{L}(\boldsymbol{\pi}|s) = \sum_{\pi|s}[\mathcal{L}(\boldsymbol{\pi}|s)\, p_\theta(\boldsymbol{\pi}|s)] \qquad (1)$$

- Gradient for optimizing

$$\nabla_\theta\, \mathcal{L}_\theta(s) = E_{\pi\sim p_\theta(.|s\}}\mathcal{L}(\boldsymbol{\pi})\nabla_\theta \log\, p_\theta(\boldsymbol{\pi}|s) \qquad (2)$$

- To proof formula 2, let us use the trick below (omitting s for simplicity)

$$\nabla_\theta p_\theta(\pi) = p_\theta\,(\pi)\boxed{\frac{\nabla_\theta p_\theta(\pi)}{p_\theta(\pi)}} = p_\theta\,(\pi)\boxed{\nabla_\theta log\; p_\theta(\pi)} \qquad (3)$$

- On both sides of (3), multiply $\mathcal{L}(\boldsymbol{\pi})$ and sum over π we have

LHS: $\sum_\pi[\mathcal{L}(\boldsymbol{\pi})\nabla_\theta p_\theta(\boldsymbol{\pi})] = \nabla_\theta(\sum_\pi[\mathcal{L}(\boldsymbol{\pi})p_\theta(\boldsymbol{\pi})]) = \nabla_\theta\mathcal{L}_\theta(s)$ (4)

RHS: $\sum_\pi[\boxed{\mathcal{L}(\boldsymbol{\pi})}\, p_\theta(\pi)\boxed{\nabla_\theta log\; p_\theta(\pi)}] = E_{\pi\sim p_\theta(.)}[\mathcal{L}(\boldsymbol{\pi})\nabla_\theta log\; p_\theta(\pi)]$ (5)

22

Optimization

- Gradient for minimization

$$\nabla \mathcal{L}_\theta(s) = \sum_{\pi \sim p_\theta(.|s)} \mathcal{L}(\boldsymbol{\pi}) \nabla \log p_\theta(\boldsymbol{\pi}|s) \qquad (2)$$

- Adjusted gradient

$$\nabla \mathcal{L}_\theta(s) = \sum_{\boldsymbol{\pi} \sim p_{\theta(.|s)}} [\mathcal{L}(\boldsymbol{\pi}) - b_s] \nabla \log p_{\theta(\pi|s)}$$

  - b_s is the cost of the predicted tour by a baseline system (GreedyRollout), reflecting the difficulty of problem s;
  - $\mathcal{L}(\boldsymbol{\pi}) - b_s$ is the adjusted weight, reflecting the additional cost the system-predicted $\boldsymbol{\pi}$ is compared to that by the baseline.

Transformer-based AR Model with RL [W Kool et al., ICLR 2019]

Algorithm 1 REINFORCE with Rollout Baseline

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

Model update with policy gradient

Table 1: Attention Model (AM) vs baselines. The gap % is w.r.t. the best value across all methods.

Exact Solvers (no learning)

ML Solvers w/ Greedy Search

ML Solvers w/ Sampling

RNN-based

Transformer-based

	Method	$n=20$ Obj.	Gap	Time	$n=50$ Obj.	Gap	Time	$n=100$ Obj.	Gap	Time
TSP	Concorde	3.84	0.00%	(1m)	5.70	0.00%	(2m)	7.76	0.00%	(3m)
	LKH3	3.84	0.00%	(18s)	5.70	0.00%	(5m)	7.76	0.00%	(21m)
	Gurobi	3.84	0.00%	(7s)	5.70	0.00%	(2m)	7.76	0.00%	(17m)
	Gurobi (1s)	3.84	0.00%	(8s)	5.70	0.00%	(2m)		-	
	Nearest Insertion	4.33	12.91%	(1s)	6.78	19.03%	(2s)	9.46	21.82%	(6s)
	Random Insertion	4.00	4.36%	(0s)	6.13	7.65%	(1s)	8.52	9.69%	(3s)
	Farthest Insertion	3.93	2.36%	(1s)	6.01	5.53%	(2s)	8.35	7.59%	(7s)
	Nearest Neighbor	4.50	17.23%	(0s)	7.00	22.94%	(0s)	9.68	24.73%	(0s)
	Vinyals et al. (gr.)	3.88	1.15%		7.66	34.48%			-	
	Bello et al. (gr.)	3.89	1.42%		5.95	4.46%		8.30	6.90%	
	Dai et al.	3.89	1.42%		5.99	5.16%		8.31	7.03%	
	Nowak et al.	3.93	2.46%			-			-	
	EAN (greedy)	3.86	0.66%	(2m)	5.92	3.98%	(5m)	8.42	8.41%	(8m)
	AM (greedy)	**3.85**	**0.34%**	(0s)	**5.80**	**1.76%**	(2s)	**8.12**	**4.53%**	(6s)
	OR Tools	3.85	0.37%		5.80	1.83%		7.99	2.90%	
	Chr.f. + 2OPT	3.85	0.37%		5.79	1.65%			-	
	Bello et al. (s.)		-		5.75	0.95%		8.00	3.03%	
	EAN (gr. + 2OPT)	3.85	0.42%	(4m)	5.85	2.77%	(26m)	8.17	5.21%	(3h)
	EAN (sampling)	3.84	0.11%	(5m)	5.77	1.28%	(17m)	8.75	12.70%	(56m)
	EAN (s. + 2OPT)	3.84	0.09%	(6m)	5.75	1.00%	(32m)	8.12	4.64%	(5h)
	AM (sampling)	**3.84**	**0.08%**	(5m)	**5.73**	**0.52%**	(24m)	**7.94**	**2.26%**	(1h)
CVRP	Gurobi	6.10	0.00%			-			-	
	LKH3	6.14	0.58%	(2h)	10.38	0.00%	(7h)	15.65	0.00%	(13h)
	RL (greedy)	6.59	8.03%		11.39	9.78%		17.23	10.12%	
	AM (greedy)	**6.40**	**4.97%**	(1s)	**10.98**	**5.86%**	(3s)	**16.80**	**7.34%**	(8s)

Concluding Remarks

- **Good News**: Early AR solvers can fund near-optimal solutions for small problems ($n \leq 100$ nodes) .
- **Bad News**: Early AR solvers cannot beat exact solvers because the latter guarantees to find optimal solutions when they can scale.
- **The Hope: Scaling up** neural solvers for large problems that cannot be solved by exact solvers.
- **Recent Development**: NAR neural solvers (DIMES in NeurIPS 2022 and DIFUSCO in NeurIPS 2023) and pretrained LLMs (e.g. OPRO in ICLR 2024)

References

- [ICLR 2017] Neural Combinatorial Optimization with Reinforcement Learning. Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, Samy Bengio.
- [NIPS 2018] Zhuwen Li, Qifeng Chen, Vlallen Koltun. Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search.
- [ICLR 2019] Attention, Learn to Solve Routing Problems! Wouter Kool, Herke van Hoof, Max Welling.
- [CP 2021] Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, Thomas Laurent. Learning the Travelling Salesperson Problem Requires Rethinking Generalization. Constraint Programming 2021