Preference Optimization for Combinatorial Optimization Problems

2025. 07. 04

임제원

I Introduction

✓ 지도학습, 강화학습을 포함하는 딥러닝은 데이터로부터 직접 휴리스틱을 학습하며 조합최적화를 해결

✓ 지도학습은 **고품질의 해를 요구**하며, 훈련 데이터 세트가 최적성을 보장하지 못할 수 있음

✓ 강화학습은 환경으로부터 보상 및 피드백을 획득하여 모델이 **시행착오 방식으로 개선**되지만 아래와 같은 어려움 존재

1. **Diminishing reward signals**
   : 정책이 개선되며 advantage value가 감소 -> vanishing gradients 및 느린 수렴
2. **Unconstrained action spaces**
   : 조합최적화 문제에서 제약이 없어, 행동공간이 매우 크므로 탐색이 어렵고 계산량이 너무 큼
3. **Additional inference time**
   : 빠르게 해를 추론하지만, 후처리 필요(local search 등) 이 과정에서 추가적인 계산 비용 발생

✓ Diminishing reward signals 및 비효율적인 탐색문제를 해결하기 위해 **양적 보상 신호 → 질적 보상 신호**로 변환

✓ 절대적인 보상 값보다는 생성된 **해 간의 우월성**에 초점을 맞추는 것을 제안

→ 이 접근 방식은 학습 과정을 안정화시키며 선호도 신호가 보상의 규모에 둔감하여 이론적으로 최적성을 강조

✓ 추가적으로 inference time을 완화하기 위해 이러한 기술을 fine-tuning 과정에 통합하여 개선된 솔루션 학습

Ⅱ Contribution

1. Preference-based Framework

: 정량적 보상 신호를 정성적 선호 신호로 변환하는 새로운 학습 프레임워크 제안

ex) reward가 A(500), B(498)이라면 수치로는 큰 차이가 없지만, A>B 라는 선호관계를 활용하여 안정된 학습 가능

reward scale에 의존하지 않아, 다양한 문제에서 robust한 학습이 가능함

2. Reparameterized Entropy-Regularized Objective

: reward 함수를 정책의 함수로 재매개변수화 하여 선호 신호와 직접 정렬되는 새로운 목적함수를 정의

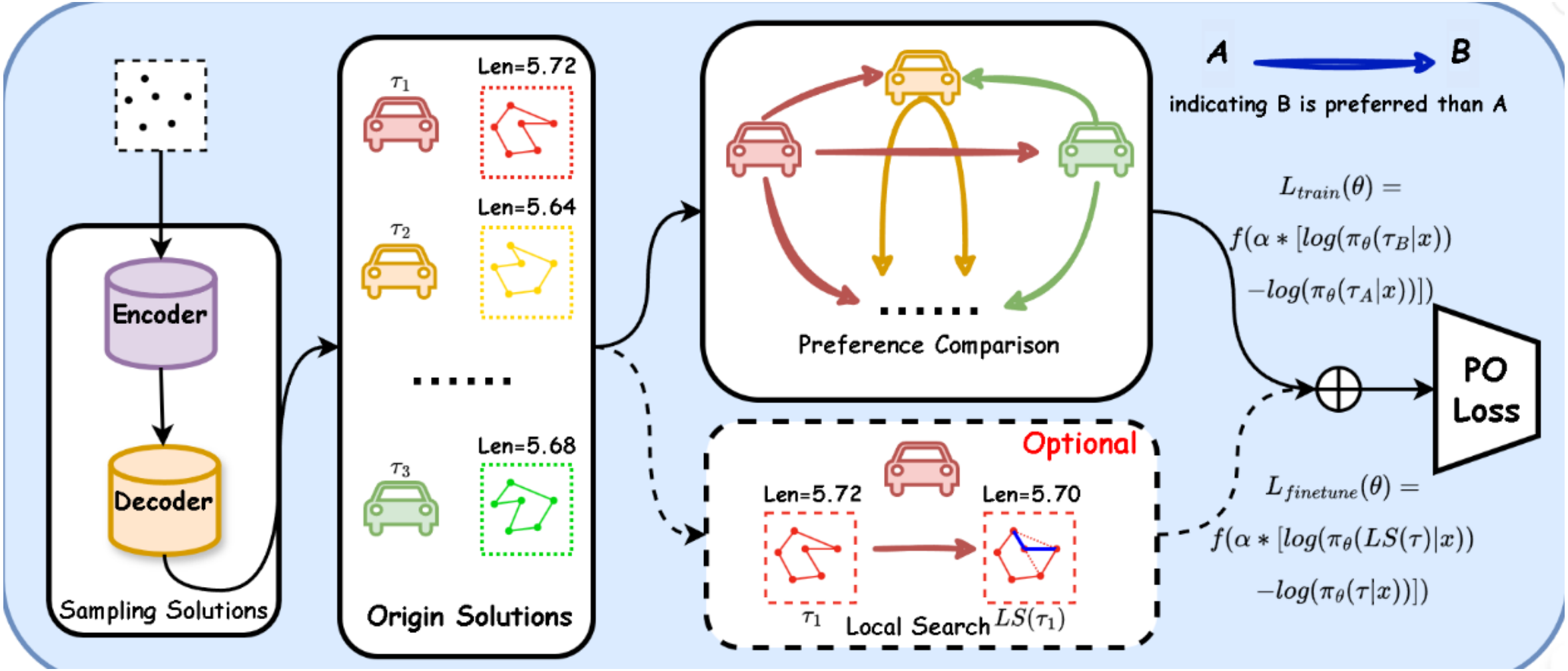
ex) 1번에서의 A > B 정보를 활용하여 A가 B보다 나은 행동을 했다면 $\log \sigma \left(\alpha(\log \pi(A) - \log \pi(B))\right)$로

A의 확률이 더 높아지도록 학습 $(\sigma: \text{sigmoid}, \pi(\text{A}): \text{ probability of A})$

3. Integration with Local Search

: Local Search를 후처리가 아니라, 훈련 과정에 통합하여 사용.

전체 모델 구조


Encoder
Decoder
Sampling Solutions
Len=5.72
Len=5.64
Len=5.68
Origin Solutions
Preference Comparison
A
B
indicating B is preferred than A
$L_{train}(\theta) =$
$f(\alpha * [log(\pi_\theta(\tau_B|x))$
$-log(\pi_\theta(\tau_A|x))])$
PO Loss
Optional
Len=5.72
Len=5.70
Local Search
$LS(\tau_1)$
$L_{finetune}(\theta) =$
$f(\alpha * [log(\pi_\theta(LS(\tau)|x))$
$-log(\pi_\theta(\tau|x))])$

❖ **Entropy-Regularized Optimal Policy**

$$\pi^*(\tau|x) = \frac{1}{Z(x)} \exp\left(\frac{1}{\alpha} r(x,\tau)\right)$$

τ: 하나의 솔루션 (e.g., 하나의 TSP 투어 경로)

$r(x,\tau)$: 해당 솔루션의 보상 (e.g., -투어 길이)

α: entropy 조절 계수 (작을수록 exploitation, 클수록 exploration)

$Z(x)$: 정규화 상수 (partition function), 전체 trajectory의 확률이 1이 되도록 함

✓ Maximum Entropy Reinforcement Learning 프레임워크에서 유도됨

✓ 기존 정책 학습 목적함수에 entropy 항을 추가하여 탐색을 유도함

→ 좋은 솔루션에 더 높은 확률을 부여하되, 일정한 entropy를 유지하여 다양한 경로도 탐색하게끔 유도하는 정책

❖ **Preference Probability via Policy Difference**

$$p(\tau_1 \succ \tau_2|x) = \sigma\left(\alpha\left[\log \pi(\tau_1|x) - \log \pi(\tau_2|x)\right]\right)$$

$\pi(\tau_1|x), \pi(\tau_2|x)$: 각각 솔루션 τ_1, τ_2의 정책 확률

$\log \pi(\tau)$: soft log-probability — reward function과 비례한 값

$\sigma(z) = \frac{1}{1+e^{-z}}$: 시그모이드 함수 (확률화)

✓ Bradley-Terry (BT) 모델 기반의 쌍 선호 모델

✓ 누가 더 나은가? 를 정성적 비교로 모델링

→ POCO 논문의 핵심은 이 선호되는 trajectory 쪽의 확률을 최대화 하도록 정책을 학습시킨 것

Algorithm 1 Preference Optimization for COPs.

Input: problem set $\mathcal{D}$, number of training steps T, fine-tune steps $T_{\text{FT}} \geq 0$,batch size B, learning rate η, ground truth reward function r, number of local search iteration I_{LS}, initialized policy π_θ.

for $step = 1, \ldots, T + T_{\text{FT}}$ **do**

//Sampling N solutions for each instance x_i

$x_i \leftarrow \mathcal{D}, \quad \forall i \in \{1, \ldots, B\}$

$\tau_i = \{\tau_i^1, \tau_i^2, \ldots, \tau_i^N\} \leftarrow \pi_\theta(x_i), \ \forall i \in \{1, \ldots, B\}$

// Fine-tuning with LS for T_{FT} steps **(Optional)**

if $step > T$ **then**

$\{\hat{\tau}_i^1, \hat{\tau}_i^2, \ldots, \hat{\tau}_i^N\} \leftarrow$ *LocalSearch*$(\tau_i, r, I_{\text{LS}}), \ \forall i$

$\tau_i \leftarrow \tau_i \cup \{\hat{\tau}_i^1, \hat{\tau}_i^2, \ldots, \hat{\tau}_i^N\}$

end if

//Calculate conflict-free preference labels via grounding reward function $r(x, \tau)$

$y_{j,k}^i \leftarrow \mathbb{1}\left(r(x_i, \tau_i^j) > r(x_i, \tau_i^k)\right), \ \forall j, k$

//Approximating the gradient according to Eq. 8

$$\nabla_\theta J(\theta) \leftarrow \frac{\alpha}{B|\tau_i|^2} \sum_{i=1}^{B} \sum_{j=1,k=1}^{|\tau_i|} \Big[\Big(g(\tau_i^j, \tau_i^k, x_i) - g(\tau_i^k, \tau_i^j, x_i)\Big) \nabla_\theta \log \pi_\theta(\tau_i^j \mid x_i)\Big]$$

$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$

end for

Ⅲ Methodology - Example

✓ 2개의 시퀀스 τ_1, τ_2가 존재한다고 가정

✓ 이 두 경로에 대해, 보상 값이 "$\boldsymbol{\tau_1 > \tau_2}$" 즉, τ_1이 더 좋음

✓ $\boldsymbol{\pi(\tau_1|x) = 0.3,\ \pi(\tau_2|x) = 0.4}$로 모델은 오히려 **안 좋은 $\boldsymbol{\tau_2}$를 더 높은 확률로 선택**하고 있는 상황

✓ $p(\tau_1 > \tau_2 \mid x) = \sigma(\alpha[\log\pi(\tau_1|x) - \log\pi(\tau_2|x)]$

$= \sigma(\alpha[\log 0.3 - \log 0.4]) \approx 0.36$, 즉, "$\tau_1 > \tau_2$" 의 확률을 36%정도로 보고 있음 → 잘못된 판단

✓ Loss 계산

: $L_{pref} = -[y \log p + (1-y)\log(1-p)] = -\log(0.36) \approx 1.02$ → 모델은 현재 예측이 잘못되었다는 신호를 받음

✓ 이후 Loss가 0에 수렴하도록 모델 업데이트 후 $\pi(\tau_1|x) = 0.45,\ \pi(\tau_2|x) = 0.35$ (예시) 로 수정

Methodology - Code

```
------------------------------------------------------------
각 TSP 문제 인스턴스 xi에 대해 POMO로 N=4개 솔루션 생성
생성된 솔루션들:
  pomo0: 0→3→1→4→2→0 → 거리: 10.5, 보상: -10.5
  pomo1: 0→1→3→2→4→0 → 거리: 8.2, 보상: -8.2
  pomo2: 0→4→2→1→3→0 → 거리: 12.1, 보상: -12.1
  pomo3: 0→2→4→3→1→0 → 거리: 9.8, 보상: -9.8
```

✓ POMO Loss Function

```python
def rl_loss_fn(self, reward, prob_list):
    advantage = reward - reward.float().mean(dim=1, keepdims=True)
    # shape: (batch, pomo)
    log_prob = prob_list.log().sum(dim=2)
    # size = (batch, pomo)
    loss = -advantage * log_prob  # Minus Sign: To Increase REWARD
    # shape: (batch, pomo)
    loss_mean = loss.mean()

    return loss_mean
```

✓ 입력 보상: [-10.5, -8.2, -12.1, -9.8]

✓ Baseline: -10.15 (평균)

✓ Advantage: [-0.35, +1.95, -1.95, +0.35]

✓ 양수 → 평균보다 좋음 → 확률 증가

✓ 음수 → 평균보다 나쁨 → 확률 감소

Methodology - Code

```
------------------------------------------------------------
각 TSP 문제 인스턴스 xi에 대해 POMO로 N=4개 솔루션 생성
생성된 솔루션들:
  pomo0: 0→3→1→4→2→0 → 거리: 10.5, 보상: -10.5
  pomo1: 0→1→3→2→4→0 → 거리: 8.2, 보상: -8.2
  pomo2: 0→4→2→1→3→0 → 거리: 12.1, 보상: -12.1
  pomo3: 0→2→4→3→1→0 → 거리: 9.8, 보상: -9.8
```

✓ POCO Loss Function

```python
def preference_among_pomo_loss_fn(self, reward, prob): # BT
    preference = reward[:, :, None] > reward[:, None, :]
    # shape: (batch, pomo, pomo)

    log_prob = torch.log(prob).sum(2)
    log_prob_pair = log_prob[:, :, None] - log_prob[:, None, :]
    pf_log = torch.log(F.sigmoid(self.alpha * log_prob_pair))
    loss = -torch.mean(pf_log * preference)

    return loss
```

```
Preference Matrix (행이 열보다 좋으면 1):
      p0  p1  p2  p3
 p0:   0   0   1   0
 p1:   1   0   1   1
 p2:   0   0   0   0
 p3:   1   0   1   0

해석:
  - pomo1이 다른 모든 pomo보다 좋음 (1행에 1이 3개)
  - pomo2가 다른 모든 pomo보다 나쁨 (2행에 1이 0개)
```

Ⅲ Methodology - Code

```
================================================================================
PO LOSS GRADIENT 계산 과정 상세 설명
================================================================================
현재 상황:
  성능 순위: pomo1 > pomo3 > pomo0 > pomo2
  확률 순위: pomo1 > pomo0 > pomo3 > pomo2
  문제: pomo3가 pomo0보다 성능 좋은데 확률이 낮음

STEP 1: 전체 쌍별 비교 (4x4 = 16개 쌍)
------------------------------------------------
각 쌍에 대해 (예상 승률 - 실제 승률)을 계산

모든 쌍별 계산:
  pomo0 vs pomo1: 져야함, 예상0.490 → pomo0 감소 (-0.490)
  pomo0 vs pomo2: 이겨야함, 예상0.528 → pomo0 증가 (+0.472)
  pomo0 vs pomo3: 져야함, 예상0.512 → pomo0 감소 (-0.512)
  pomo1 vs pomo0: 이겨야함, 예상0.510 → pomo1 증가 (+0.490)
  pomo1 vs pomo2: 이겨야함, 예상0.538 → pomo1 증가 (+0.462)
  pomo1 vs pomo3: 이겨야함, 예상0.522 → pomo1 증가 (+0.478)
  pomo2 vs pomo0: 져야함, 예상0.472 → pomo2 감소 (-0.472)
  pomo2 vs pomo1: 져야함, 예상0.462 → pomo2 감소 (-0.462)
  pomo2 vs pomo3: 져야함, 예상0.484 → pomo2 감소 (-0.484)
  pomo3 vs pomo0: 이겨야함, 예상0.488 → pomo3 증가 (+0.512)
  pomo3 vs pomo1: 져야함, 예상0.478 → pomo3 감소 (-0.478)
  pomo3 vs pomo2: 이겨야함, 예상0.516 → pomo3 증가 (+0.484)

  Bradley-Terry 모델로 현재 예상 승률 계산
------------------------------------------------------------
g(τj, τk, xi) = sigmoid(α × (log π(τj|xi) - log π(τk|xi)))
현재 모델이 예상하는 승률 (α=0.05):
        p0    p1    p2    p3
  p0: 0.500 0.490 0.528 0.512
  p1: 0.510 0.500 0.538 0.522
  p2: 0.472 0.462 0.500 0.484
  p3: 0.488 0.478 0.516 0.500
```

```
STEP 2: 각 pomo별 총 gradient 효과
------------------------------------------------
pomo0 총 gradient: -0.530
  이겨야 할 경우들:
    vs pomo2: +0.472
  져야 할 경우들:
    vs pomo1: -0.490
    vs pomo3: -0.512

pomo1 총 gradient: +1.429
  이겨야 할 경우들:
    vs pomo0: +0.490
    vs pomo2: +0.462
    vs pomo3: +0.478

pomo2 총 gradient: -1.418
  져야 할 경우들:
    vs pomo0: -0.472
    vs pomo1: -0.462
    vs pomo3: -0.484

pomo3 총 gradient: +0.519
  이겨야 할 경우들:
    vs pomo0: +0.512
    vs pomo2: +0.484
  져야 할 경우들:
    vs pomo1: -0.478
------------------------------------------------------
pomo0 분석:
  이겨야 함: pomo2 (예상99.9% vs 실제100%) → +0.001
  져야 함: pomo1 (예상8.1% vs 실제0%) → -0.081
  져야 함: pomo3 (예상95.0% vs 실제0%) → -0.950
  총합: +0.001 - 0.081 - 0.950 = -1.030
  → pomo0 확률 감소!

pomo3 분석:
  이겨야 함: pomo0 (예상5.0% vs 실제100%) → +0.950
  이겨야 함: pomo2 (예상97.7% vs 실제100%) → +0.023
  져야 함: pomo1 (예상0.5% vs 실제0%) → -0.005
  총합: +0.950 + 0.023 - 0.005 = +0.968
  → pomo3 확률 대폭 증가!
```

```
업데이트 전 확률:
  pomo0: 0.268
  pomo1: 0.603
  pomo2: 0.029
  pomo3: 0.100

업데이트 후 확률:
  pomo0: 0.303 (↑0.035)
  pomo1: 0.560 (↓0.043)
  pomo2: 0.035 (↑0.007)
  pomo3: 0.102 (↑0.002)
```

Table 1: Experiment results on TSP and CVRP. Gap is evaluated on 10k instances and Times are summation of them.

	Solver	Algorithm	TSP						CVRP					
			$N = 20$		$N = 50$		$N = 100$		$N = 20$		$N = 50$		$N = 100$	
			Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time	Gap	Time
Heuristic	Concorde	-	0.00%	13m	0.00%	21.5m	0.00%	1.2h	-	-	-	-	-	-
	LKH3	-	0.00%	28s	0.00%	4.3m	0.00%	15.6m	0.09%	0.5h	0.18%	2h	0.55%	4h
	HGS	-	-	-	-	-	-	-	0.00%	1h	0.00%	3h	0.00%	5h
Neural Solvers	AM (Kool et al., 2019)	RF	0.28%	0.1s	1.66%	1s	3.40%	2s	4.40%	0.1s	6.02%	1s	7.69%	3s
		PO	0.33%	0.1s	1.56%	1s	2.86%	2s	4.60%	0.1s	5.65%	1s	6.82%	3s
	Pointerformer (Jin et al., 2023)	RF	**0.00%**	6s	0.02%	12s	0.15%	1m	-	-	-	-	-	-
		PO	**0.00%**	6s	0.01%	12s	0.06%	1m	-	-	-	-	-	-
	Sym-NCO (Kim et al., 2022)	RF	0.01%	1s	0.16%	2s	0.39%	8s	0.72%	1s	1.31%	4s	2.07%	16s
		PO	**0.00%**	1s	0.08%	2s	0.28%	8s	0.63%	1s	1.20%	4s	1.88%	16s
	POMO (Kwon et al., 2020)	RF	0.01%	1s	0.04%	15s	0.15%	1m	0.37%	1s	0.94%	5s	1.76%	3.3m
		PO	**0.00%**	1s	0.02%	15s	0.07%	1m	0.16%	1s	0.68%	5s	1.37%	3.3m
		PO+Finetune	**0.00%**	1s	**0.00%**	15s	**0.03%**	1m	**0.08%**	1s	**0.53%**	5s	**1.19%**	3.3m

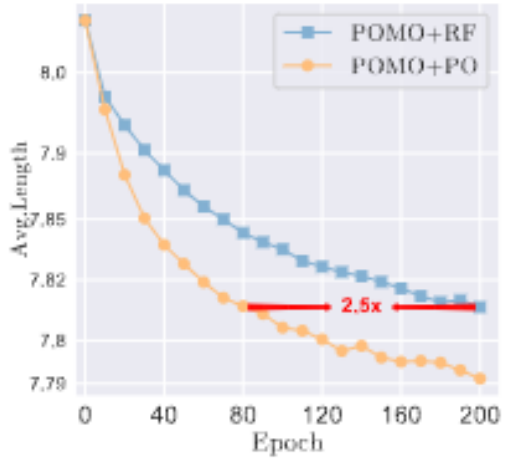
Experiment

	Solver	FFSP20			FFSP50			FFSP100		
		MS. ↓	Gap	Time	MS. ↓	Gap	Time	MS. ↓	Gap	Time
Heuristic	CPLEX (60s)*	46.4	84.13%	17h		×			×	
	CPLEX (600s)*	36.6	45.24%	167h		×			×	
	Random	47.8	89.68%	1m	93.2	88.28%	2m	167.2	87.42%	3m
	Shortest Job First	31.3	24.21%	40s	57.0	15.15%	1m	99.3	11.33%	2m
	Genetic Algorithm	30.6	21.43%	7h	56.4	13.94%	16h	98.7	10.65%	29h
	Particle Swarm Opt.	29.1	15.48%	13h	55.1	11.31%	26h	97.3	9.09%	48h
Neural Solver	MatNet (RF)	27.3	8.33%	8s	51.5	4.04%	14s	91.5	2.58%	27s
	MatNet (PO)	27.0	7.14%	8s	51.3	3.64%	14s	91.1	2.13%	27s
	MatNet (RF+Aug.)	25.4	0.79%	3m	49.6	0.20%	8m	89.7	0.56%	23m
	MatNet (PO+Aug.)	**25.2**	**0.00%**	3m	**49.5**	**0.00%**	8m	**89.2**	**0.00%**	23m

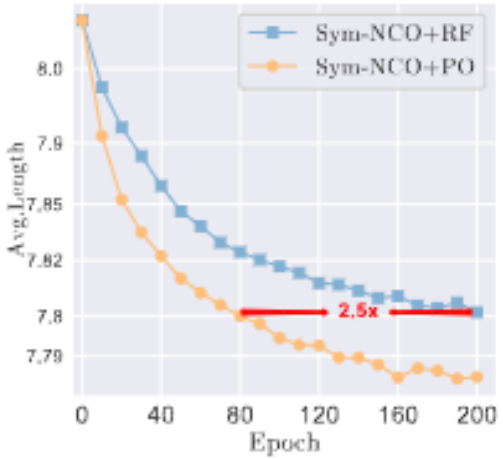
Experiment

Solver	Paradigm	TSPLib				CVRPLib			
		(0, 200] Gap	(200, 1002] Gap	Total Gap	Time	(0, 200] Gap	(200, 1000] Gap	Total Gap	Time
LKH3	Heuristic	0.00%	0.00%	0.00%	24s	0.36%	1.18%	1.00%	16m
HGS	Heuristic	-	-	-	-	0.01%	0.13%	0.11%	16m
NeuroLKH (Xin et al., 2021b)	Heuristic+SL	0.00%	0.00%	0.00%	24s	0.47%	1.16%	0.88%	16m
POMO (Kwon et al., 2020)	Neural Solver+RL	3.07%	13.35%	7.45%	0.41s	5.26%	11.82%	10.37%	0.80s
Sym-NCO (Kim et al., 2022)		2.88%	15.35%	8.29%	0.34s	9.99%	27.09%	23.32%	0.87s
Omni-POMO (Zhou et al., 2023)		1.74%	7.47%	4.16%	0.34s	5.04%	6.95%	6.52%	0.75s
Pointerformer (Jin et al., 2023)		2.31%	11.47%	6.32%	0.24s	-	-	-	-
LEHD (Luo et al., 2023)	Neural Solver+SL	2.03%	3.12%	2.50%	1.28s	11.11%	12.73%	12.25%	1.67s
BQ-NCO (Drakulic et al., 2023)		1.62%	**2.39%**	**2.22%**	2.85s	10.60%	10.97%	10.89%	3.36s
DIFUSCO (Sun & Yang, 2023)		1.84%	10.83%	5.77%	30.68s	-	-	-	-
T2TCO (Li et al., 2023)		1.87%	9.72%	5.30%	30.82s	-	-	-	-
ELG (RF) (Gao et al., 2024)	Neural Solver+RL	1.12%	5.90%	3.08%	0.63s	4.51%	6.46%	6.03%	1.90s
ELG (PO)		**1.04%**	5.84%	3.00%	0.63s	**4.39%**	**6.37%**	**5.94%**	1.90s

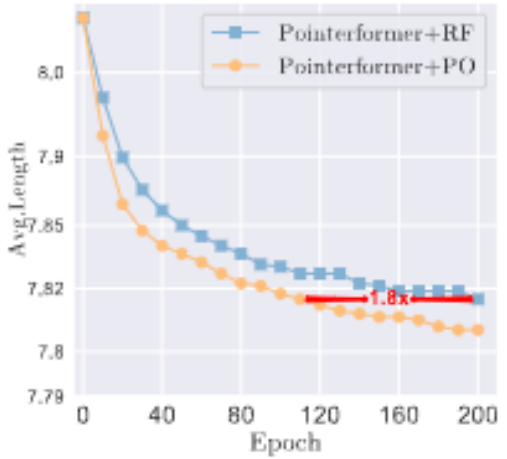
Experiment



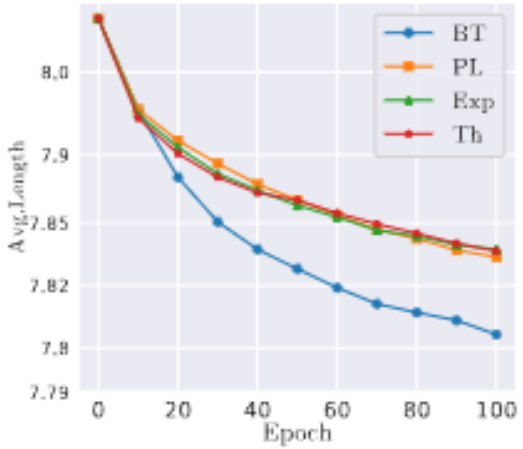
(a) POMO



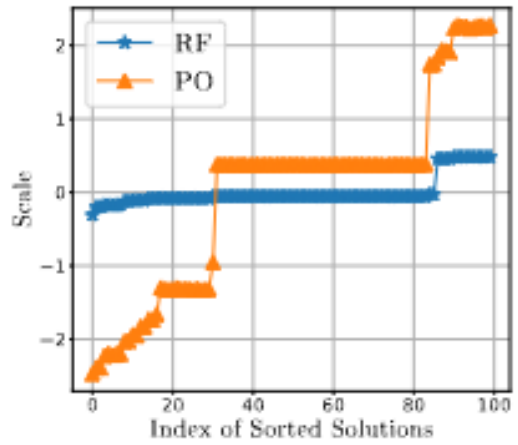
(b) Sym-NCO



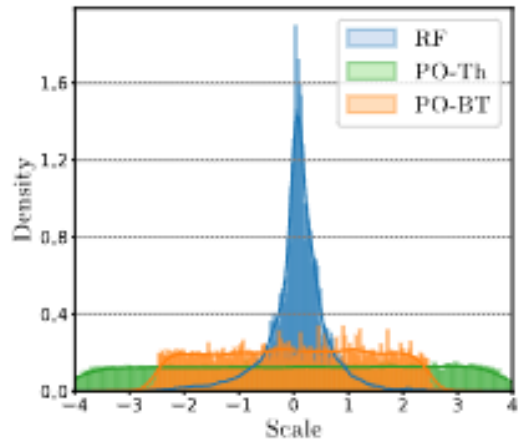
(c) Pointerformer



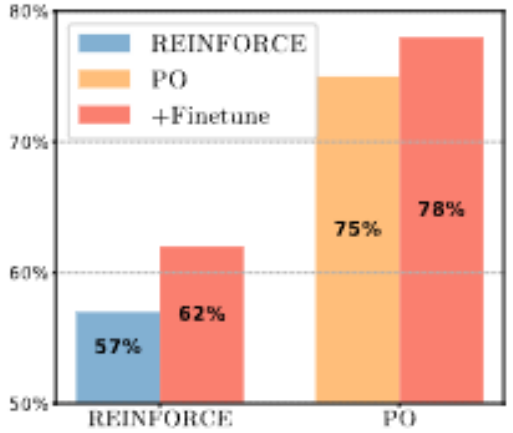
(d) Preference Models



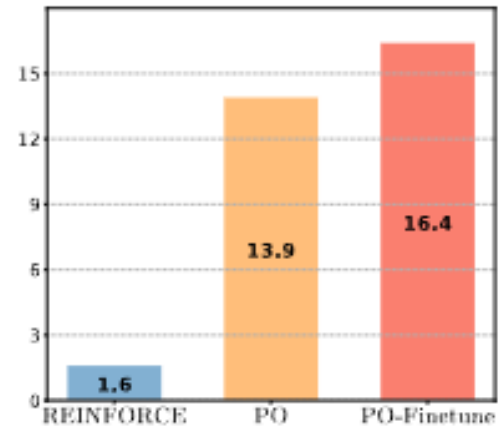
(a) Advantage Separation



(b) Advantage Scales



(c) Consistency of Policies



(d) Trajectory Entropy

V Conclusion

- ✓ POCO는 기존 강화학습 기반 조합최적화의 한계였던 보상 신호 축소 문제와 탐색 비효율성을 해결하고자, 정량적 보상을 정성적 선호 신호로 변환하는 새로운 학습 프레임워크를 제안함.
- ✓ 이 방식은 보상 스케일에 영향을 받지 않으면서도 항상 더 나은 솔루션을 강조하도록 정책을 안정적으로 학습시킴.
- ✓ 로컬 서치 기법을 학습 과정에 통합함으로써 추론 시간을 증가시키지 않으면서도 해의 품질을 향상시킬 수 있도록 설계함.
- ✓ TSP, CVRP, FFSP 등 다양한 문제에서 기존 RL보다 더 빠르게 수렴하고 더 높은 품질의 해를 도출함을 실험적으로 입증함.
- ✓ POCO는 향후, 보상 설계가 어려운 문제나 다목적 최적화 문제에도 확장 가능성이 높은 일반적인 최적화 프레임워크로 활용될 수 있음.