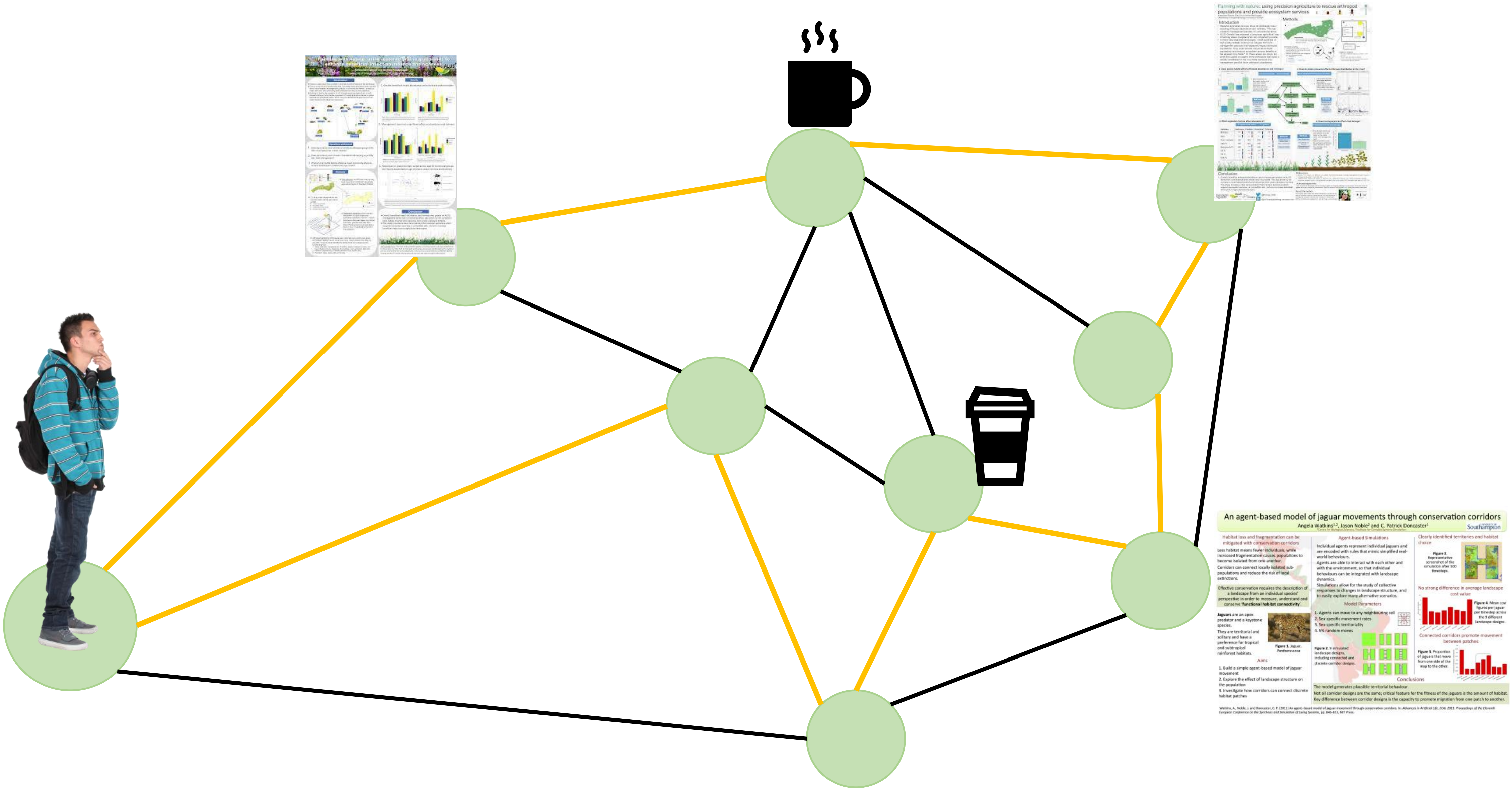
Attention, Learn to Solve Routing Problems

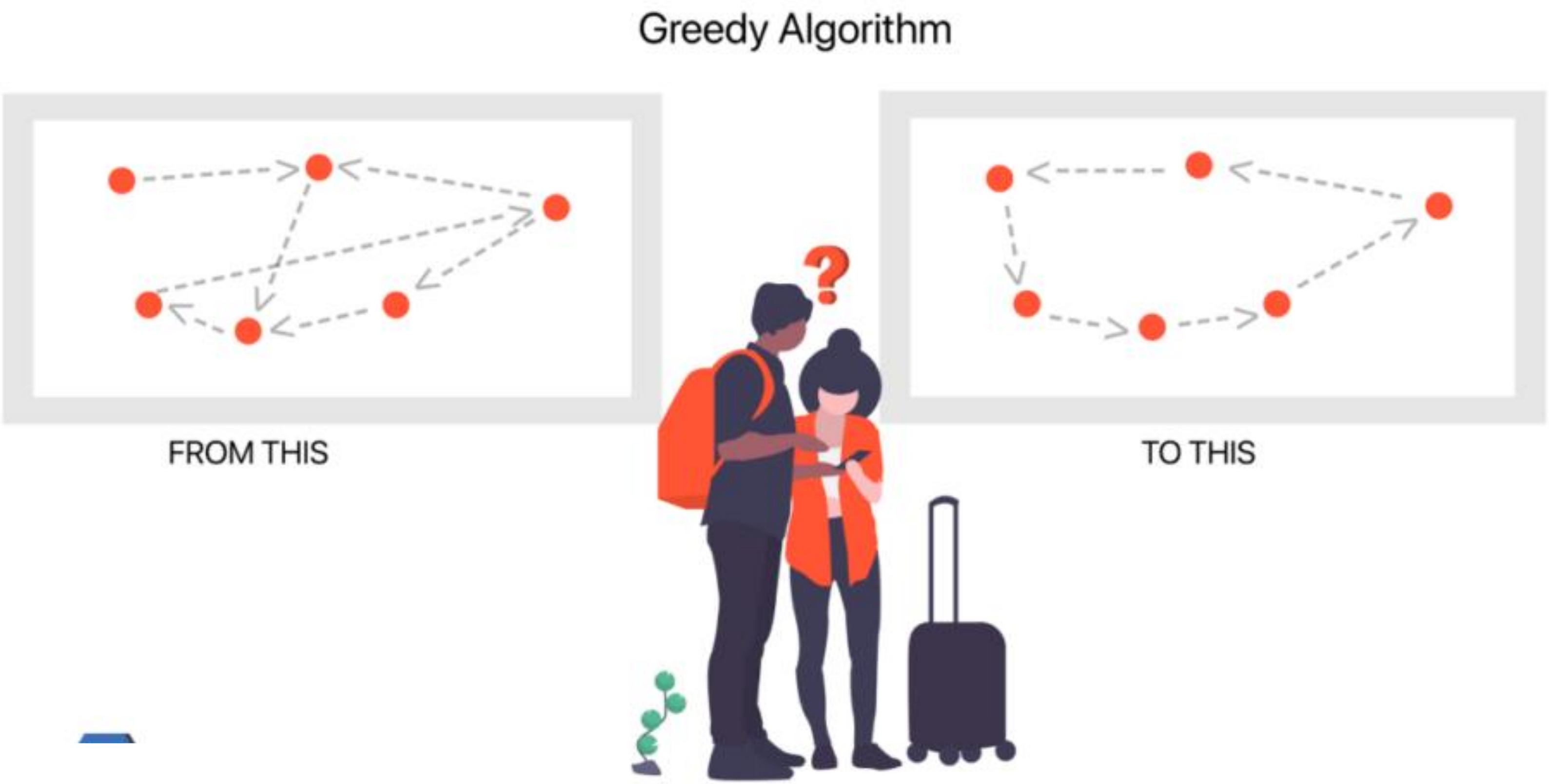
Wouter Kool, Herke van Hoof, Max Welling

ICLR 2019

Conference

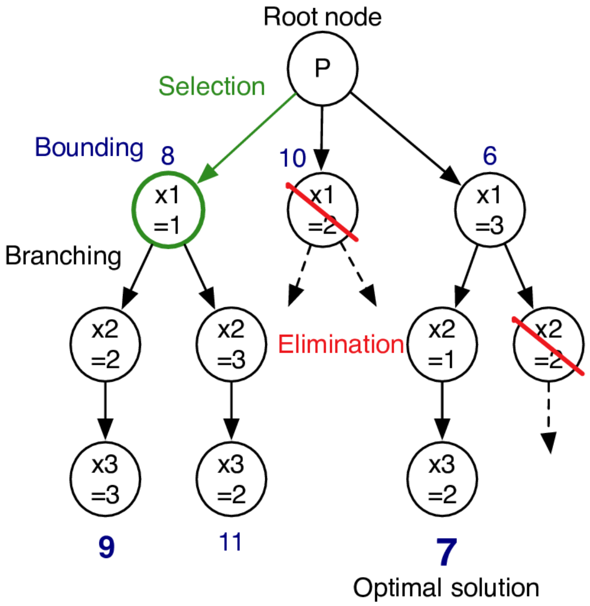

An agent-based model of jaguar movements through conservation corridors

Travelling Salesman Problem

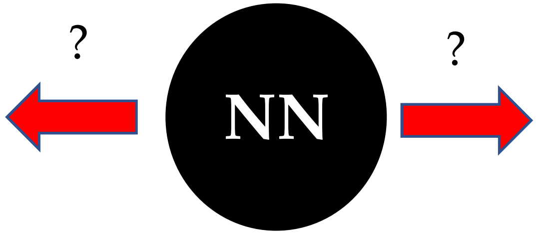


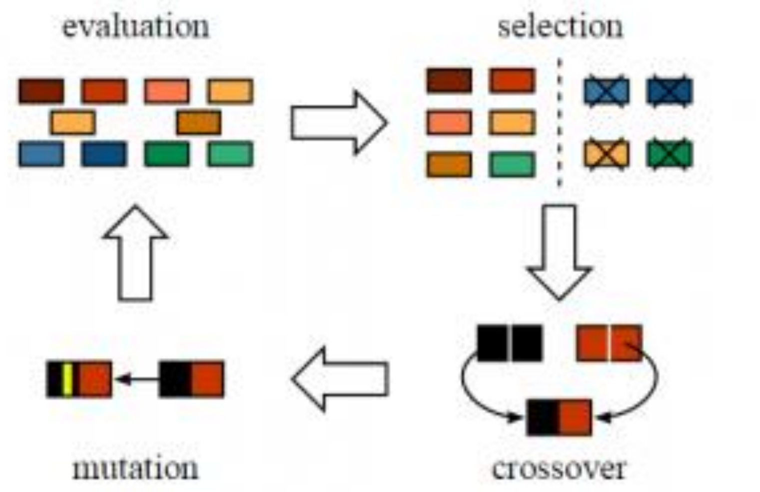
Solutions

- **Exact Solution:** Linear Programming using **Branch and Bound**
- **Heuristics:** trade off optimality for computational c



Exact Solution
Branch and Bound



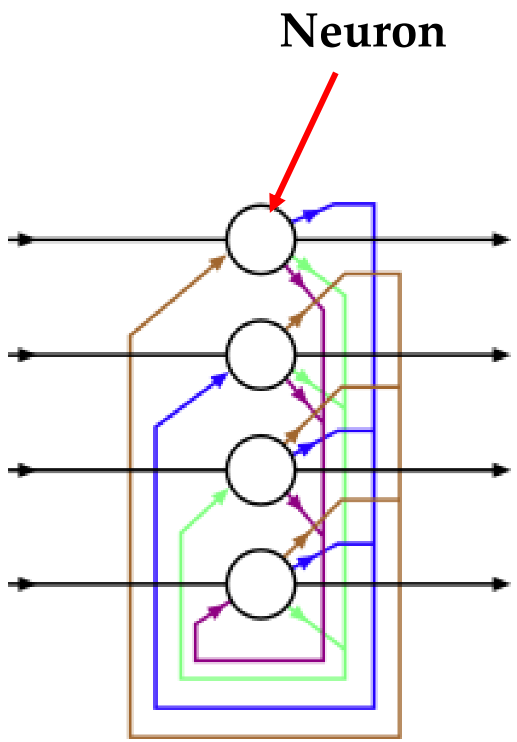


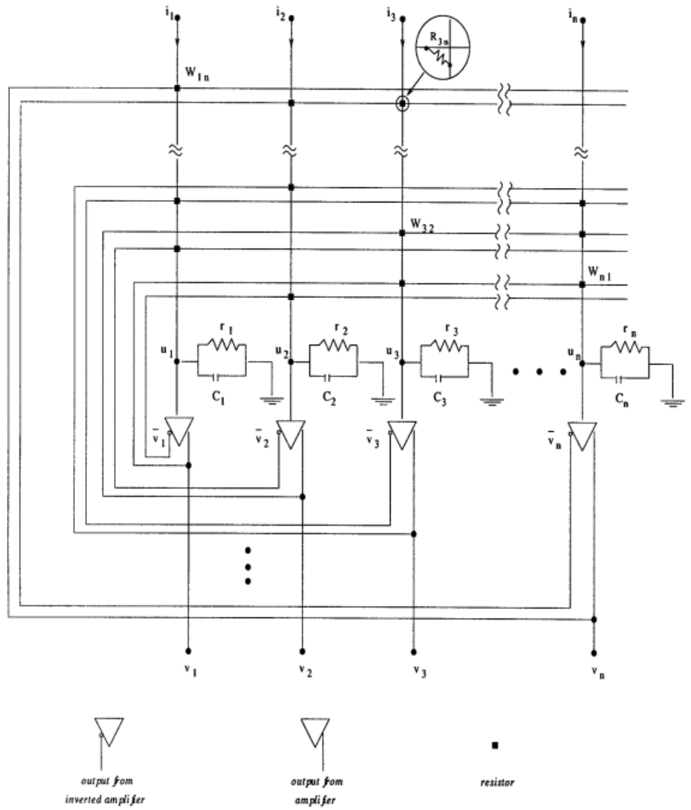
Meta Heuristics
Genetic Algorithm

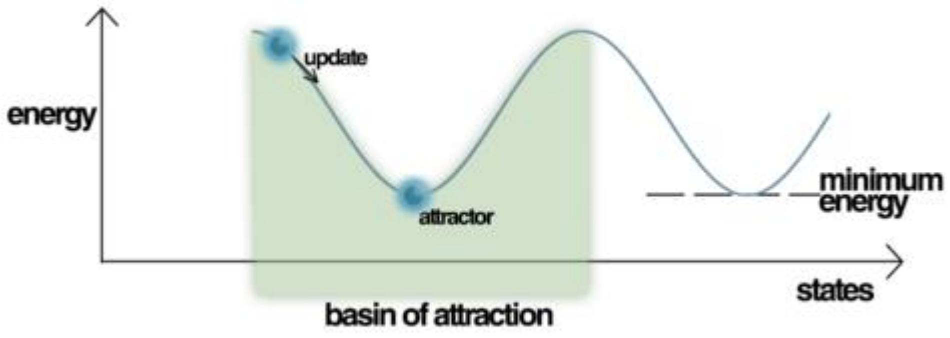
딥러닝 접근 케이스

- Hopfield-Network
- [Supervised Learning] Pointer Network
- [Reinforcement Learning] Bello, Actor-Critic to train the PN
- [Graph Neural Network] Nowak et al. Graph Neural Network
- **[Attention] Attention, Learn to Solve Routing Problems**

Hopfield Network, Hopfield & Tank (1985)



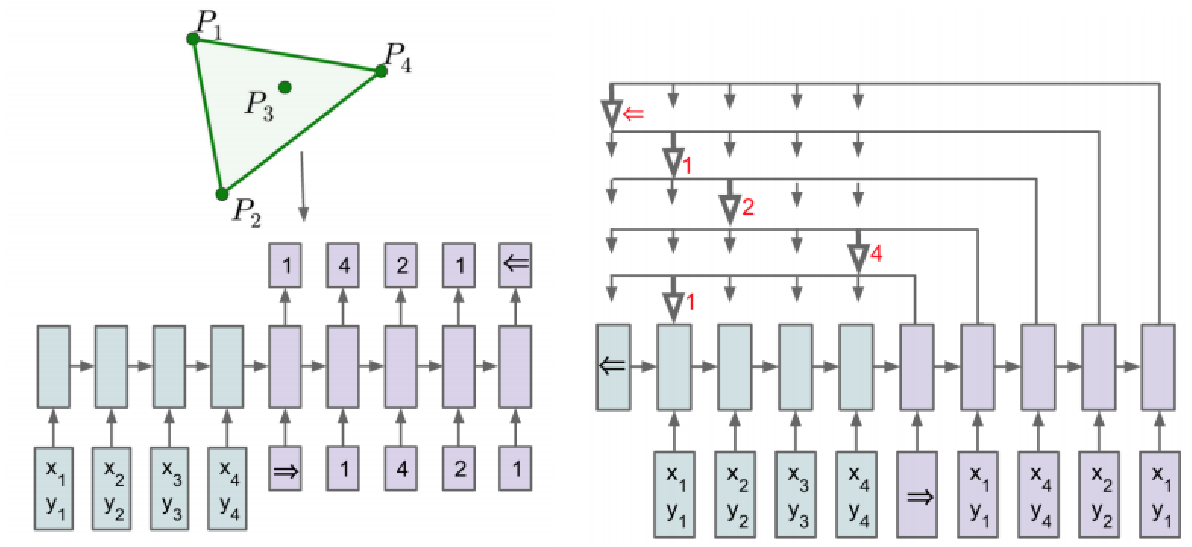




- Energy Based
- it either decreases or stays the same upon network units being updated

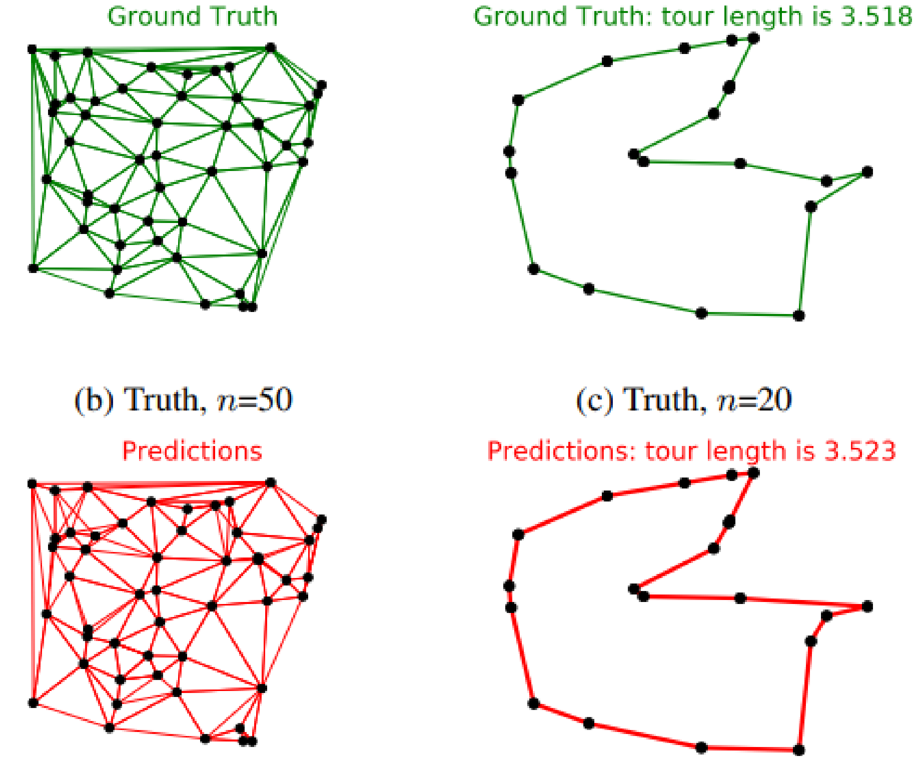
Hopfield
Network

Pointer Network, Vinyals et al. (2015)



(a) Sequence-to-Sequence

(b) Ptr-Net

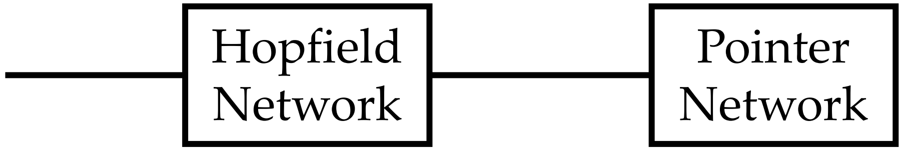


(b) Truth, n=50

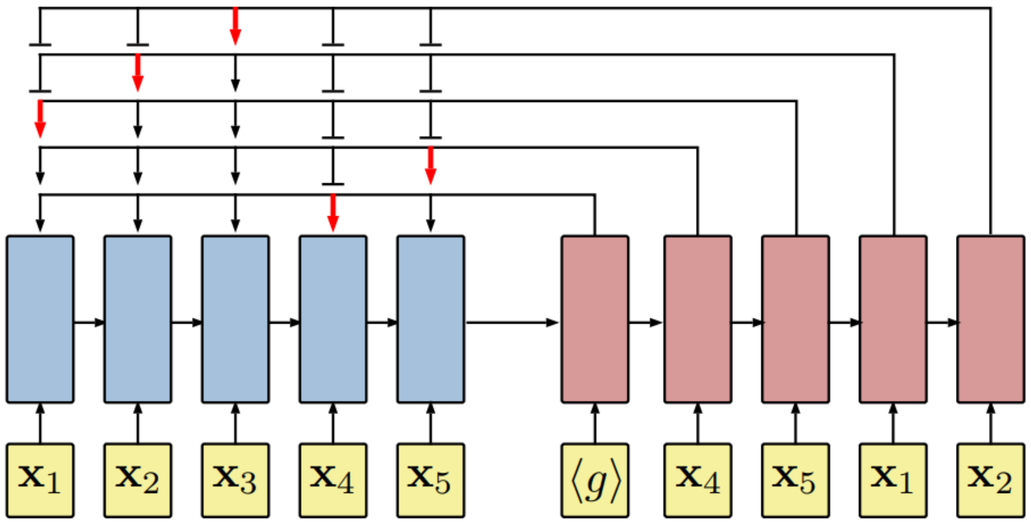
(c) Truth, n=20

(e) Ptr-Net , m=50, n=50

(f) Ptr-Net , m=5-20, n=20

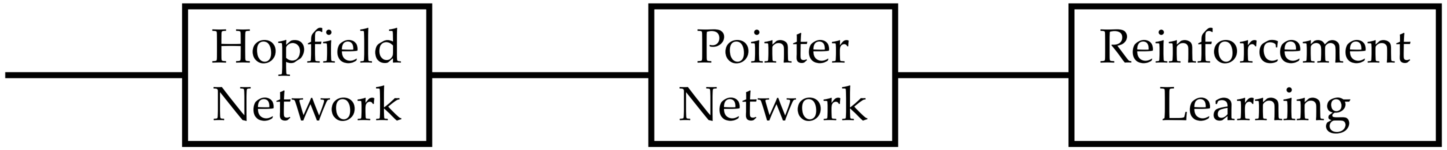


Using Reinforcement Learning, Bello et al. (2017)



Algorithm 1 Actor-critic training

procedure TRAIN(training set S, number of training steps T, batch size B)
    Initialize pointer network params θ
    Initialize critic network params θ_v
    for $t = 1$ to T **do**
        $s_i \sim \textsc{SampleInput}(S)$ for $i \in \{1, \ldots, B\}$
        $\pi_i \sim \textsc{SampleSolution}(p_\theta(.|s_i))$ for $i \in \{1, \ldots, B\}$
        $b_i \leftarrow b_{\theta_v}(s_i)$ for $i \in \{1, \ldots, B\}$
        $g_\theta \leftarrow \frac{1}{B} \sum_{i=1}^{B} (L(\pi_i|s_i) - b_i) \nabla_\theta \log p_\theta(\pi_i|s_i)$
        $\mathcal{L}_v \leftarrow \frac{1}{B} \sum_{i=1}^{B} \|b_i - L(\pi_i)\|_2^2$
        $\theta \leftarrow \textsc{Adam}(\theta, g_\theta)$
        $\theta_v \leftarrow \textsc{Adam}(\theta_v, \nabla_{\theta_v} \mathcal{L}_v)$
    end for
    return θ
end procedure



Graph Convolution, Joshi et al. (2019)

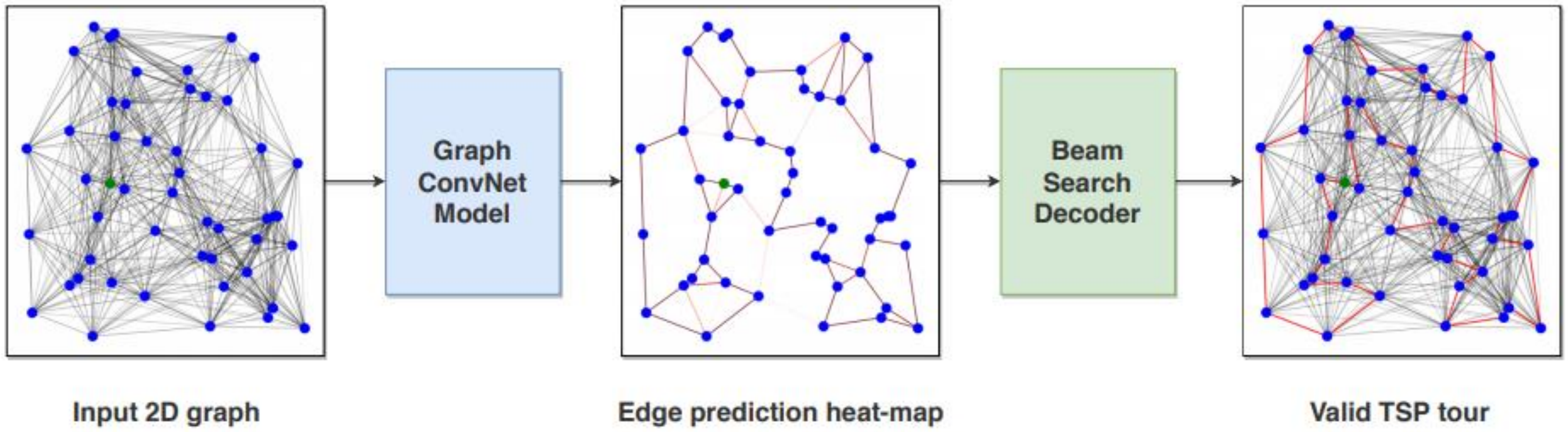
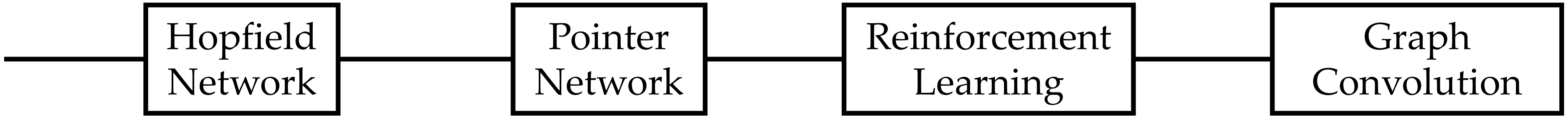


Figure 1: Overview of our approach. Taking a 2D graph as input, the graph ConvNet model outputs an edge adjacency matrix denoting the probabilities of edges occurring on the TSP tour. This is converted to a valid tour using beam search. All components are highly parallelized and solutions are produced in a one-shot, non-autoregressive manner.



문제 정의

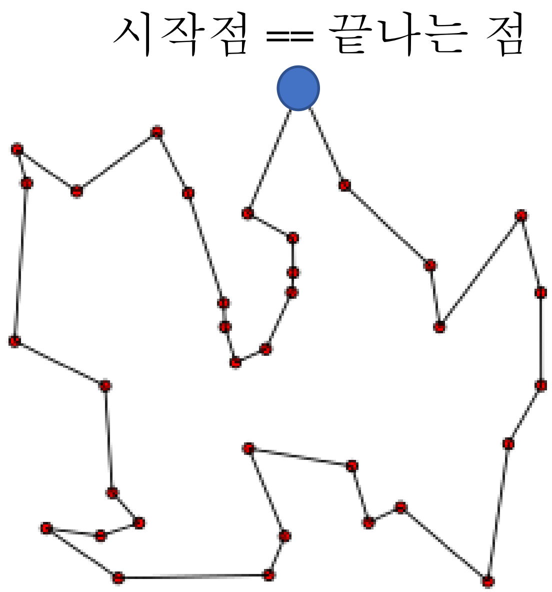
- A problem instance $\boldsymbol{s}$ is a graph with n nodes, where i $\in \{1, \cdots, n\}$
- For TSP, x_i is the coordinate of node i and the graph is fully connected
- A solution (tour) $\boldsymbol{\pi} = (\pi_1, \cdots, \pi_n)$ as a permutation of the nodes,

$$\pi_t \in \{1, \cdots, n\}, \pi_t \neq \pi_{t'}, \forall\, t \neq t'$$

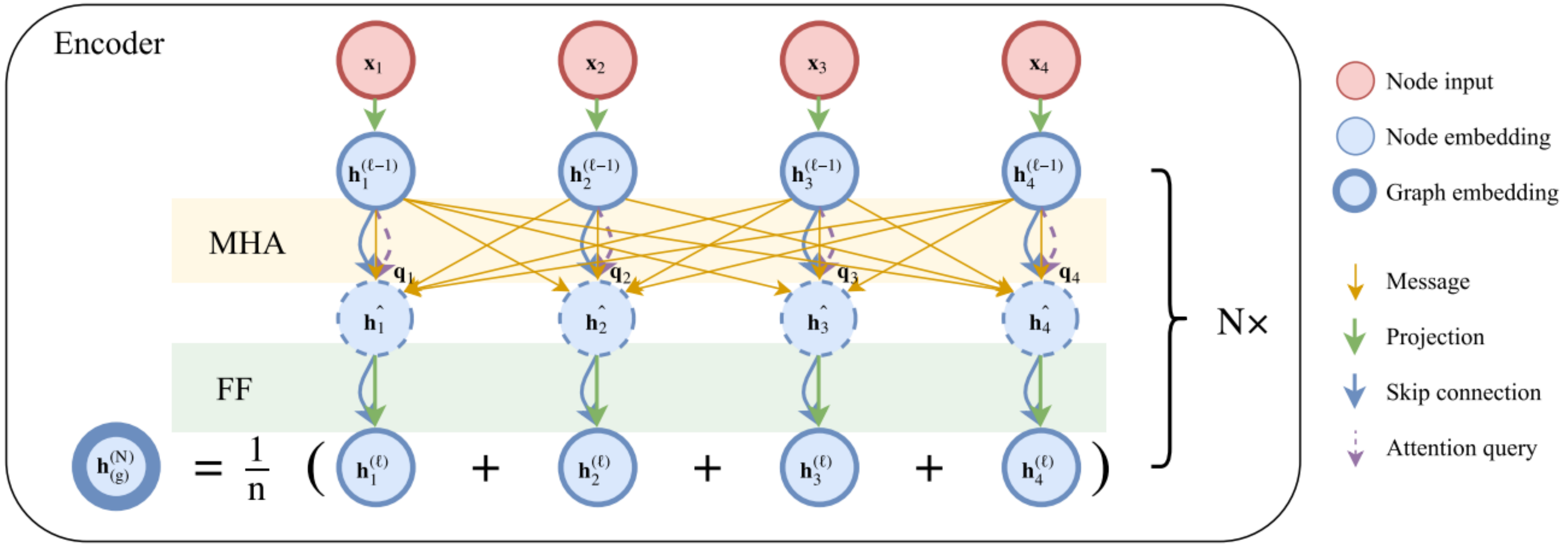
- A stochastic policy $p(\boldsymbol{\pi}|s)$

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

Chain Rule



Encoder



- Linear project the input $h_i^{(0)} = W^X x_i + b^X$
- $d_x = 2,\ d_h = 128,\ num\ heads\ = 8$
- Attention Layer.

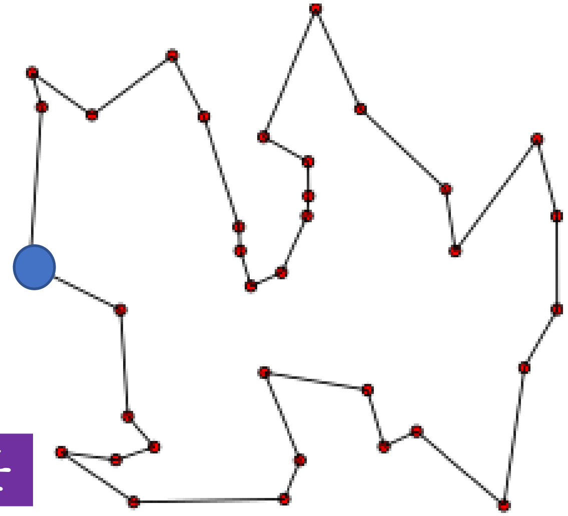
$$\hat{h}_i = \mathrm{BN}^l\left(h_i^{(l-1)} + \mathrm{MHA}_i^l\ \left(h_1^{(l-1)}, \cdots, h_n^{(l-1)}\right)\right)$$

$$h_i^{(l)} = \mathrm{BN}^l(\hat{h}_i\ + \mathrm{FF}^l(\hat{h}_i))$$

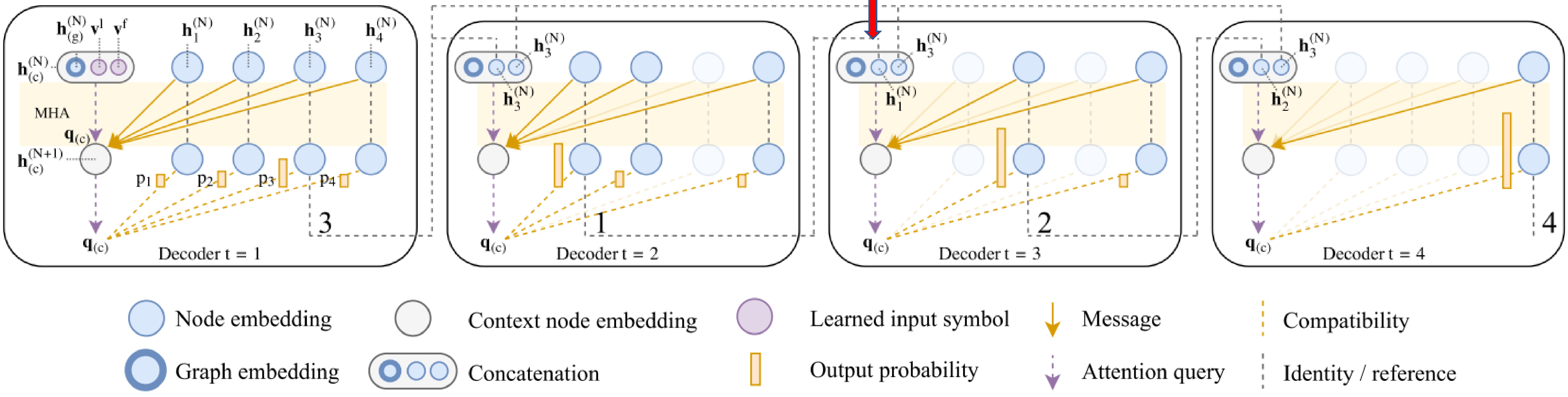
Decoder

- Context Query, Key, Value $\mathbf{q}_{(c)}, \mathbf{k}_i, \mathbf{v}_i = W^Q \mathbf{h}_{(c)}\,,\; W^K \mathbf{h}_i,\; W^V \mathbf{h}_i$

$$\mathbf{h}_c^{(N)} = \begin{cases} \left[\bar{h}^{(N)}, h_{\pi_{t-1}}^{(N)}, h_{\pi_1}^{(N)}\right], & t > 1 \\ \left[\bar{h}^{(N)}, v^l, v^f\right], & t = 1 \end{cases}$$



Graph	이전 방문	맨 처음 방문



Decoder

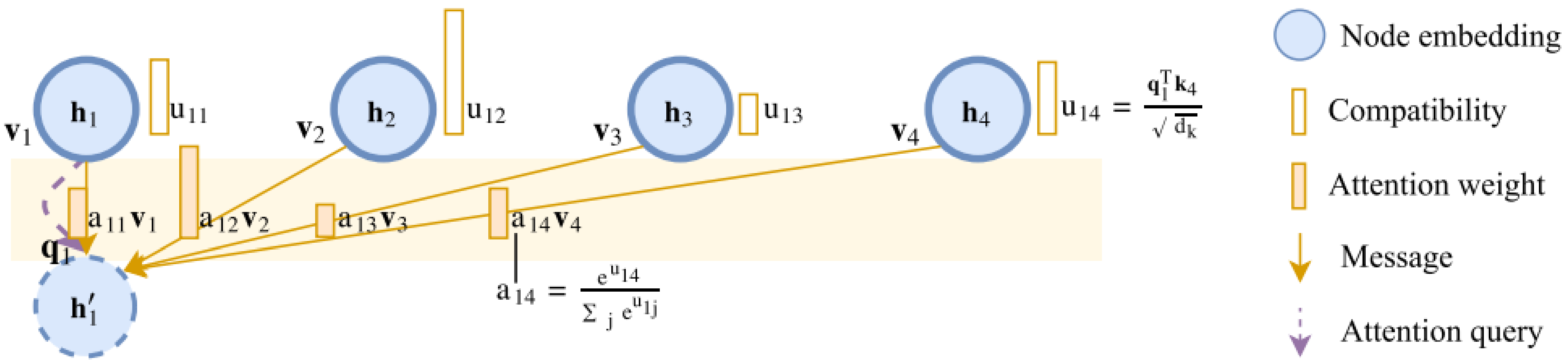


Figure 4: Illustration of weighted message passing using a dot-attention mechanism. Only computation of messages received by node 1 are shown for clarity. Best viewed in color.

$$u_{(c)j} = \begin{cases} C \cdot \tanh\left(\frac{q_{(c)}^T k_j}{\sqrt{d_k}}\right), if \ \ j \neq \pi_{t'}, \forall\, t' < t \\ -\infty, \qquad\qquad otherwise \end{cases} \quad \text{for clip } C \in [-10, 10]$$

$$\text{Final Probability} : p_i = p_\theta(\pi_{t=i} | s, \pi_{1:t-1}) = \frac{e^{u(c)j}}{\sum_j e^{u(c)j}}$$

RL and Baselines for RL

$$\nabla \mathcal{L}(\theta|s) = E_{p_\theta(\pi|s)}\left[\left(L(\pi) - \boldsymbol{b}(\boldsymbol{s})\right)\nabla \log p_\theta(\pi|s)\right]$$

Variance를 줄이기 위해서 사용

3 가지 선택지

1. **Exponential moving average** $b(s) = M$ with decay β.
   $M = L(\pi)$ in the first iteration and gets updated as $M \leftarrow \beta M + (1-\beta)L(\pi)$

2. **Value function** (Critic) $\hat{v}(s, w)$

3. **Rollout.** $\boldsymbol{b}(\boldsymbol{s})$ 를 현재 모델로부터 Greedy 하게 solution 선택

Code 1 - Decoding

```python
    def decode(self, probs, mask):
        if self.decode_type == "greedy":
            _, idxs = probs.max(1)
            assert not mask.gather(1, idxs.unsqueeze(-1)).data.any(), \
                "Decode greedy: infeasible action has maximum probability"
        elif self.decode_type == "sampling":
            idxs = probs.multinomial(1).squeeze(1)
            # Check if sampling went OK, can go wrong due to bug on GPU
            while mask.gather(1, idxs.unsqueeze(-1)).data.any():
                print(' [!] resampling due to race condition')
                idxs = probs.multinomial().squeeze(1)
        else:
            assert False, "Unknown decode type"

        return idxs
```

https://github.com/wouterkool/attention-learn-to-route

https://github.com/wouterkool/attention-learn-to-route/blob/master/nets/pointer_network.py

Code 2 – Forward (PointerNet)

```python
    def forward(self, decoder_input, embedded_inputs, hidden, context, eval_tours=None):

        outputs = []
        selections = []
        steps = range(embedded_inputs.size(0))

        for i in steps:
            hidden, log_p, probs, mask = self.recurrence(decoder_input, hidden, mask, idxs, i, context)
            # select the next inputs for the decoder [batch_size x hidden_dim]
            idxs = self.decode(
                probs,
                mask
            ) if eval_tours is None else eval_tours[:, i]

            idxs = idxs.detach()  # Otherwise pytorch complains it want's a reward, todo implement this more properly?

            # Gather input embedding of selected
            decoder_input = torch.gather(
                embedded_inputs,
                0,
                idxs.contiguous().view(1, batch_size, 1).expand(1, batch_size, *embedded_inputs.size()[2:])
            ).squeeze(0)

            # use outs to point to next object
            outputs.append(log_p)
            selections.append(idxs)
        return (torch.stack(outputs, 1), torch.stack(selections, 1)), hidden
```

- log_p : 강화학습용
- Probs : 다음 선택지를 위해서
- mask : 이전에 선택한 것 제외

Attention model의 경우
추가적인 embedder!

```python
self.embedder = GraphAttentionEncoder(
    n_heads=n_heads,
    embed_dim=embedding_dim,
    n_layers=self.n_encode_layers,
    normalization=normalization
)
```

Code 3 – Batch Train

```python
def train_batch(
        model,
        optimizer,
        baseline,
        epoch,
        batch_id,
        step,
        batch,
        tb_logger,
        opts
):
    x, bl_val = baseline.unwrap_batch(batch)
    x = move_to(x, opts.device)
    bl_val = move_to(bl_val, opts.device) if bl_val is not None else None

    # Evaluate model, get costs and log probabilities
    cost, log_likelihood = model(x)

    # Evaluate baseline, get baseline loss if any (only for critic)
    bl_val, bl_loss = baseline.eval(x, cost) if bl_val is None else (bl_val, 0)

    # Calculate loss
    reinforce_loss = ((cost - bl_val) * log_likelihood).mean()
    loss = reinforce_loss + bl_loss

    # Perform backward pass and optimization step
    optimizer.zero_grad()
    loss.backward()
    # Clip gradient norms and get (clipped) gradient norms for logging
    grad_norms = clip_grad_norms(optimizer.param_groups, opts.max_grad_norm)
    optimizer.step()
```

강화학습 안정성을 위한 Baseline

REINFORCE with Rollout Baseline

Algorithm 1 REINFORCE with Rollout Baseline

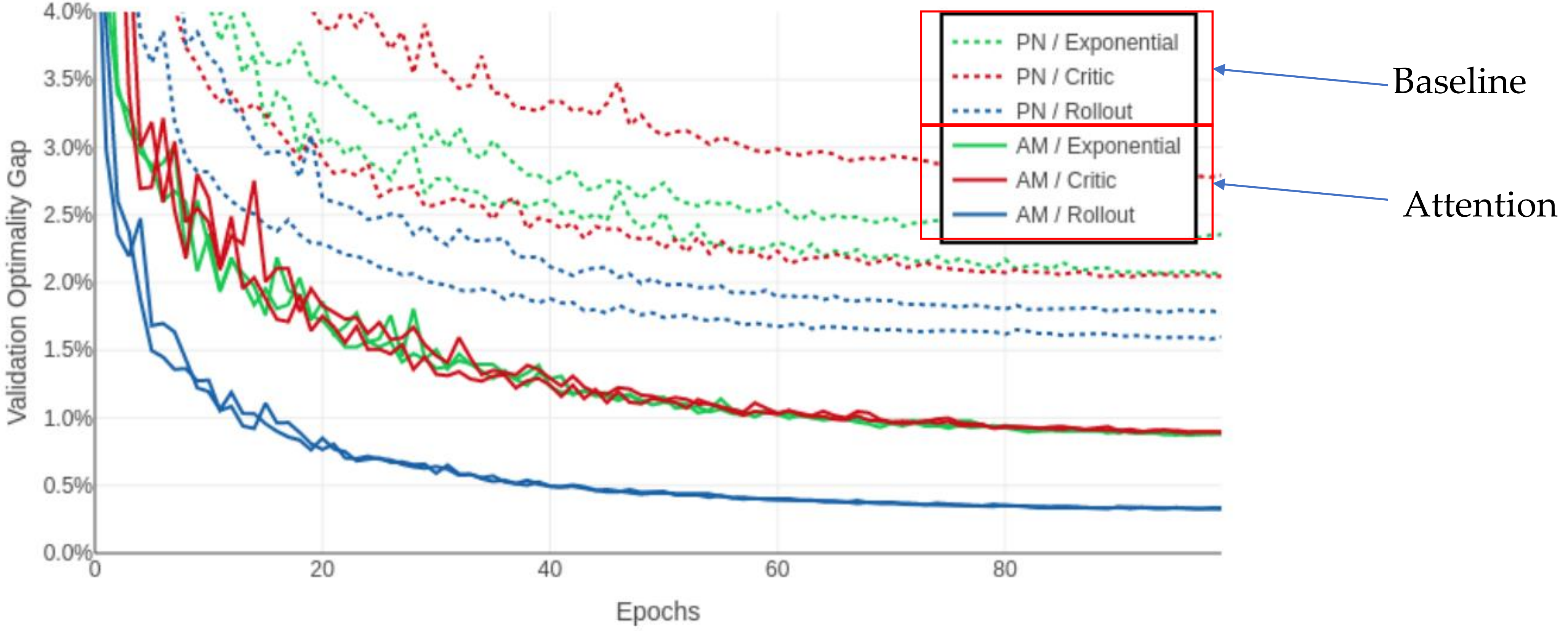
Input: number of epochs E, steps per epoch T, batch size B, significance α
Init $\boldsymbol{\theta}$, $\boldsymbol{\theta}^{\mathrm{BL}} \leftarrow \boldsymbol{\theta}$
for epoch $= 1, \dots, E$ **do**
    for step $= 1, \dots, T$ **do**
        $s_i \leftarrow \mathrm{RandomInstance}()\ \forall i \in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

Stochastic Policy π^{BL} 로부터 Greedy

학습이 충분히 의미있으면 θ^{BL} 업데이트

at the end of every epoch, replace the parameters θ^{BL} of the baseline policy only **if the improvement is significant** according to a **paired t-test (α = 5%), on 10000 separate (evaluation) instances.**

Baselines for RL



- **Assumption** The difficulty of an instance can (on average) be estimated by the performance of an algorithm applied to it

Experiments

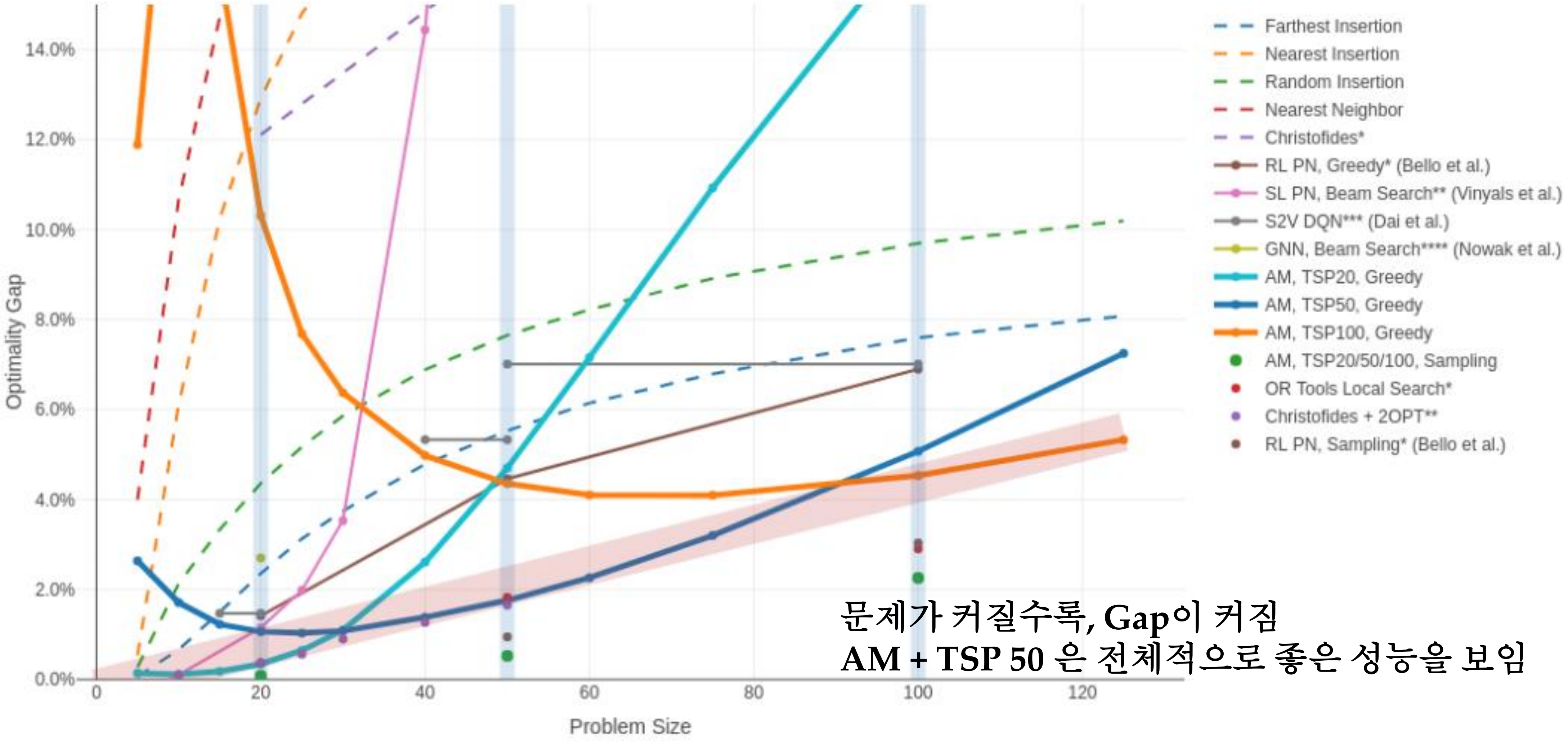
결론 :
1. 높은 성능
2. **Solution output 시간이 오래 안 걸림**

- Training data로부터 100 epoch 학습
- 10000 개의 테스트 데이터
- 매 스텝마다, greedy 하게 가장 확률이 높은 point 선택

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

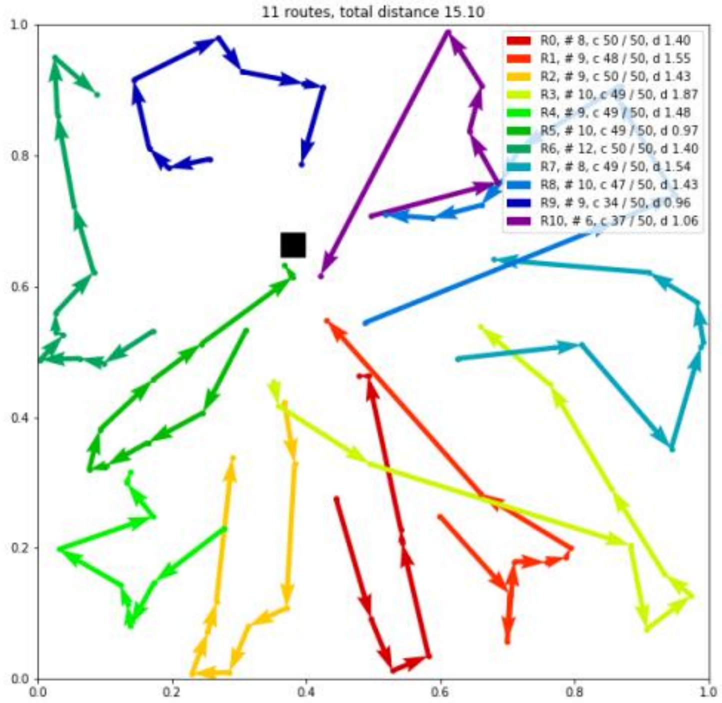
	Method	$n = 20$ Obj.	Gap	Time	$n = 50$ Obj.	Gap	Time	$n = 100$ Obj.	Gap	Time	
TSP	Concorde	3.84	0.00%	(1m)	5.70	0.00%	(2m)	7.76	0.00%	(3m)	Optimal
	LKH3	3.84	0.00%	(18s)	5.70	0.00%	(5m)	7.76	0.00%	(21m)	
	Gurobi	3.84	0.00%	(7s)	5.70	0.00%	(2m)	7.76	0.00%	(17m)	
	Gurobi (1s)	3.84	0.00%	(8s)	5.70	0.00%	(2m)		-		
	Nearest Insertion	4.33	12.91%	(1s)	6.78	19.03%	(2s)	9.46	21.82%	(6s)	Greedy
	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%		Solution을 1280 개 Sampling
	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)	

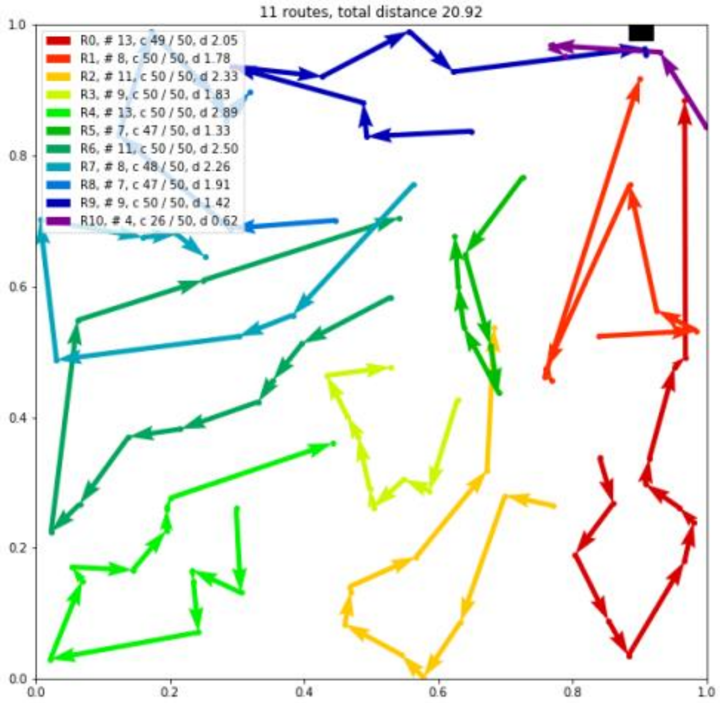
Generalization



문제가 커질수록, Gap이 커짐
AM + TSP 50 은 전체적으로 좋은 성능을 보임

Another Permutation Problem





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)
	RL (beam 10)	6.40	4.92%		11.15	7.46%		16.96	8.39%	
	Random CW	6.81	11.64%		12.25	18.07%		18.96	21.18%	
	Random Sweep	7.08	16.07%		12.96	24.91%		20.33	29.93%	
	OR Tools	6.43	5.41%		11.31	9.01%		17.16	9.67%	
	AM (sampling)	**6.25**	**2.49%**	(6m)	**10.62**	**2.40%**	(28m)	**16.23**	**3.72%**	(2h)

Conclusion

- This model can be applied to any permutation based combinatorial optimization problem
(순열 조합 문제를 순차적으로 풀 수 있다.)

$$(\pi_1, \cdots, \pi_n)$$

- Combinatorial Problem using Neural network for searching solution in a better way
(강화학습의 장점 → Sequential 문제 해결)

- Generalization doesn't work well.