Towards Generalizable Multi-Policy Optimization with Self-Evolution for Job Scheduling

26.01.08

임제원

I Introduction

✓ 대부분의 Job Scheduling Problems (JSPs)에서는 지도학습이 어려운 상황

→ 라벨 데이터를 구하기 힘들기 때문

→ 이러한 이유로 JSPs에서는 강화학습이 주목 받고 있음

✓ 하지만 강화학습에는 두 가지 문제가 존재

1. Exploration : 강화학습은 탐색 공간이 기하급수적으로 증가하므로 고품질 해를 찾으려면 넓은 탐색이 필수 TSP와 같은 문제는 대칭성으로 이러한 탐색 효율을 향상시키지만, JSPs에서는 대칭성을 구하기 어려움

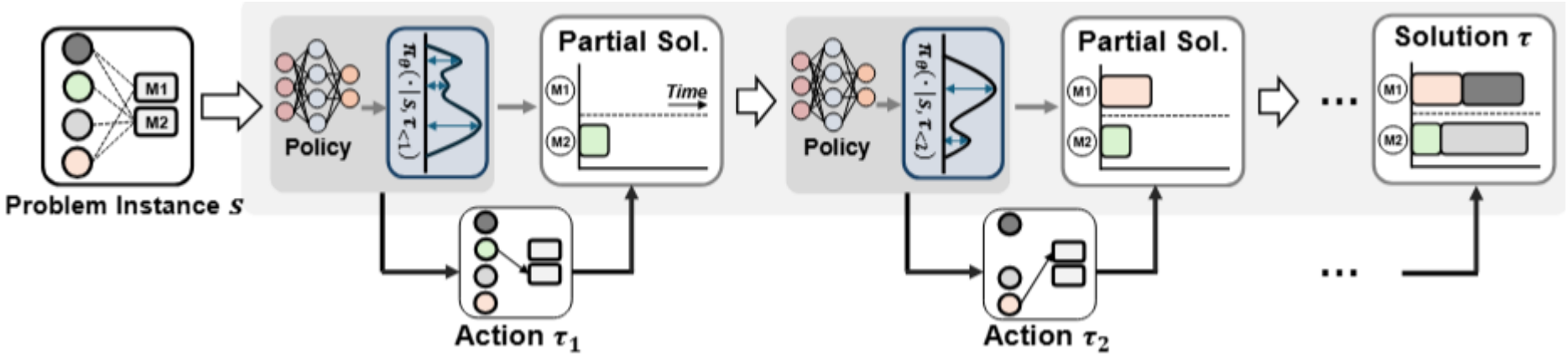
   → 충분한 탐색 부족으로 지역최적해에 빠지는 문제 발생

2. Reward Shaping : 수많은 변동이 생기는 상황에서 step 마다 이러한 보상을 매번 설계하는 것은 매우 어려움 최종적으로 얻고자 하는 목적함수의 값만 이용할 수 있지만, 중간 과정에서 어떠한 선택이 최종 목적 값에 영향을 미치는지 파악이 불가능하다는 단점이 존재함

Ⅱ Contribution

- ✓ 이러한 강화학습의 한계점을 해결하기 위해 저자는 새롭고 일반적인 학습 프레임워크를 제안
- ✓ 목표와 **모델 파라미터를 공유**하지만, 서로 다른 전략을 사용하여 문제를 해결하는 여러 정책을 학습하고 **각 정책이 완전히 자체 진화적인 방식**으로 문제를 해결하는 **MP-ASIL**(Multi-Policy Optimization with Adaptive Self-Imiation Learning)을 제안
- ✓ 이러한 방식은 탐색 능력을 강화하여 지역 최적해를 벗어날 수 있음
- ✓ Self Labeling 과정에서 스스로 생산한 좋은 해를 라벨로 삼아 모방하여, 계산이 복잡한 MDP가 필요하지 않음
- ✓ 기존 Self Labeling과 다르게, 학습하고자 하는 해의 품질에 따라 모방 강도를 조절하는 적응형 모방학습 제안
- ✓ 총 6개의 JSPs 문제에서 SOTA를 달성함 (Benchmark & Synthetic)

❖ Illustrative example of sequential decision-making process using neural constructive heuristics to build a solution τ



❖ Multi-Policy Representation: Latent Conditioned Polices

$$\Pi = \{\pi_\theta(\cdot \mid s, z^i) \mid z^i \sim Z, i = 1, 2, \ldots, k\}$$

Policy Action State Policy i Policy Sets

= latent variable

→ 정책 파이가 특정 state에서 policy i로 action을 할 확률

❖ **MP-ASIL: Multi-Policy Optimization with Adaptive Self-Imitation Learning**

$$\mathbb{E}_{s\sim D}\,\mathbb{E}_{z^1,\ldots,z^k\sim Z}\,\mathbb{E}_{\tau^1\sim\pi_\theta\left((\cdot\mid s,z^1\right),\ldots,\tau^k\sim\pi_\theta\left((\cdot\mid s,z^k\right)}\min\{f(\tau^1,s),\ldots,f\left(\tau^k,s\right)\}$$

Problem Instance　Each Policy　Probability Distribution with Variable　Minimizing

→ 하나의 job instance에 대해, 서로 다른 latent variable을 가진 동일한 base policy로 k개의 해를 샘플링하고, 그 중 가장 좋은 해의 목적값을 최소화하도록 학습한다.

❖ **Training procedure**

$|\mathcal{O}|$: operation 수

k개 중 최적　k개 평균　k개 표준편차

$$\mathcal{L}_{MP-ASIL}=-\left(\frac{|f(\tau^*,s)\ -mean(s)|}{std(s)}\right)\frac{1}{|\mathcal{O}|}\sum_{t=1}^{|\mathcal{O}|}\log\pi_\theta(\tau_t^*\mid s,\ \tau_{<t}^*,z^*)$$

→ 선택된 최적 해 τ*의 행동 시퀀스를 현재 policy가 얼마나 높은 확률로 재현하는지를 학습

❖ **Example**

$$\mathcal{L}_{MP\text{-}ASIL} = -\underbrace{\left(\frac{f(\tau^*, s) - \text{mean}(s)}{\text{std}(s)}\right)}_{\text{정규화된 개선량}} \cdot \underbrace{\frac{1}{|O|}\sum_{t=1}^{|O|} \log \pi_\theta(\tau_t^* \mid s, \tau_{<t}^*, z^*)}_{\text{최적해}(\tau^*)\text{를 따라하게 만드는 항}}$$

→ **정규화된 개선량(C)**을 통해 현재 샘플링 된 해 중 최적해가 좋으면 강하게 학습, 좋지 않으면 약하게 학습

➢ If k=3 | (k = sampling 수), minimizing makespan

(a) 강하게 학습되는 예시

Policy 1 = 12
Policy 2 = 20
Policy 3 = 22

Mean(s) = (12+20+22) / 3 = 18
Std(s) = 4.3

C = (12-8) / 4.3 = 1.39

(b) 약하게 학습되는 예시

Policy 1 = 18
Policy 2 = 19
Policy 3 = 30

Mean(s) = (18+19+30)/3 = 22.3
Std(s) = 5.4

C = (18-22.3) / 5.4 = 0.79

이러한 loss function은 단순히 best trajectory를 따라하는 게 아니라, best가 "**평균 대비 얼마나 좋은지**"를 표준편차로 정규화해서 가중치로 쓰기 때문에, 어떤 best는 강하게, 어떤 best는 약하게 모방할 수 있음

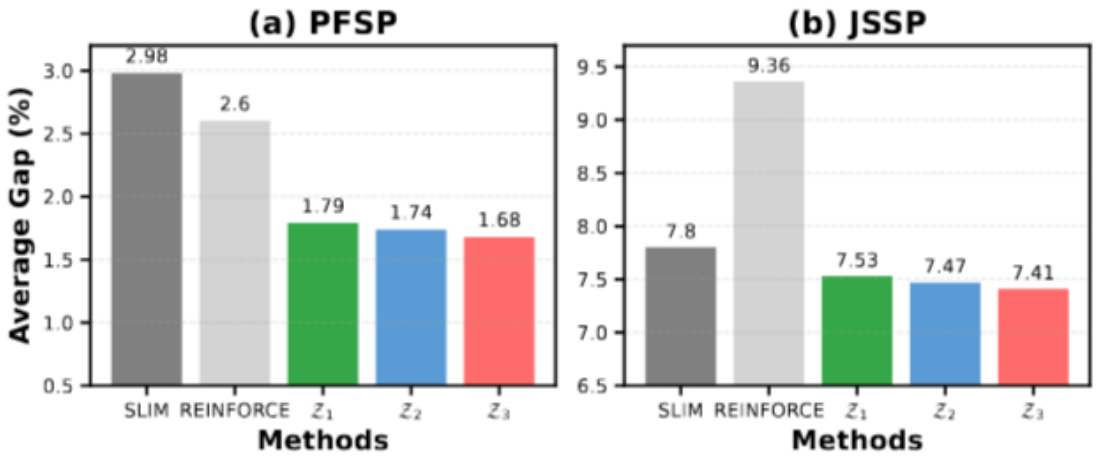
❖ The Effect of Latent Distributions, Z

✓ 다른 정책은 인덱싱하는 잠재변수 z는 분포 Z에 따라 모델 성능에 영향을 미칠 수 있음

✓ 본 논문에서는 세 가지 잠재 분포 하에서 훈련된 모델 평가 진행(Appendix B.3)

1. $Z_1 = U(-1,1)^{16}$
2. $Z_2 = U(-1,1)^{8}$+ 8-dimensional categorical (one-hot) distribution
3. $Z_3 = U(-1,1)^{4}$+ 4-dimensional categorical (one-hot) distribution

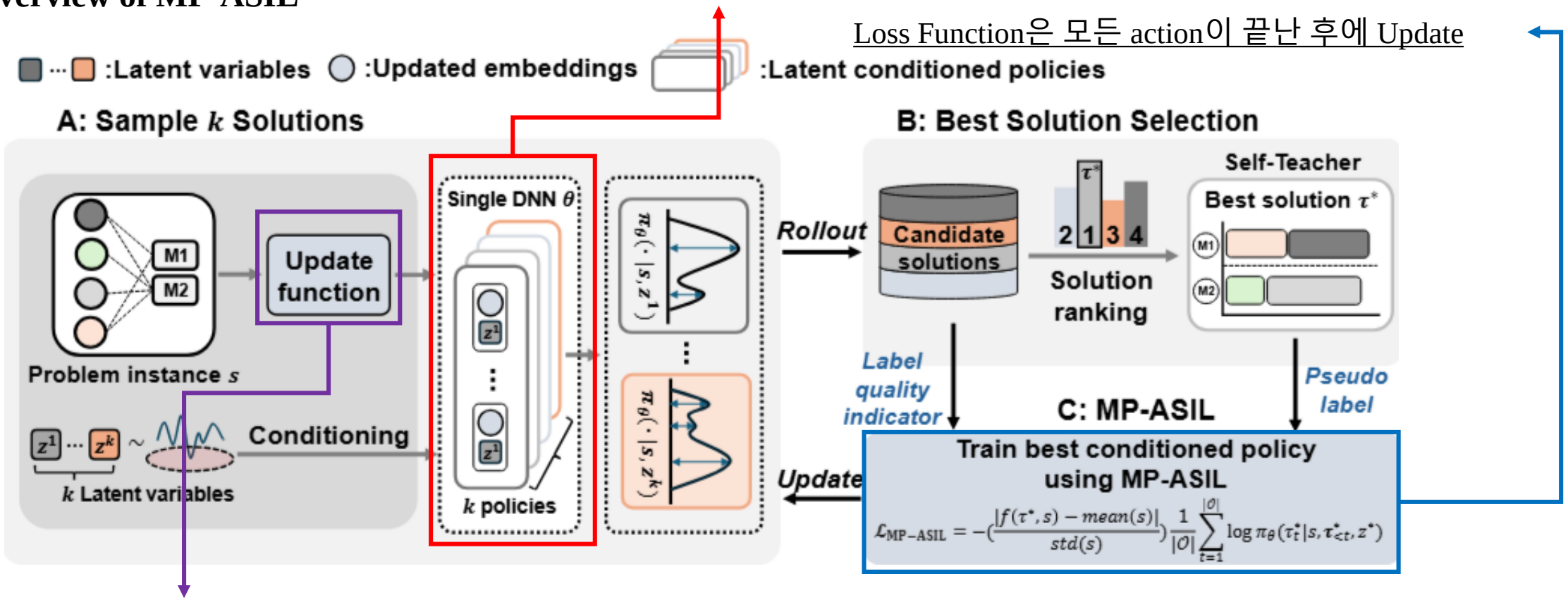
✓ 이렇게 샘플링 된 z는 Q,K,V에서 Concat 되어 모델의 Probability Distribution 에 영향을 줌 (Appendix B.1)



❖ **An overview of MP-ASIL**

LEHD & HELD 기반 Encoder & Decoder 구조 (Appendix B.1)

Loss Function은 모든 action이 끝난 후에 Update



본 논문에서는 총 6가지의 문제를 HELD와 LEHD로 나눠서 해결 (LEHD: SMSP, UPMSP, JSSP, FJSSP | HELD : PFSP, FFSP)

HELD는 instance-level job embedding은 고정한 상태에서, 선택 이력과 같은 동적 정보를 query에 반영하여 update function으로 작동
LEHD는 매 step마다 동적 정보를 job embedding에 직접 반영하여 update function을 수행한다. (동적 정보 : 현재까지 선택된 작업)

위와 같은 두 경우로 나뉘기 때문에 Update Function으로 표현함 (Appendix B.1)

❖ Algorithm

Algorithm 1 `MP-ASIL` training

Input: Model parameters θ, instance distribution $\mathcal{D}$, latent variable distribution $\mathcal{Z}$, number of epochs E, number of training steps T, batch size B, and number of policies k.
Initialize model parameters θ.
for $epoch = 1$ to E **do**
  for $step = 1$ to T **do**
    $s_i \leftarrow \texttt{SampleInstance}(\mathcal{D}), \quad \forall i \in \{1, \ldots, B\}$
    $\Pi_i = \{\pi_\theta(\boldsymbol{\tau}_i^j \mid s_i, z_i^j)\}_{j=1}^k \leftarrow \texttt{SamplePolicy}(\mathcal{Z}), \quad \forall i \in \{1, \ldots, B\}$
    $\{\boldsymbol{\tau}_i^1, \ldots, \boldsymbol{\tau}_i^k\} \leftarrow \texttt{SampleRollout}(\Pi_i), \quad \forall i \in \{1, \ldots, B\}$
    $\boldsymbol{\tau}_i^* = \arg\min_{\boldsymbol{\tau}_i^j \in \{\boldsymbol{\tau}_i^1, \ldots, \boldsymbol{\tau}_i^k\}} f(\boldsymbol{\tau}_i^j, s_i), \quad \forall i \in \{1, \ldots, B\}$ ▷ Select the best solution.
    $\mathcal{L}_{\text{MP-ASIL}} = -\frac{1}{B}\frac{1}{|\mathcal{O}|}\sum_{i=1}^{B}\frac{|f(\boldsymbol{\tau}_i^*, s_i) - mean(s_i)|}{std(s_i)}\sum_{t=1}^{|\mathcal{O}|}\log \pi_\theta(\tau_{i,t}^* \mid s_i, \boldsymbol{\tau}_{i,<t}^*, z_i^*)$
    $\theta \leftarrow \text{Adam}(\theta, \nabla_\theta \mathcal{L}_{\text{MP-ASIL}})$ ▷ Update solely based on the performant policy.
  end for
end for
Output: Trained model parameters θ.

Optimization Problem Definitions

❖ Problem Constraints and Objective Functions (Appendix C)

- ✓ SMSP : **Minimizing Total Weighted Tardiness**
- ✓ UPMSP : Machine Eligibility, Ready Time, Sequence Dependent setup Time → **Minimizing Weighted Tardiness**
- ✓ PFSP : **Minimizing Makespan**
- ✓ FFSP : Machine Eligibility → **Minimizing Makespan**
- ✓ JSSP : Precedence, Machine Eligibility → **Minimizing Makespan**
- ✓ FJSSP : Precedence, Machine Eligibility → **Minimizing Makespan**

V Experiment Result

Method	Type	SMSP 50 Obj. ↓	SMSP 50 Gap ↓	SMSP 50 Time ↓	SMSP 100 • Obj. ↓	SMSP 100 • Gap ↓	SMSP 100 • Time ↓	SMSP 500 • Obj. ↓	SMSP 500 • Gap ↓	SMSP 500 • Time ↓
EDD	Heuristics	0.3268	49.84%	(0s)	0.3950	66.24%	(0s)	0.7287	97.70%	(1s)
ACO [27]	Heuristics	0.7787	>100%	(1.1m)	6.9138	>100%	(2.4m)	646.81	>100%	(28.9m)
DeepACO [27]	Hybrid	0.2296	5.27%	(1.1m)	0.2551	7.36%	(2.6m)	0.5944	61.30%	(29m)
GFACS [16]	Hybrid	0.4202	92.64%	(1.5m)	1.2153	>100%	(3m)	14.612	>100%	(33.7m)
MP-ASIL (k=128)	NCH	**0.2181**	**0.00%**	(5s)	**0.2376**	**0.00%**	(17s)	**0.3691**	**0.16%**	(14.5m)
MP-ASIL (Large)	NCH	0.2181	0.00%	(1m)	0.2376	0.00%	(2.3m)	0.3685	0.00%	(53.5m)

Method	Type	UPMSP 50×3 • Obj. ↓	UPMSP 50×3 • Gap ↓	UPMSP 50×3 • Time ↓	UPMSP 50×6 • Obj. ↓	UPMSP 50×6 • Gap ↓	UPMSP 50×6 • Time ↓	UPMSP 100×6 • Obj. ↓	UPMSP 100×6 • Gap ↓	UPMSP 100×6 • Time ↓
EDD	Heuristics	2836.8	>100%	(1s)	778.5	>100%	(1s)	2472.1	>100%	(2s)
ATCSR_Rm [60]	Heuristics	877.0	22.91%	(6.6m)	294.5	14.40%	(6.9m)	735.6	75.94%	(10.0m)
Cho et al. (S=6) † [41]	NCH	784.4	9.94%	(1.8m)	294.2	14.29%	(2.4m)	502.4	20.16%	(10.6m)
MP-ASIL (k=6)	NCH	**751.9**	**5.37%**	(1.8m)	**275.7**	**7.09%**	(2.4m)	**458.1**	**9.57%**	(10.7m)
MP-ASIL (Large)	NCH	713.5	0.00%	(6.1m)	257.4	0.00%	(24.6m)	418.1	0.00%	(1h)

Method	Type	FFSP 20×12 Obj. ↓	FFSP 20×12 Gap↓	FFSP 20×12 Time ↓	FFSP 50×12 Obj. ↓	FFSP 50×12 Gap↓	FFSP 50×12 Time ↓	FFSP 100×12 Obj. ↓	FFSP 100×12 Gap↓	FFSP 100×12 Time ↓
CPLEX (1m)* [61]	Exact	46.4	81.04%	(17h)	–	–	–	–	–	–
CPLEX (10m)* [61]	Exact	36.6	42.80%	(167h)	–	–	–	–	–	–
SPT* [12]	Heuristics	31.3	22.12%	(40s)	57.0	14.22%	(1m)	99.3	10.71%	(2m)
GA* [62]	Heuristics	30.6	19.39%	(7h)	56.4	13.03%	(16h)	98.7	10.04%	(29h)
PSO* [63]	Heuristics	29.1	13.54%	(13h)	55.1	10.42%	(26h)	97.3	8.48%	(48h)
MatNet (S=24) † [12]	NCH	27.3	6.51%	(8s)	51.5	3.21%	(13s)	91.5	2.02%	(26s)
PolyNet (k=24) † [56]	NCH	26.9	5.11%	(8s)	51.2	2.56%	(13s)	91.1	1.59%	(27s)
MP-ASIL (k=24)	NCH	**26.9**	**4.88%**	(8s)	**51.1**	**2.40%**	(13s)	**90.9**	**1.32%**	(27s)
MP-ASIL (Large)	NCH	25.6	0.00%	(26s)	49.9	0.00%	(1.1m)	89.7	0.00%	(3.2m)

DeepACO, GFACS : Hybrid 신경망 알고리즘

Cho et al.
(2024 Winter Simulation Conference)
REINFOCEMENT 알고리즘 사용
여러 제약이 붙은 UPMSP 해결 (RL)

PolyNet : MP-ASIL처럼 Multi-policy로 샘플링한 논문
(ICLR 2025)
MP-ASIL과 다르게 무조건 best trajectory를 모방
(강도 부여 x) (RL)

V Experiment Result

Table 2: Experiment results on PFSP using the TA benchmark. G: Greedy action selection. The Time metric is reported in Appendix D.2.

Method	Type	20×5 Gap ↓	20×10 Gap ↓	50×5 ● Gap ↓	50×10 ● Gap ↓	100×5 ● Gap ↓	100×10 ● Gap ↓	200×10 ● Gap ↓	**Avg.** Gap ↓
ILS [64]	Heuristics	6.81%	9.45%	4.14%	11.69%	3.44%	9.57%	6.63%	7.39%
IGA [65]	Heuristics	3.36%	10.56%	1.97%	7.53%	1.03%	5.73%	3.43%	4.80%
NEH [66]	Heuristics	2.40%	4.45%	0.66%	4.69%	0.41%	2.04%	1.28%	2.28%
IL (G) [42]	NCH	18.20%	26.96%	12.30%	26.76%	10.13%	19.03%	15.25%	18.38%
PFSPNet (G)* [38]	NCH	–	14.78%	–	11.95%	–	8.21%	–	11.65%
Q-Learning (S=5)* [37]	NCH	9.90%	13.41%	6.24%	15.43%	4.87%	11.64%	8.74%	10.03%
MP-ASIL (k=128)	NCH	**0.37%**	**3.32%**	**0.22%**	**4.05%**	**0.16%**	**2.15%**	**1.51%**	**1.68%**

IL
(2024 AAAI)
Graph Encoder + Imitation Learning 으로 PFSP 문제 해결 (SIL)

PFSPNet
(2023 IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE)
Graph Encoder + Imitation Learning 으로 PFSP 문제 해결 (SIL)

Q-Learning
(2024 ESWA)
Q-Learning 에 NEH 알고리즘을 결합 (RL)

V Experiment Result

Table 3: Experiment results on JSSP using the TA benchmark. NIH: Neural improvement heuristics. The Time metric is reported in Appendix D.2.

Method	Type	15×15 Gap ↓	20×15 Gap ↓	20×20 Gap ↓	30×15 • Gap ↓	30×20 • Gap ↓	50×15 • Gap ↓	50×20 • Gap ↓	100×20 • Gap ↓	**Avg.** Gap ↓
Gurobi (3600s)* [20]	Exact	0.1%	3.2%	2.9%	10.7%	13.2%	12.2%	13.6%	11.0%	8.4%
OR-Tools (3600s)* [67]	Exact	0.1%	0.2%	0.7%	2.1%	2.8%	3.0%	2.8%	3.9%	2.0%
L2D (G)* [11]	NCH	26.0%	30.0%	31.6%	33.0%	33.6%	22.4%	26.5%	13.6%	27.1%
L2D (S=128)* [11]	NCH	17.1%	23.7%	22.6%	24.4%	28.4%	17.1%	20.4%	10.3%	20.5%
SN (G)* [29]	NCH	15.3%	19.4%	17.2%	19.1%	23.7%	13.9%	13.5%	6.7%	16.1%
RASCL (G)* [36]	NCH	14.3%	16.5%	17.3%	18.5%	21.5%	12.2%	13.2%	5.9%	14.9%
RS (G)* [39]	NCH	14.8%	16.5%	16.9%	14.4%	17.7%	6.7%	10.0%	2.6%	12.5%
SI GD (G)* [45]	NCH	9.6%	9.9%	11.1%	9.5%	13.8%	**2.7%**	**6.7%**	1.7%	8.4%
SLIM (S=512)*† [20]	NCH	**6.5%**	8.8%	9.0%	10.6%	**12.7%**	4.9%	7.6%	2.1%	7.8%
L2S-500* [25]	NIH	9.3%	11.6%	12.4%	14.7%	17.5%	11.0%	13.0%	7.9%	12.2%
TBGAT-500* [26]	NIH	8.0%	9.9%	10.0%	13.3%	16.4%	9.6%	11.9%	6.4%	10.7%
MP-ASIL (k=512)	NCH	6.8%	**8.5%**	**8.7%**	**10.4%**	12.8%	4.2%	7.0%	**1.0%**	**7.4%**

L2D
(2020 NeuriPs)
GNN + PPO 알고리즘으로 JSSP 해결 (RL)

SN
(2023 AAAMS(Multi Agent 학회))
Multi Agent + GNN으로 JSSP 문제 해결 (RL)

RASCL
(2023 IJCAI)
Curriculum Learning을 이용한 GIN + LSTM
알고리즘으로 JSSP 문제 해결 (RL)

RS
(2023 IEEE Access)
GNN 을 이용한 RL로 JSSP 해결 (RL)

SIGD
(2024 Transactions on Machine Learning Research)
Transformer 구조 + Self Improvement Learning
Gumbeldore 알고리즘을 이용한 Sampling 기법 사용 (RL)

SLIM
(2024 NeuriPs)
GAT Encoder + Transformer Decoder
Self Labeling을 통해 학습 (SIL)

L2S-500
(2024 ICLR)
Transformer + Self Imitation Learning

TBGAT-500
(2024 UAI)
Topology-aware Bidirectional GAT 제안

해당 논문들은 NCH(Neural Constructive Heuristic)이 아닌
NIH(Neural Improvement Heuristic)으로
신경망 구축이 아닌 신경망 개선 휴리스틱
백지 상태에서 구축하는 NCH와 다르게 초기 해에서
개선하는 방식을 학습

V Experiment Result

Table 12: Experiment results on FJSSP. Symbols follow the definitions provided in Table 1.

Method	Type	10×5		20×5●		15×10 ●		20×10 ●		30×10 ●		40×10 ●	
		Gap ↓	Time ↓	Gap ↓	Time ↓	Gap ↓	Time ↓	Gap ↓	Time ↓	Gap ↓	Time ↓	Gap ↓	Time ↓
OR-Tools (1800s)*	Exact	0.00%	(50h)	0.00%	(50h)	0.00%	(50h)	0.00%	(50h)	0.00%	(50h)	0.00%	(50h)
FIFO	Heuristics	24.06%	(16s)	14.87%	(32s)	28.65%	(51s)	19.22%	(1.2m)	19.50%	(1.8m)	16.67%	(2.5m)
MOR	Heuristics	19.87%	(16s)	13.85%	(32s)	20.68%	(51s)	12.20%	(1.2m)	15.57%	(1.8m)	15.13%	(2.5m)
SPT	Heuristics	34.76%	(16s)	22.56%	(32s)	38.22%	(51s)	30.25%	(1.2m)	27.47%	(1.8m)	21.66%	(2.5m)
MWKR	Heuristics	17.58%	(16s)	11.51%	(32s)	19.41%	(51s)	10.30%	(1.2m)	13.96%	(1.8m)	13.37%	(2.5m)
HGNN (S=100)*	NCH	9.66%	(1.9m)	10.31%	(3.9m)	12.13%	(6.6m)	9.64%	(10.7m)	12.36%	(21.3m)	12.26%	(40.9m)
MCGA (S=100)*	NCH	9.01%	–	8.36%	–	11.77%	–	7.70%	–	12.44%	–	12.50%	–
RS (S=100)*	NCH	7.26%	–	7.22%	–	9.59%	–	6.06%	–	11.14%	–	11.29%	–
DANIEL (S=100)*†	NCH	5.57%	(1.2m)	2.46%	(3.1m)	6.79%	(6.5m)	-1.03%	(10.2m)	4.43%	(20.6m)	3.77%	(37.6m)
MP-ASIL (k=100)	NCH	**3.00%**	(1.2m)	**0.67%**	(3.1m)	**4.61%**	(6.5m)	**-3.00%**	(10.3m)	**-0.15%**	(20.7m)	**-0.59%**	(37.8m)

HGNN
(2023 IEEE TRANSACTION ON INDUSTRIAL INFORMATICS)
Heterogenueous GNN 으로 FJSSP 해결 (RL)

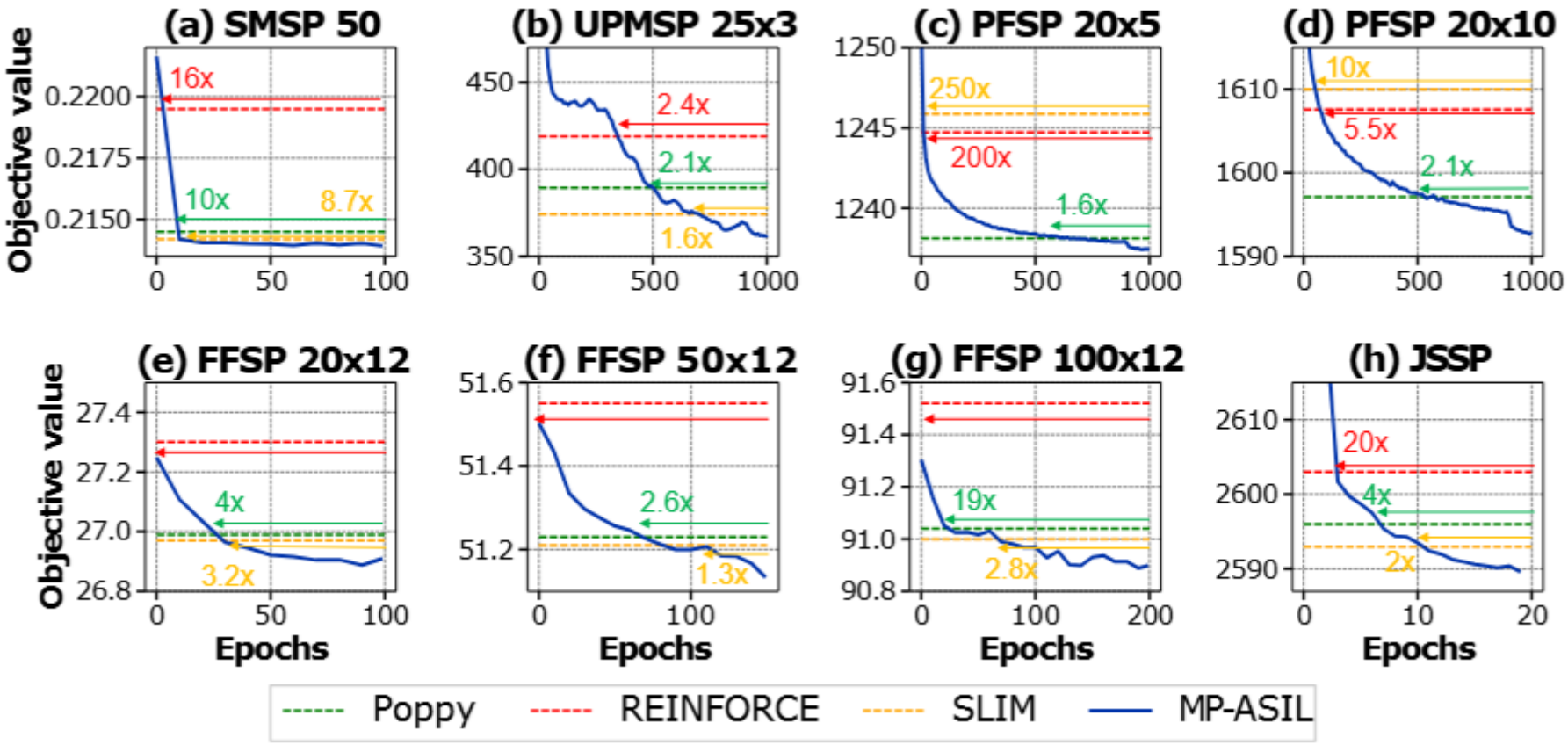
RS
(2023 IEEE Access)
GNN 을 이용한 RL로 FJSSP 해결 (RL)

MCGA
(2025 Applied Soft Computing)
Multi Channel Graph Attention으로 FJSSP 해결 (RL)

DANIEL
(2024 IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS)
Dual Attention Network 기반 GAT로 FJSSP 해결 (RL)

Experiment Result

❖ Validation Graph



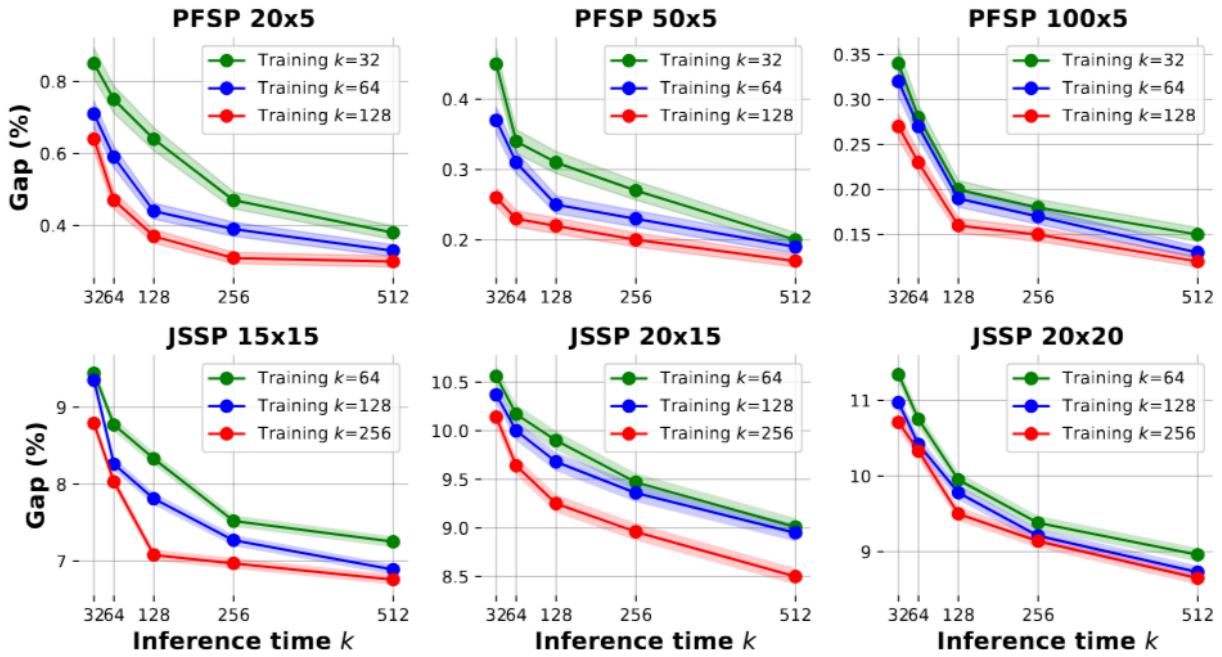
Experiment Result

❖ Result of ablation studies

AdvW	MP	SIL	SMSP 100	UPMSP 100×6	PFSP 100×10	FFSP 100×12	JSSP 100×20
✓	✓	✓	**0.00%**	**9.57%**	**2.15%**	**1.32%**	**0.96%**
✗	✓	✓	0.67% (0.67% ↑)	10.30% (0.73% ↑)	3.56% (1.41% ↑)	1.37% (0.05% ↑)	1.75% (0.79% ↑)
✗	✗	✓	3.37% (3.37% ↑)	14.33% (4.76% ↑)	3.57% (1.42% ↑)	1.40% (0.08% ↑)	2.10% (1.14% ↑)
✗	✓	✗	35.77% (35.77% ↑)	13.50% (3.93% ↑)	2.16% (0.01% ↑)	1.59% (0.27% ↑)	2.47% (1.51% ↑)
✗	✗	✗	6.69% (6.69% ↑)	20.16% (10.59% ↑)	3.78% (1.63% ↑)	2.02% (0.70% ↑)	3.96% (3.00% ↑)

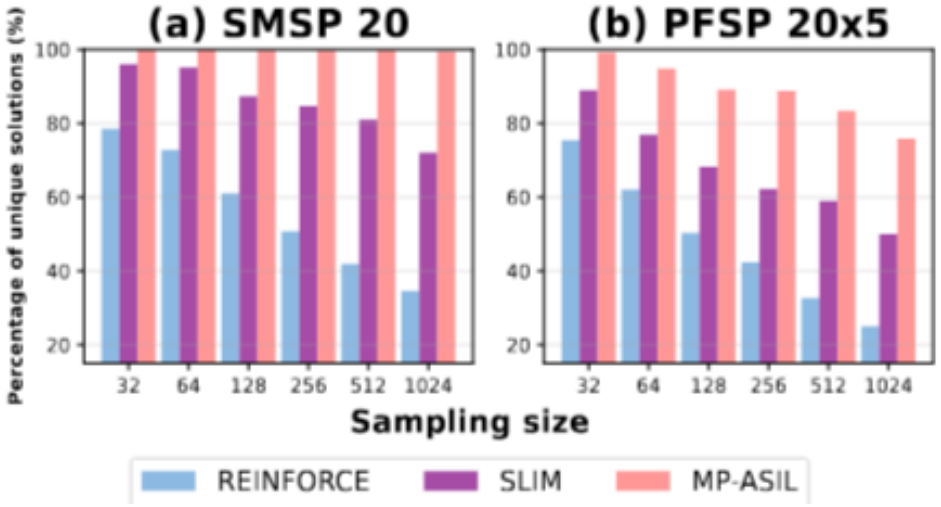
AdvW : Advantage weight
MP : Multiple polices
SIL : Self Improvement Learning

❖ The effect of k on model performance



V Experiment Result

❖ Average percentage of unique solutions



MP-ASIL이 SLIM과 REINFORCE 대비
모든 Sampling Size에서 Unique한 Solution을 가장 많이도출

❖ TSP & CVRP Result

Table 5: Experiment results on TSP 100 and CVRP 100. Poppy uses 16 decoders for TSP 100 and 32 decoders for CVRP 100. d: Days. Other symbols follow definitions provided in Table 1.

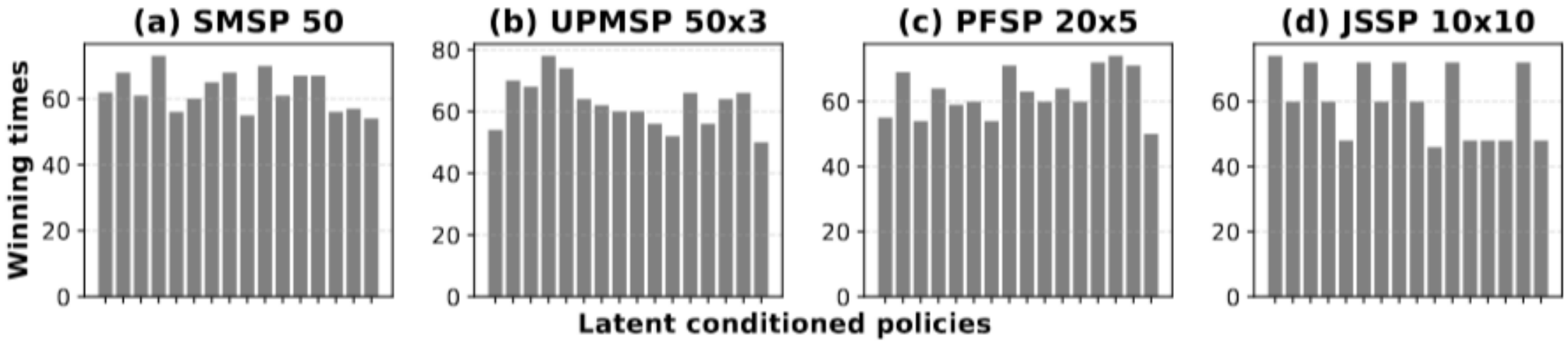
Method	TSP 100		CVRP 100	
	Gap ↓	Time ↓	Gap ↓	Time ↓
LKH3*	0.000%	(8h)	0.00%	(6d)
POMO *†	0.146%	(1m)	0.76%	(2m)
Sym-NCO *†	0.180%	(1m)	0.89%	(2m)
Poppy*	0.07%	(1m)	0.51%	(5m)
MP-ASIL	**0.000%**	(1m)	**0.28%**	(2m)

Sym-NCO
(2022 NeuriPs)
문제와 해의 대칭성을 정규화 항으로 활용해,여러 조합최적화 문제 해결
(RL)

Poppy
(2023 NeuriPs)
PolyNet이 참고한 문헌으로, 샘플링 시 Decoder을 여러 개 사용 (RL)

Experiment Result

❖ **Number of instances where each policy performs best**



Latent Conditioned Polices : 단일 신경망을 여러 개로 분기하며 잠재 변수를 사용했음. X축의 각 막대가 이러한 잠재변수들을 의미
Winning Times : 최적 솔루션 제공 횟수, 여러 Instance들에 대해 특정 잠재 조건부 정책이 다른 정책보다 더 나은 솔루션을 생성한 횟수

조금씩 높이가 다른 점이 각 정책들이 특정 문제에 맞게 “전문화” 되었음을 의미

IV Conclusion

- ✓ MP-ASIL (Multi-Policy Autonomous Self-Imitation Learning)
  작업 스케줄링 문제를 위한 범용 학습 프레임워크 제안
- ✓ 외부 감독 없이 다중 정책 학습 가능, 여러 정책이 자율적으로 서로 다른 문제 해결 전략을 학습
- ✓ 기존 강화학습 기반 정책경사 방법의 한계 극복, 단일 정책 수렴 문제 완화 및 탐색 다양성 향상
- ✓ Self-Imitation Learning(SIL)의 한계 보완, 자체 생성 레이블의 준최적성 문제를 완화하는 학습 기법 제안
  → 샘플 효율성(sample efficiency) 향상
- ✓ 모델 구조 및 문제 독립적인 프레임워크로 특정 신경망 구조나 문제 유형에 의존하지 않음
- ✓ 우수한 실험 성능
  6개 스케줄링 문제에서 새로운 SOTA 달성, 기존 신경망 기반 라우팅 솔버 대비 유의미한 성능 향상
- ✓ NCO 적용 범위 확장
  다양한 스케줄링 및 라우팅 문제로 Neural Combinatorial Optimization의 활용성 확대