BOPO: Neural Combinatorial Optimization viaBest-anchored and Objective-guided Preference Optimization

2025. 07. 22

황규상

I Introduction

✓ 기존의 강화 학습(Reinforcement Learning, RL)기반 방법은 희소한 보상과 활용되지 않는 솔루션으로 낮은 효율성문제를 겪음.

✓ 이에 본 논문에서는 목적 함수 값을 통해 solution preference를 활용하는 훈련 패러다임인 Best-anchored and Objective-guided Prefer-ence Optimization (BOPO)를 제안

(1) 더 나은 탐색 및 활용 솔루션을 위한 best-anchored 선호도 쌍 구성

(2) 보상 모델 또는 참조 정책에 대한 의존성을 제거하여 COPs의 목적 함수 차이를 통해 적응적으로 기울기를 조정하는 objective-guided pairwise 손실 함수를 도입

✓ ob-shop Scheduling Problem (JSP), Traveling Salesman Problem (TSP) 및 Flexible Job-shop Scheduling Problem (FJSP)에 대한 실험 결과 BOPO의 우수성 인증

✓ BOPO는 아키텍처에 구애 받지않아 기존 NCO모델과 원활한 통합가능

Methodology

1. Hybrid Rollout

: greedy rollout과 sampling rollout의 상호 보완적인 강점을 활용하기 위해 두 가지 접근 방식을 결합한 전략

Sampling에서 B-1개, greedy 에서 1개를 포함하여 B개의 솔루션 생성

2. Uniform Filtering

: 모든 B개의 솔루션을 사용하여 선호도 쌍을 구성하면 BC2쌍의 조합이 생성되어 높은 계산 비용과 많은 저품질 쌍이 발생

->Uniform Filtering을 사용하여 솔루션 선택하여 유사한 솔루션 클러스터에 과적합되는 것을 방지

구체적으로, 솔루션을 최적순대로 solutions $S = \{y'1 > \cdots > y'B\}$ 와 같이 나열 후 C={y1>y2>⋯>yK} 와 같이 k개만 골라 사용한다. 여기서 골라내는 방식은 $yk = y'\lfloor B/K \rfloor(k-1)+1, \forall k \in \{1, \cdots, K\}$ 와 같다

Ex) B=100, K=5 라면 → 인덱스는 1, 21, 41, 61, 81

3. Best-anchored Pairing

: COP는 최적의 솔루션을 찾는 데만 집중하므로 고품질 예제를 통한 학습이 우선시됨

➔ K개의 솔루션 $\{y1 \succ \cdots \succ yK\}$에 대해 최적의 솔루션과 차선의 솔루션을 결합한 $K-1$개의 선호도 쌍을 생성

즉, $P = \{(x, y1, yk) | k \in \{2, \cdots, K\}\}$

이는 최적의 솔루션으로부터 학습하도록 장려함과 동시에 다양한 차선의 솔루션으로부터 학습하는 것을 억제하여 BC2쌍을 사용하는 것보다 효율적

✓ 전체 모델 구조

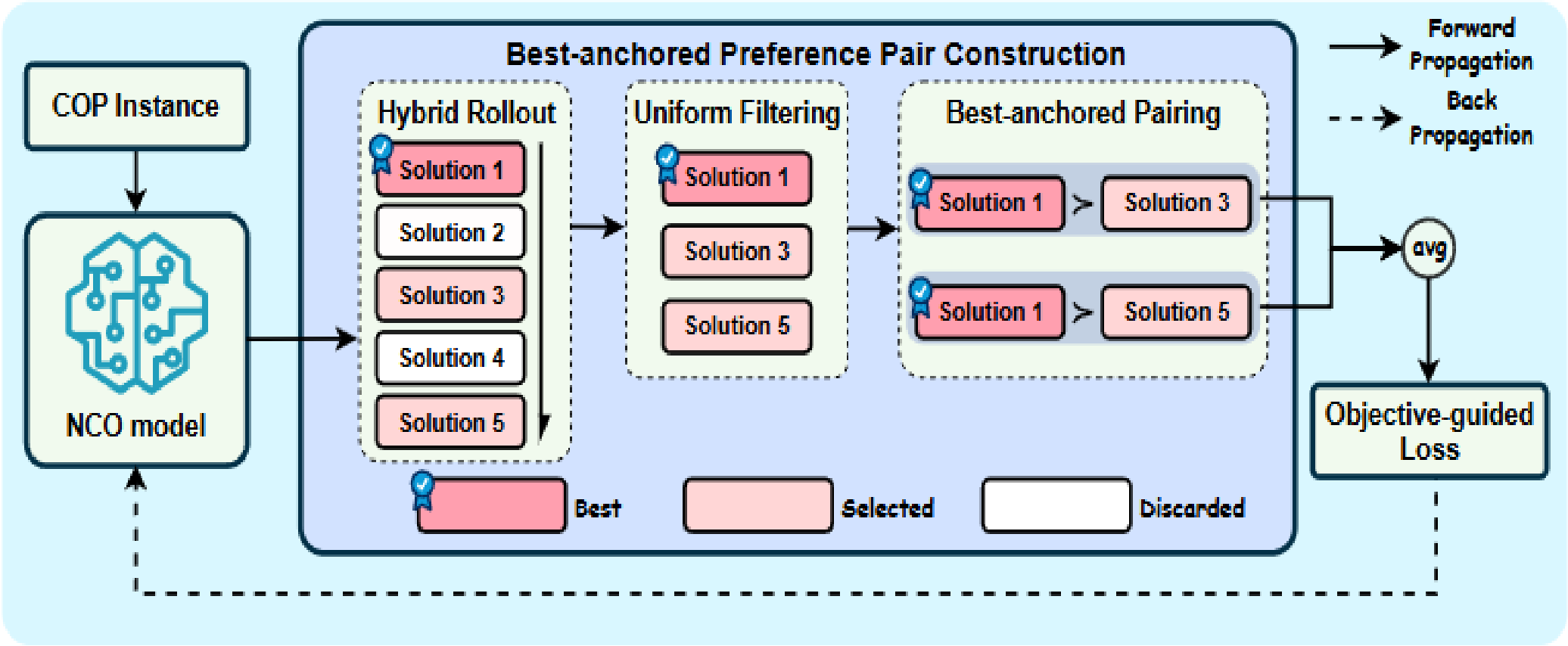


Figure 1. The pipeline of best-anchored and objective-guided preference optimization (BOPO).

❖ Objective-guided Preference Optimization Loss

(1) Objective function

the negative of the objective function:

$$f^*(\boldsymbol{y}, \boldsymbol{x}) = -g(\boldsymbol{y}). \tag{1}$$

(2) average log-likelihood

$$f_{\boldsymbol{\theta}}(\boldsymbol{y}, \boldsymbol{x}) = \frac{1}{|\boldsymbol{y}|} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{|\boldsymbol{y}|} \sum_{t=1}^{|\boldsymbol{y}|} \log \pi_{\boldsymbol{\theta}}(y_t|\boldsymbol{y}_{<t}, \boldsymbol{x}). \tag{2}$$

✓ 해 y가 입력 x에서 생성될 log-likelihood의 step의 평균값

(3) Bradley-Terry (BT) 모델 기반의 쌍 선호 모델

$$p_{\boldsymbol{\theta}}(\boldsymbol{y}_w \succ \boldsymbol{y}_l|\boldsymbol{x}) = \sigma(\beta(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_l)(f_{\boldsymbol{\theta}}(\boldsymbol{y}_w, \boldsymbol{x}) - f_{\boldsymbol{\theta}}(\boldsymbol{y}_l, \boldsymbol{x}))), \tag{3}$$

✓ 더 좋은 해 yw가 yl보다 더 좋다고 판단될 확률 → fθ 점수 차이를 sigmoid 함수로 확률화

→ 스케일링 factor β를 곱해 **선호 강도 조정**

"좋은 해가 선택될 확률 $p_\theta(y_w \succ y_l \mid x)$ 를 높이자"
→ 즉, log-likelihood를 최대화하자
→ 또는 $-\log p_\theta(y_w \succ y_l \mid x)$ 를 최소화하자"

✓ 위의 목적으로 (1),(2),(3) 식들을 이용하여 BOPO의 Loss function(4)을 도출함

$$\mathcal{L}_{BOPO}(\pi_\theta, x, y_w, y_l) = -\log \sigma \left(\underbrace{\frac{g(y_l)}{g(y_w)}}_{\text{Adaptive Scaling}} \left(\underbrace{\frac{\log \pi_\theta(y_w|x)}{|y_w|} - \frac{\log \pi_\theta(y_l|x)}{|y_l|}}_{\text{Average Log-likelihood Difference}} \right) \right). \quad (4)$$

Algorithm 1 BOPO Training

Input: Dataset $\mathcal{X}$, number of epochs E, number of training steps T, batch size D, number of obtained solutions B, number of filtered solutions K, and learning rate η
Initialize model parameter $\boldsymbol{\theta}$
for $epoch = 1$ **to** E **do**
  for $step = 1$ **to** T **do**
    $\boldsymbol{x}_i \leftarrow \textsc{SampleInstance}(\mathcal{X})\ \forall i \in \{1, \ldots, D\}$
    $\mathcal{S}_i \leftarrow \textsc{HybridRollout}(\boldsymbol{x}_i, B)\ \forall i \in \{1, \ldots, D\}$
    $\mathcal{C}_i \leftarrow \textsc{UniformFiltering}(\mathcal{S}_i, K)\ \forall i \in \{1, \ldots, D\}$
    $\mathcal{P}_i \leftarrow \textsc{Best-anchoredPairing}(\mathcal{C}_i)\ \forall i \in \{1, \ldots, D\}$
    Compute $\mathcal{L}_{BOPO}(\pi_{\boldsymbol{\theta}}, \boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_l)$ using Equation (4)
    $\mathcal{L}(\boldsymbol{\theta}) \leftarrow \frac{1}{D}\sum_{i=1}^{D}\frac{1}{|\mathcal{P}_i|}\sum_{(\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l)\in\mathcal{P}_i}\mathcal{L}_{BOPO}(\pi_{\boldsymbol{\theta}}, \boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_l)$
    $\boldsymbol{\theta} \leftarrow \text{Adam}(\boldsymbol{\theta}, \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}), \eta)$
  end for
end for

- **θ** : 학습할 정책 네트워크(π)의 파라미터
- **x_i:** 학습 데이터셋에서 무작위로 선택한 문제 인스턴스
- **S_i:** 인스턴스 x_i에 대해 hybrid rollout 방식으로 생성한 **B개의 후보해** 집합
- **C_i**: 후보해 S_i 중 **K개를 균등하게 샘플링**하여 확보한 서브셋
- **P_i**: C_i로부터 anchor pair을 생성 k - 1 개의 쌍

Methodology- Code

TSPModel.py 코드 중 **hybrid rollout** 적용코드

```
            break

    elif self.training or self.model_params['eval_type'] == 'hybrid':
        while True:
            selected = probs.reshape(batch_size * sols_num, -1).multinomial(1) \
                .squeeze(dim=1).reshape(batch_size, sols_num)
            # greedy
            selected[torch.arange(batch_size), 0] = probs[torch.arange(batch_size), 0].argmax(dim=-1)
            # shape: (batch, B)

            prob = probs[state.BATCH_IDX, state.B_IDX, selected] \
                .reshape(batch_size, sols_num)
            # shape: (batch, B)

            if (prob != 0).all():
                break
```

typical POMO (Kwon et al., 2020)모델을 활용하여 설계.

Methodology- Code

TSPTrainer.py 코드 중 **Uniform Filtering**과 **Best-anchored Pairing** 의 적용코드

```
# goal shape: (batch_size, B)
sorted_idx = torch.argsort(goal, descending=True, dim=-1)
b = goal.shape[1]

idx = sorted_idx[:, :: b // k]
bs_idx = torch.arange(batch_size).view(-1, 1).repeat(1, idx.shape[1])

log_prob = prob_list.log().mean(dim=2)
# shape: (batch_size, B)

f = goal[bs_idx[:, 1:], idx[:, 1:]] / goal[bs_idx[:, 0], idx[:, 0]].view(-1, 1)

first_idx_values = log_prob[torch.arange(batch_size), idx[:, 0]]
other_idx_values = log_prob.gather(1, idx[:, 1:])
log_prob_pairs = f * (first_idx_values.unsqueeze(1) - other_idx_values)  # [batch_size, K-1]
loss_list = -torch.log(torch.sigmoid(log_prob_pairs))  # [batch_size, K-1]
loss = loss_list.mean()

score = -torch.max(goal.float(), dim=1)[0].mean() # negative sign to make positive value

self.model.zero_grad()
loss.backward()
self.optimizer.step()

return score.item(), loss.item()
```

Methodology- Code

`first_idx_values = log_prob[torch.arange(batch_size), idx[:, 0]]` 해석

```python
# 예시 데이터
batch_size = 3
B = 5  # 솔루션 개수
K = 3  # 선택할 솔루션 개수

# log_prob 예시 (각 배치, 각 솔루션의 로그 확률)
log_prob = torch.tensor([
    [-2.1, -1.8, -2.5, -1.9, -2.3],  # 배치 0
    [-1.7, -2.0, -1.5, -2.2, -1.9],  # 배치 1
    [-2.4, -1.6, -2.1, -1.8, -2.0]   # 배치 2
])

# idx 예시 (정렬된 솔루션 인덱스들)
idx = torch.tensor([
    [1, 3, 0],  # 배치 0: 솔루션 1이 최고, 3이 두번째, 0이 세번째
    [2, 4, 1],  # 배치 1: 솔루션 2가 최고, 4가 두번째, 1이 세번째
    [1, 3, 4]   # 배치 2: 솔루션 1이 최고, 3이 두번째, 4가 세번째
])

# idx[:, 0] = [1, 2, 1] (각 배치의 최고 솔루션 인덱스)
```

고급 인덱싱 작동 방식:

```python
torch.arange(batch_size)  # [0, 1, 2]
idx[:, 0]                 # [1, 2, 1]

# Advanced indexing: log_prob[행_인덱스, 열_인덱스]
first_idx_values = log_prob[torch.arange(batch_size), idx[:, 0]]
# = log_prob[[0, 1, 2], [1, 2, 1]]
# = [log_prob[0, 1], log_prob[1, 2], log_prob[2, 1]]
# = [-1.8, -1.5, -1.6]
```

Methodology- Code

`log_prob_pairs = f * (first_idx_values.unsqueeze(1) - other_idx_values)` 상세 해석

✓ # f = 차선책들의 목적함수 값 / 최적 해의 목적함수 값

unsqueeze(1) 작동 과정

python

```python
# STEP 1: unsqueeze(1) 적용
first_idx_values.unsqueeze(1)
# 원본: [-1.8, -1.5, -1.6]  shape: (3,)
# 결과: [[-1.8], [-1.5], [-1.6]]  shape: (3, 1)

print("Before unsqueeze:", first_idx_values.shape)          # torch.Size([3])
print("After unsqueeze(1):", first_idx_values.unsqueeze(1).shape)  #
torch.Size([3, 1])
```

Methodology- Code

`log_prob_pairs = f * (first_idx_values.unsqueeze(1) - other_idx_values)` 상세 해석

Broadcasting 과정

```python
# STEP 2: Broadcasting을 통한 차이 계산
first_expanded = first_idx_values.unsqueeze(1)  # shape: (3, 1)
# [[-1.8],
#  [-1.5],
#  [-1.6]]

other_idx_values  # shape: (3, 2)
# [[-1.9, -2.1],
#  [-1.9, -2.0],
#  [-1.8, -2.0]]

# Broadcasting: (3,1) - (3,2) = (3,2)
difference = first_expanded - other_idx_values
# [[-1.8] - [-1.9, -2.1]] = [[-1.8-(-1.9), -1.8-(-2.1)]] = [[0.1, 0.3]]
# [[-1.5] - [-1.9, -2.0]] = [[-1.5-(-1.9), -1.5-(-2.0)]] = [[0.4, 0.5]]
# [[-1.6] - [-1.8, -2.0]] = [[-1.6-(-1.8), -1.6-(-2.0)]] = [[0.2, 0.4]]

difference = torch.tensor([
    [0.1, 0.3],   # 배치 0: 최고해가 차선책들보다 높은 로그확률 차이
    [0.4, 0.5],   # 배치 1
    [0.2, 0.4]    # 배치 2
])  # shape: (3, 2)
```

Methodology- Code

`log_prob_pairs = f * (first_idx_values.unsqueeze(1) - other_idx_values)` 상세 해석

최종 결과 ✓ # f = 차선책들의 목적함수 값 / 최적 해의 목적함수 값

python

```python
# STEP 3: Objective-guided scaling 적용
log_prob_pairs = f * difference
# shape: (3, 2) * (3, 2) = (3, 2)

log_prob_pairs = torch.tensor([
    [1.05 * 0.1, 1.23 * 0.3],  # = [0.105, 0.369]
    [1.15 * 0.4, 1.33 * 0.5],  # = [0.460, 0.665]
    [1.12 * 0.2, 1.25 * 0.4]   # = [0.224, 0.500]
])  # shape: (3, 2)
```

Methodology- Code

 # 1. 차원 맞추기 위한 unsqueeze

✓ first_idx_values.unsqueeze(1) # (3,) → (3,1)

 # 2. Broadcasting으로 차이 계산

✓ difference = first_expanded - other_idx_values # (3,1) - (3,2) = (3,2)

 # 3. Objective-guided scaling 적용

✓ log_prob_pairs = f * difference # (3,2) * (3,2) = (3,2)

 # 4. 최종적으로 BOPO loss 계산에 사용

✓ loss_list = -torch.log(torch.sigmoid(log_prob_pairs))

V Experimental Results

✓ Table 1. Average gaps (%) of evaluated methods on JSP

		Non-constructive						Greedy Constructive									Sampling Constructive								
		Exact Solver		RL-based Improvement				Traditional PDR			RL			SLL		BOPO	$B' = 128$					$B' = 512$			
	Shape	Gurobi	OR-Tools	$L2S_{500}$	NLS_A	TGA_{500}	$L2S_{5k}$	SPT	MOR	MWR	L2D	SchN	CL	$SLIM_{MGL}$	SLIM	BOPO	L2D	CL	$SLIM_{MGL}$	SLIM	BOPO	$SLIM_{MGL}$	SLIM	BOPO
TA	15×15	0.1	0.1	9.3	7.7	8.0	6.2	25.8	20.5	19.2	26.0	15.3	14.3	13.1	13.8	13.6	17.1	9.0	8.8	7.2	7.1	7.2	6.5	**6.3**
	20×15	3.2	0.2	11.6	12.2	9.9	8.3	32.9	23.6	23.4	30.0	19.4	16.5	16.1	15	14.3	23.7	10.6	11.0	9.3	9.0	10.4	8.8	**8.3**
	20×20	2.9	0.7	12.4	11.5	10.0	9.0	27.8	21.7	21.8	31.6	17.2	17.3	15.3	15.2	15.1	22.6	10.9	11.1	10.0	9.8	10.0	**9.0**	9.1
	30×15*	10.7	2.1	14.7	14.1	13.3	9.0	35.1	22.7	23.7	33.0	19.1	18.5	17.7	17.1	16.6	24.4	14.0	14.0	11.0	11.0	12.2	10.6	**10.3**
	30×20*	13.2	2.8	17.5	16.4	16.4	12.6	34.4	24.9	25.2	33.6	23.7	21.5	19.3	18.5	17.1	28.4	16.1	16.3	13.4	13.3	14.9	12.7	**12.2**
	50×15*	12.2	3.0	11.0	11.0	9.6	4.6	24.1	17.3	16.8	22.4	13.9	12.2	13.4	10.1	9.8	17.1	9.3	9.2	5.5	5.8	8.2	**4.9**	**4.9**
	50×20*	13.6	2.8	13.0	11.2	11.9	6.5	25.6	17.7	17.9	26.5	13.5	13.2	14.0	11.6	11.8	20.4	9.9	10.6	8.4	8.0	9.8	7.6	**7.4**
	100×20*	11.0	3.9	7.9	5.9	6.4	3.0	14.4	9.2	8.3	13.6	6.7	5.9	7.4	5.8	4.9	13.3	4.0	4.8	2.3	1.8	4.4	2.1	**1.4**
	Avg	8.4	2.0	12.2	11.3	10.7	7.4	27.5	19.7	19.5	27.1	16.1	14.9	14.5	13.4	12.9	20.8	10.4	10.7	8.4	8.2	9.6	7.8	**7.5**
LA	10×5*	0.0	0.0	2.1	-	2.1	1.8	14.8	16.0	16.0	14.3	12.1	-	8.6	9.3	6.0	8.8	-	3.7	1.9	2.7	2.5	**1.1**	2.1
	10×10	0.0	0.0	4.4	-	1.8	0.9	15.7	18.1	12.2	23.7	11.9	-	9.1	8.9	8.2	10.4	-	3.5	3.1	2.3	2.4	2.5	**2.1**
	15×5*	0.0	0.0	0.0	-	0.0	0.0	14.9	3.9	5.5	7.8	2.7	-	1.5	2.6	1.1	1.8	-	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
	15×10	0.0	0.0	6.4	-	3.6	3.4	28.7	23.7	17.8	27.2	14.6	-	11.7	11.6	11.0	15.2	-	6.3	5.2	5.8	5.6	5.0	**4.9**
	15×15	0.0	0.0	7.3	-	5.5	5.9	24.6	18.1	18.2	27.1	16.1	-	13.5	13.6	12.2	17.4	-	7.1	6.8	6.5	6.7	5.6	**4.9**
	20×5*	0.0	0.0	0.0	-	0.0	0.0	13.7	3.8	5.2	6.3	3.6	-	1.5	2.1	0.4	1.1	-	0.5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
	20×10	0.0	0.0	7.0	-	5.0	2.6	33.4	20.9	17.2	24.6	15.7	-	14.3	12.1	12.2	18.3	-	7.9	6.9	5.9	7.1	5.6	**4.6**
	30×10*	0.0	0.0	0.2	-	0.0	0.0	13.9	6.5	8.6	8.4	3.1	-	3.1	2	2.4	1.8	-	0.3	**0.0**	**0.0**	0.1	**0.0**	**0.0**
	Avg	0.0	0.0	3.4	-	2.3	1.8	20.0	13.9	12.6	17.4	10.0	-	7.9	7.8	6.7	10.6	-	3.7	3.0	2.9	3.0	2.5	**2.3**
DMU	20×15	5.3	1.8	-	-	-	-	28.0	30.9	28.8	39.0	-	-	17.0	18	17.5	29.3	19.4	13.7	12.0	11.2	12.7	11.3	**10.4**
	20×20	4.7	1.9	-	-	-	-	31.3	27.4	27.3	37.7	-	-	22.6	19.4	20.3	27.1	16.0	15.3	13.5	12.7	14.1	12.3	**11.8**
	30×15*	14.2	2.5	-	-	-	-	31.5	37.4	32.3	42.0	-	-	24.1	21.8	19.1	34.0	16.5	18.4	14.4	13.9	17.5	14.0	**12.9**
	30×20*	16.7	4.4	-	-	-	-	34.4	34.7	31.4	39.7	-	-	25.6	25.7	25.6	33.6	20.2	19.0	17.1	16.5	17.8	15.8	**15.5**
	40×15*	16.3	4.1	-	-	-	-	24.0	36.7	27.5	35.6	-	-	20.1	17.5	15.9	31.5	17.6	15.8	11.7	11.4	15.3	**10.9**	**10.9**
	40×20*	22.5	4.6	-	-	-	-	37.2	37.1	32.9	39.6	-	-	23.5	22.2	22.3	35.8	25.6	19.8	16.0	16.7	19.0	**14.8**	15.9
	50×15*	14.9	3.8	-	-	-	-	24.8	35.5	28.0	36.5	-	-	18.2	15.7	14.5	32.7	21.7	15.6	11.2	11.2	15.3	10.6	**10.4**
	50×20*	22.5	4.8	-	-	-	-	30.1	37.0	30.8	39.5	-	-	25.8	22.4	25.2	35.1	15.2	20.8	15.8	16.5	20.0	**15.0**	15.5
	Avg	14.6	3.5	-	-	-	-	30.2	34.6	29.9	38.7	-	-	22.1	20.3	20.0	32.5	19.0	17.3	14.0	13.8	16.5	13.1	**12.9**

V Experimental Results

✓ Table 2. Results on 1000 uniformly generated TSP instance

Table 2. Results on 1000 uniformly generated TSP instances.

Method	TSP20			TSP50			TSP100		
	Obj.↓	Gap↓	Time↓	Obj.↓	Gap↓	Time↓	Obj.↓	Gap↓	Time↓
Concorde	3.83	0.00	5m	5.69	0.00	13m	7.75	0.00	1h
Gurobi	3.83	0.00	7s	5.69	0.00	2m	7.75	0.00	17m
LKH3	3.83	0.00	42s	5.69	0.00	6m	7.75	0.00	25m
POMO	3.83	0.04	3.3s	5.70	0.21	6.4s	7.80	0.46	11.4s
DABL	3.83	0.01	3.3s	5.69	0.04	6.4s	7.77	0.29	11.4s
SLIM	3.85	0.22	3.3s	5.78	1.51	6.4s	8.18	5.51	11.4s
BOPO	3.83	0.02	3.3s	5.70	0.14	6.4s	7.78	0.37	11.4s
POMO (aug.)	3.83	0.00	3.6s	5.69	0.03	6.6s	7.77	0.14	18.1s
DABL (aug.)	**3.83**	**0.00**	3.6s	**5.69**	**0.00**	6.6s	7.75	0.05	18.1s
SLIM (aug.)	3.84	0.01	3.6s	5.70	0.15	6.6s	7.84	1.17	18.1s
BOPO (aug.)	**3.83**	**0.00**	3.6s	5.69	0.01	6.6s	**7.75**	**0.04**	18.1s

✓ Table 3. Generalization on TSPLIB with various problem shapes

Table 3. Generalization on TSPLIB with various problem shapes.

Method	$n < 100$ (6 instances)			$100 \leq n < 200$ (21 instances)			$200 \leq n < 500$ (16 instances)			$500 \leq n < 1k$ (6 instances)		
	Obj.↓	Gap↓	Time↓	Obj.↓	Gap↓	Time↓	Obj.↓	Gap↓	Time↓	Obj.↓	Gap↓	Time↓
POMO (aug.)	6.26	2.36	0.15s	6.75	3.08	0.27s	10.63	14.81	0.95s	16.22	30.14	4.6s
SLIM (aug.)	6.19	1.36	0.15s	6.88	5.24	0.27s	10.82	16.99	0.95s	19.40	55.57	4.6s
BOPO (aug.)	**6.19**	**1.26**	0.15s	**6.72**	**2.55**	0.27s	**10.21**	**10.41**	0.95s	**15.29**	**22.44**	4.6s

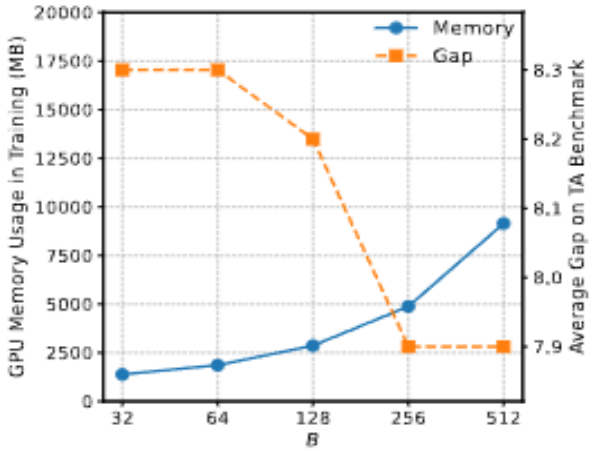
✓ Table 4. Average gaps (%) on FJSP benchmarks.

Table 4. Average gaps (%) on FJSP benchmarks.

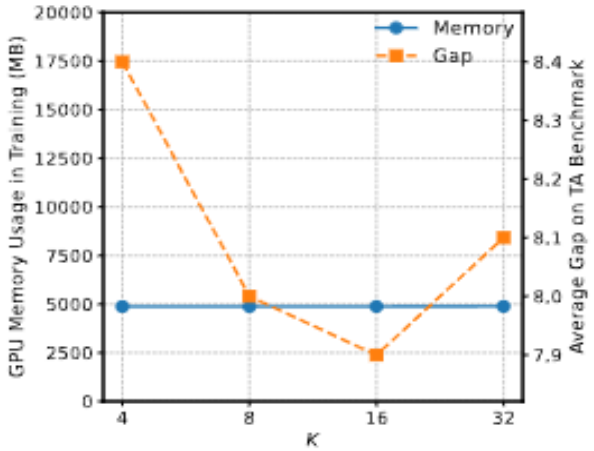
	Greedy Constructive							Sampling Constructive				
	Traditional PDR			RL			BOPO	B'=100			B'=256	B'=512
Benchmarks	SPT	MOR	MWR	DNN	HG	RS	BOPO	HG	RS	BOPO	BOPO	BOPO
LA(e-data)	26.1	17.7	20.5	15.5	15.5	13.2	14.5	8.2	6.9	6.1	5.4	**5.0**
LA(r-data)	28.7	14.4	17.8	12.1	11.2	9.6	8.4	5.8	4.7	4.0	3.6	**3.4**
LA(v-data)	17.8	6.0	6.6	5.4	4.3	3.8	1.8	1.4	0.8	0.6	0.5	**0.4**

V Hyper parameter

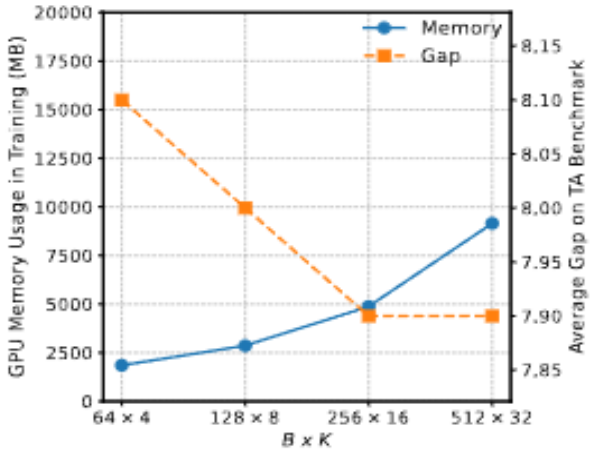
- ✓ 두 가지 중요한 Hyper paramete로 hybrid rollout의 솔루션 수 B와 필터링된 솔루션 수 K가 존재
- ✓ 성능은 (B = 256)까지 크게 향상, 추가적인 증가는 수익이 감소하므로 (B = 256)이 합리적인 선택
- ✓ K를 늘리면 더 많은 선호도 쌍이 생성, but 솔루션 간의 유사성도 증가 이는 솔루션이 동일한 로컬 영역에서 나올 가능성이 더 높기 때문. 실험 결과 (K = 16)이 최고의 성능을 달성
- ✓ 이는 권장 파라미터로서 (B = 256) 및 (K = 16)을 선택.



(a)



(b)



(c)

VI Conclusion

- ✓ 최고 앵커 선호도 쌍 구성과 COP를 위한 새로운 목표 지향적 쌍별 손실 함수를 도입하여 주류 RL 및 최근 SLL 패러다임보다 더 높은 샘플 효율성을 달성.
- ✓ JSP, TSP 및 FJSP에 대한 광범위한 실험은 BOPO가 훨씬 적은 시간이 소요되면서 최첨단 신경 구성 방법보다 우수한 성능을 보여줌.
- ✓ 제안된 방법은 비용이 많이 드는 레이블이나 마르코프 결정 프로세스의 특수 설계를 필요로 하지 않으므로 비용이 적게듬
. 또한 다양한 COP 해결을 위해 다양한 신경 모델에 쉽게 적용할 수 있는 일반적인 훈련 패러다임을 확립 가능
- ✓ 제한 사항으로 SLIM과 유사하게 여전히 상대적으로 많은 수의 롤아웃 솔루션이 필요함. 이는 효과적인 모델 학습을 위해 고품질 솔루션을 수집하는 데 비용이 발생
- ✓ 연구방향으로 문제 불변성 및 솔루션 대칭성을 활용하여 다양하고 고품질의 훈련 데이터를 효율적으로 생성. 또 다른 방향은 훈련 중에 문제별 휴리스틱을 통합하여 솔루션 품질을 효율적으로 향상시켜 모델에 더 나은 학습 신호 제공