Attention, Learn to Solve Routing Problems!

Wouter Kool, Herke van Hoof, Max Welling

University of Amsterdam & ORTEC

Presented by: George Giapitzakis

Routing Problems: What and Why?

Key Problems:

- ➔ **TSP:** Shortest tour visiting all cities (e.g., delivery routes)
- ➔ **VRP:** Multiple vehicles with capacity constraints (e.g., logistics)
- ➔ **OP:** Maximize prizes while staying under distance limit (e.g., tourist routes)
- ➔ **PCTSP:** Balance tour length with penalties for skipped nodes
- ➔ **SPCTSP:** Like PCTSP but penalties are not stochastic (only the mean is known a priori)

Challenge: NP-hard problems require efficient heuristics

Key Insight: Learn heuristics with neural networks instead of hand-crafting

Previous Approaches

- **Traditional Methods:**
  - Exact solvers (Concorde, Gurobi) - Optimal but slow
  - Handcrafted heuristics (Nearest Neighbor, Insertion)
- **Learning-Based:**
  - Pointer Networks (Vinyals et al. 2015)
  - Reinforcement Learning (Bello et al. 2016)
  - Graph Neural Networks (Nowak et al. 2017)

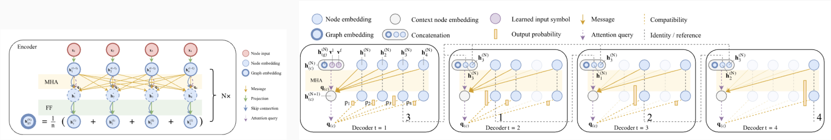
Limitations: Problem-specific architectures, suboptimal training methods

Attention Model Overview

Encoder-Decoder Structure:

- ➔ **Encoder:** Processes node features with self-attention layers
- ➔ **Decoder:** Builds solution sequentially using masked attention

In simple terms: Encoder creates feature, decoder constructs path one node at a time (autoregressive)



(a) Encoder. **(b)** Decoder.

Figure 1: Encoder and decoder architectures.

Key Technical Details

Encoder Features:

- ➔ Node coordinates (+ demands/prizes for variants)
- ➔ $N = 3$ attention layers with multi-head attention (8 heads)
- ➔ Batch normalization instead of layer norm

Decoder Dynamics:

- ➔ Context = [graph embedding, last node embedding, first node embedding]
- ➔ Masking enforces problem constraints (eg. mask all visited nodes for TSP)
- ➔ Single-head attention for final probability distribution

REINFORCE: Policy Gradient Basics

Goal: Maximize expected reward $J(\theta) = \mathbb{E}_{\pi_\theta}[R(\tau)]$

Key Idea: Adjust policy parameters θ using gradient ascent:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[R(\tau) \nabla_\theta \log \pi_\theta(\tau)\right]$$

- ➔ **Monte Carlo Approach:** Uses full trajectories (τ)
- ➔ **Unbiased but Noisy:** High variance due to stochasticity
- ➔ **Baseline Trick:** Subtract a baseline $b(s)$ to reduce variance:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[(R(\tau) - b(s)) \nabla_\theta \log \pi_\theta(\tau)\right]$$

Intuition: Reinforce actions leading to rewards. Use a baseline to stabilize the trajectory.

REINFORCE with Greedy Rollout Baseline

Key Innovation: Pick as baseline the cost from *greedy application* of current policy

- **Training Step:**
  - Sample solution $\pi \sim p_\theta(\cdot|s)$ (stochastic)
  - Compute baseline $b(s) = \text{Cost}(\text{greedy-rollout}(s))$
  - Update θ using gradient:

$$\nabla \mathcal{L} = (L(\pi) - b(s)) \nabla_\theta \log p_\theta(\pi|s)$$

- **Baseline Policy:**
  - Frozen copy of θ updated periodically (paired t-test)
  - Prevents "chasing a moving target"

Why This Works? Greedy rollout captures instance difficulty (harder instances will have higher cost)

Algorithm Pseudocode

Algorithm 1 REINFORCE with Rollout Baseline

Require: Number of epochs E, steps per epoch T, batch size B, significance α

Initialize θ, $\theta^{BL} \leftarrow \theta$

for epoch $= 1, \ldots, E$ **do**

  for step $= 1, \ldots, T$ **do**

    $s_i \leftarrow \text{RandomInstance}() \quad \forall i \in \{1, \ldots, B\}$

    $\pi_i \leftarrow \text{SampleRollout}(s_i, p_\theta) \quad \forall i \in \{1, \ldots, B\}$

    $\pi_i^{BL} \leftarrow \text{GreedyRollout}(s_i, p_{\theta^{BL}}) \quad \forall i \in \{1, \ldots, B\}$

    $\nabla \mathcal{L} \leftarrow \sum_{i=1}^{B} (L(\pi_i) - L(\pi_i^{BL})) \nabla_\theta \log p_\theta(\pi_i)$

    $\theta \leftarrow \text{Adam}(\theta, \nabla \mathcal{L})$

  end for

  if $\text{OneSidedPairedTTest}(p_\theta, p_{\theta^{BL}}) < \alpha$ **then**

    $\theta^{BL} \leftarrow \theta$

  end if

Experiments

Contributions and Impact

Key Advances:

- ➔ First general architecture for multiple routing problems
- ➔ Superior training with greedy rollout baseline
- ➔ Practical efficiency close to specialized solvers
- ➔ Parallelizable due to attention (instead of LSTMs)

Future Directions:

- ➔ Scaling to larger instances
- ➔ Handling complex constraints
- ➔ Integration with classical methods (backtracking)

Questions?

Thank you!
Any Questions?