Self-Labeling the Job Shop Scheduling Problem

2025. 08. 01

서세민

I Introduction

❖ 논문의 핵심 제안: SLIM (Self-Labeling Improvement Method):

✓ 지도 학습의 높은 라벨링 비용 문제를 해결하고 일반화 성능을 향상시키기 위해, 새로운 자기 지도 학습 훈련 전략인 SLIM을 도입

✓ SLIM은 대부분의 RL 알고리즘보다 단순하며, Markov Decision Process(MDP)를 명확히 정의할 필요 없이 모델 자체에서 생성된 정보만을 활용하여 값비싼 정답 데이터의 필요성을 제거함

✓ 이 방법은 두 가지 주요 가정에 기반합니다:
① 생성 모델이 여러 솔루션(Deep generative models: Survey)을 생성할 수 있다는 점
② 문제 목표(예: makespan)에 따라 솔루션을 구별할 수 있다는 점

✓ 훈련 과정에서 모델은 여러 솔루션을 샘플링하고, 문제 목표에 따라 가장 좋은 솔루션을 pseudo-label로 활용하여 모델을 반복적으로 개선함

Ⅱ Job Shop scheduling

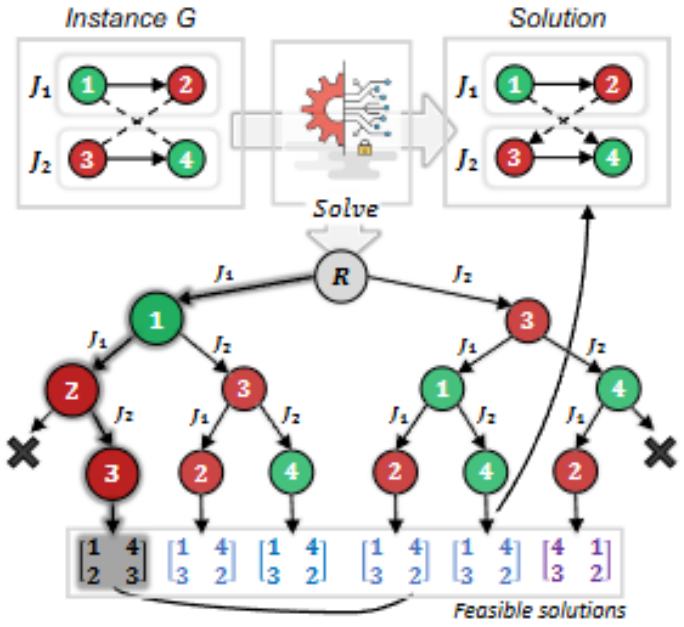
정의: 여러 대의 기계에 걸쳐 일련의 작업(jobs)을 스케줄링하는 것을 포함하며, 각 작업은 특정 순서에 따라 정해진 기계에서 처리

❖ 목표: 일반적으로 모든 작업을 완료하는 데 필요한 총 시간인 makespan($C_{\max}$)을 최소화하는 것입니다.

- ✓ 구성 요소: 작업(J): 스케줄링해야 할 n개의 작업 집합 ($J = \{J_1, \dots, J_n\}$)
- ✓ 기계(M): 작업을 처리하는 m개의 기계 집합 ($M = \{M_1, \dots, M_m\}$).
- ✓ 작업(Operations, O): 각 작업 $j \in J$는 m_j개의 연속적인 작업들 $O_j = (l_j, \dots, l_j + m_j - 1) \subset O$로 구성
- ✓ 각 작업 $i \in O$는 특정 기계 $\mu_i \in M$에서 $\tau_i \in \mathbb{R}_{\geq 0}$시간 동안 중단 없이 수행되어야 함

❖ JSP 인스턴스는 $G = (V, A, E)$형태의 이산 그래프로 표현

- ✓ V: 각 작업 $i \in O$를 나타내는 정점(vertex) 집합
- ✓ A: 작업 간의 선행 관계(precedences)를 나타내는 방향성 아크(directed arcs) 집합, 작업 내에서 작업을 수행해야 하는 순서를 반영
- ✓ E: 동일한 기계에서 수행되어야 하는 작업들을 연결하는 이산(undirected) 엣지 집합



❖ 솔루션(π):

- ✓ 그래프 G에서 모든 이산 엣지 E에 방향을 부여하여, 결과 그래프가 방향성을 가지며 순환이 없도록(all precedences are respected) 만드는 것을 의미
- ✓ 이는 각 기계에서 작업들의 순열을 결정하는 것
- ✓ $C_i(\pi)$는 솔루션 π에서 작업 i의 완료 시간을 나타냄

Ⅱ Methodology

확률 수식: $p_\theta(\pi|G) = \prod_{t=1}^{|O|} f_\theta(y_t|\pi_{<t}, G)$

$p_\theta(\pi|G)$:주어진 JSP 인스턴스 G에 대해 특정 해 π가 고품질일 확률

$|O|$: Job Shop Scheduling Problem에서 총 작업(operations)의 개수

$f_\theta(y_t|\pi_{<t}, G)$: 주어진 JSP 인스턴스 G와 t단계까지 구성된 부분 해 $\pi_{<t}$를 조건으로, 다음 단계 t에서 작업 y_t를 선택할 확률을 추정하는 모델 f_θ

- ✓ 인스턴스 G가 주어졌을 때, 모델의 파라미터 θ에 의해 생성된 솔루션 π의 확률을 나타냅니다
- ✓ 솔루션 π를 $t = 1$부터 $|O|$ (인스턴스에 포함된 전체 작업의 수)까지 구성하는 각 의사결정 단계마다의 확률을 모두 곱하여 최종 솔루션의 확률을 계산
- ✓ t번째 의사결정 단계에서 특정 작업 y_t를 선택할 확률

자기회귀(autoregressive) 방식으로 구성하는 과정을 확률적으로 모델링각 단계에서 모델은 현재까지의 부분 솔루션과 전체 문제 인스턴스를 바탕으로 다음 작업을 선택할 확률을 예측

자기 지도 학습 전략을 사용하여 이 생성 모델을 훈련하며, 이를 통해 고가의 정답 레이블(최적해) 없이도 모델의 솔루션 생성 능력을 반복적으로 개선

Generative Encoder-Decoder Architecture

❖ Pointer Network에서 영감을 받은 인코더-디코더(encoder-decoder) 아키텍처

Encoder

$$e_i = [x_i || \sigma\big(GAT_2([x_i || \sigma(GAT_1(x_i, G)\big)], G))]$$

e_i: 최종적으로 생성된 작업 i의 임베딩

x_i (table) : 작업 i의 초기 원시(raw) 특징 벡터

σ: ReLU

$GAT_n(x_i, G)$: Graph Attention Network (GAT) 계층

||: 벡터 연결(concatenation)

- ✓ 전체 인스턴스를 나타내는 disjunctive graph G를 입력으로 받음
- ✓ 인코더는 입력된 G와 x_i를 단 한 번의 순방향 계산(single forward computation)으로 처리함- 논문은 Graph Attention Networks (GAT)와 같은 Graph Neural Networks (GNNs)를 사용
- ✓ 인코더는 모든 작업(operation)에 대한 임베딩(embedding) $e_i \in \mathbb{R}^h$를 생성
- ✓ 전체 문제 상황에 대한 정적인 요약 정보를 제공

ID	Description
1	The processing time τ_i of the operation.
2	The completion of job j up to i: $\sum_{b=l_j}^{i} \tau_b \,/\, \sum_{b \in O_j} \tau_b$.
3	The remainder of job j after i: $\sum_{b=i+1}^{l_j+m_j-1} \tau_b \,/\, \sum_{b \in O_j} \tau_b$.
4-6	The 1^{st}, 2^{nd}, and 3^{rd} quartile among processing times of operations on job j.
7-9	The 1^{st}, 2^{nd}, and 3^{rd} quartile among processing times of operations on machine μ_i.
10-12	The difference between τ_i and the 1^{st}, 2^{nd}, and 3^{rd} quartile among processing times of operations in job j.
13-15	The difference between τ_i and the 1^{st}, 2^{nd}, and 3^{rd} quartile among processing times of operations on machine μ_i.

Ⅱ Methodology

Decoder

$$s_j = RELU([c_j W_1 + MHA_{b \in J}(c_b W_1)]W_2)$$

$$z_j = FNN([e_{o(t,j)} \parallel s_j])$$

$$p_j = \frac{e^{z_j}}{\Sigma_{b \in J} e^{z_b}}$$

ID	Description	Rationale
1	$C_{o(t,j)-1}(\pi_{<t})$ minus the completion time of machine $\mu_{o(t,j)}$.	The idle time of job j if scheduled on its next machine $\mu_{o(t,j)}$ at time t. Negative if the job could have started earlier.
2	$C_{o(t,j)-1}(\pi_{<t})$ divided by the makespan of $\pi_{<t}$.	How close job j is to the makespan of the partial solution $\pi_{<t}$.
3	$C_{o(t,j)-1}(\pi_{<t})$ minus the average completion time of all jobs.	How early/late job j completes w.r.t. the average among all jobs in $\pi_{<t}$.
4 - 6	The difference between $C_{o(t,j)-1}(\pi_{<t})$ and the 1st, 2nd, and 3rd quartile computed among the completion time of all jobs.	Describe the relative completion of j w.r.t. other jobs. These features complement that with ID = 3.
7	The completion time of machine $\mu_{o(t,j)}$ divided by the makespan of the partial solution $\pi_{<t}$.	How close the completion time of machine $\mu_{o(t,j)}$ is to the makespan of the partial solution $\pi_{<t}$.
8	The completion time of machine $\mu_{o(t,j)}$ minus the average completion of all machines in $\pi_{<t}$.	How early/late machine $\mu_{o(t,j)}$ completes w.r.t. the average among machines in $\pi_{<t}$.
9 - 11	The difference between the completion of $\mu_{o(t,j)}$ and the 1st, 2nd, and 3rd quartile among the completion time of all machines.	Describe the relative completion time of machine $\mu_{o(t,j)}$ w.r.t. all the other machines. These features complement that with ID = 8.

❖ 디코더는 두 가지 주요 정보를 입력:
- ✓ 인코더가 생성한 작업 임베딩 $e_{o(t,j)}$
- ✓ 이는 인코더가 파악한 전체 인스턴스 정보 중 현재 의사결정에 관련된 부분을 가져옴

❖ 부분 솔루션 $\pi_{<t}$에서 추출한 컨텍스트 벡터 c_j:
- ✓ 현재까지 구축된 부분 솔루션 $\pi_{<t}$ 의 동적인 상태 정보를 반영

❖ 디코더는 Memory Network와 Classifier Network의 두 가지 구성 요소를 통해 의사결정을 수행:
- ✓ Memory Network:
  - 컨텍스트 벡터 c_j를 사용하여 각 작업 j의 현재 상태 s_j를 생성
  - 이때 Multi-Head Attention (MHA)을 활용하여 모든 작업의 컨텍스트를 종합적으로 고려

- ✓ Classifier Network:
  - 인코더의 출력인 작업 임베딩 $e_{o(t,j)}$ 와 Memory Network의 출력인 작업 상태 s_j 를 결합하여 현재 단계에서 각 작업 j 를 선택할 확률 z_j를 계산

❖ 출력:
- ✓ Classifier Network에서 계산된 각 작업 j의 스코어 z_j를 바탕으로, 해당 작업이 다음 스케줄링 단계에서 선택될 확률 p_j를 계산하기 위해 softmax 함수를 사용

SLIM

$$L_{SL}(\bar{\pi}) = -\frac{1}{|O|}\sum_{t=1}^{|O|}\log f_\theta(y_t|\pi_{<t}, G)$$

$L_{SL}(\bar{\pi})$: Self-Labeling 손실

$\bar{\pi}$: f_θ가 샘플링한 여러 솔루션 중에서 가장 우수한 솔루션

$|O|$: 인스턴스에 포함된 총 작업의 수

- ✓ 주어진 훈련 인스턴스 G에 대해, 생성 모델 f_θ는 여러 개의 솔루션(논문에서는 β개)을 병렬로 생성
- ✓ 생성된 이 β개의 솔루션 중에서 문제의 목표(makespan 최소화)에 따라 가장 우수한 솔루션을 하나 선택
- ✓ 가장 우수한 솔루션이 바로 $\bar{\pi}$이며, 이것이 이 훈련 단계의 pseudo-label이 됨
- ✓ 수식은 본질적으로 크로스 엔트로피 손실이며, 모델 f_θ가 의사 레이블 $\bar{\pi}$를 구성하는 각 결정 단계 t에서 선택된 작업 y_t에 대해 부여한 확률의 로그 값을 최대화

계산된 손실$L_{SL}(\bar{\pi})$을 사용하여 경사 하강법(gradient descent) 기반의 최적화 알고리즘(논문에서는 Adam optimizer 사용)을 통해 모델의 매개변수 θ를 업데이트 합니다.

기존 Cross-Entropy Loss

❖ 레이블(정답)의 출처:
각 JSP 인스턴스에 대해, 가장 Makespan이 짧은 최적의 스케줄은 사람이 직접 손으로 만들거나, 매우 강력하고 계산 비용이 많이 드는 '정확한 솔버(Exact Solver)'를 오랜 시간 동안 돌려서 얻어낸 것

❖ 정답 스케줄: 인스턴스 G1의 최적 스케줄이 미리 "작업 A → 작업 C →작업 B" 순서로 결정되어야 가장 Makespan이 짧다고 가정

❖ 모델의 예측: 모델은 G1을 보고 첫 번째 결정을 내림
e.g 모델은 "작업 A를 선택할 확률 0.7, 작업 B를 선택할 확률 0.2, 작업 C를 선택할 확률 0.1"이라고 예측

❖ 손실 계산:
- ✓ 정답은 '작업 A', 모델이 '작업 A'에 0.7의 확률을 부여했으니, 이는 '정답'인 '작업 A'에 높은 확률을 부여했지만, 여전히 1.0이 아니므로 손실이 발생
- ✓ 만약 모델이 '작업 C'에 가장 높은 확률을 부여했다면, 정답인 '작업 A'에는 낮은 확률을 부여했으므로 손실이 훨씬 커짐

업데이트: 이 손실을 줄이는 방향으로 모델의 매개변수를 업데이트
목표는 모델이 G1과 같은 상황에서 항상 '작업 A'에 가장 높은 확률을 부여
그 다음 단계에서도, 정답 스케줄이 지시하는 '작업 C'에 높은 확률을 부여하도록 학습

SLIM의 Self-Labeling Loss

❖ 모델이 현재까지 학습한 지식으로 스스로 여러 개의 JSP 스케줄을 생성

❖ 그렇게 생성된 여러 스케줄 중에서 Makespan이 가장 짧게 나온 스케줄을 '가상의 정답(Pseudo-Label, $\bar{\pi}$)'으로 선정

❖ 모델은 G1 인스턴스를 받습니다. $\beta = 3$이라고 가정하고 3개의 스케줄을 생성

✓ 스케줄 1: "작업 B →작업 A →작업 C" (Makespan = 120)
✓ 스케줄 2: "작업 A →작업 C →작업 B" (Makespan = 90)
✓ 스케줄 3: "작업 A →작업 B → 작업 C" (Makespan = 105)

이제 모델은 선정된 의사 레이블 $\bar{\pi}$ ("작업 A →작업 C →작업 B")를 '모범 답안'처럼 사용

❖ 모델은 $\bar{\pi}$ 를 만들 때, 각 결정 단계에서 실제로 선택했던 작업에 대해 부여했던 확률을 측정
1단계: '작업 A'를 선택. 모델은 이때 '작업 A'에 $P_1(\{\text{작업 } A\})$작업 확률을 부여
2단계: '작업 C'를 선택. 모델은 이때 '작업 C'에 $P_2(\{\text{작업 } C\})$작업 확률을 부여
3단계: '작업 B'를 선택. 모델은 이때 '작업 B'에 $P_3(\{\text{작업 } B\})$작업 확률을 부여

SLIM 손실은 $-\log P_1(\{\text{작업 } A\}) - \log P_2(\{\text{작업 } C\}) - \log P_3(\{\text{작업 } B\})$와 같은 형태로 계산

업데이트: 이 손실을 줄이는 방향으로 모델의 매개변수를 업데이트
목표는 모델이 G1과 같은 상황에서 '작업 A → 작업 C → 작업 B'와 같이 스스로 찾아낸 고품질 스케줄을 다시 생성할 가능성을 높이는 것

Ⅲ Results

❖ 운영체제:
- ✓ Ubuntu 22.04CPU: Intel Core i9-11900KGPU: NVIDIA GeForce RTX 3090 (24GB)

❖ 훈련 데이터셋:
- ✓ 모델 훈련을 위해 30,000개의 인스턴스로 구성된 데이터셋을 생성
- ✓ 데이터셋은 다음과 같은 크기(작업 수 × 기계 수)의 인스턴스를 각 5,000개씩 무작위로 생성: $(\{10 \times 10, 15 \times 10, 15 \times 15, 20 \times 10, 20 \times 15, 20 \times 20\})$

❖ 테스트 벤치마크:
- ✓ 모델의 성능을 평가하고 다른 연구 결과와 비교하기 위해 Taillard's benchmark, Demirkol's benchmark 벤치마크 당 80개 인스턴스(각 크기당 10개 인스턴스)

Ⅱ Results

다양한 인스턴스에 대해 여러 스케줄링 알고리즘의 성능을 비교

Table 2: The average PG of the algorithms on the benchmarks. In each row, we highlight in **blue** (**bold**) the best constructive (non-constructive) gap. Shapes marked with * are larger than those seen in training by our GM.

		Greedy Constructives							*Multiple (Randomized) Constructives*								*Non-constructive Approaches*				
		PDRs				RL		Our	$PDRs_{\beta=128}$				$RL_{\beta=128}$		Our						
	Shape	SPT	MWR	MOR	INSA	L2D	CL	GM	SPT	MWR	MOR	SBH	L2D	CL	GM_{128}	GM_{512}	$L2S_{500}$	NLS_A	$L2S_{5k}$	MIP	CP
Taillard's benchmark	15 × 15	26.1	19.1	20.4	14.4	26.0	14.3	13.8	13.5	13.5	12.5	11.6	17.1	9.0	7.2	**6.5**	9.3	7.7	6.2	**0.1**	**0.1**
	20 × 15	32.3	23.3	24.9	18.9	30.0	16.5	15.0	18.4	17.2	16.4	10.9	23.7	10.6	9.3	**8.8**	11.6	12.2	8.3	3.2	**0.2**
	20 × 20	28.3	21.8	22.9	17.3	31.6	17.3	15.2	16.7	15.6	14.7	13.6	22.6	10.9	10.0	**9.0**	12.4	11.5	9.0	2.9	**0.7**
	30 × 15*	35.0	24.1	22.9	21.1	33.0	18.5	17.1	23.2	19.0	17.3	15.1	24.4	14.0	11.0	**10.6**	14.7	14.1	9.0	10.7	**2.1**
	30 × 20*	33.4	24.8	26.8	22.6	33.6	21.5	18.5	23.6	19.9	20.4	17.7	28.4	16.1	13.4	**12.7**	17.5	16.4	12.6	13.2	**2.8**
	50 × 15*	24.0	16.4	17.6	15.9	22.4	12.2	10.1	14.1	13.5	13.5	13.2	17.1	9.3	5.5	**4.9**	11.0	11.0	4.6	12.2	**3.0**
	50 × 20*	25.6	17.8	16.8	20.3	26.5	13.2	11.6	17.6	14.6	14.0	20.8	20.4	9.9	8.4	**7.6**	13.0	11.2	6.5	13.6	**2.8**
	100 × 20*	14.0	8.3	8.7	13.5	13.6	5.9	5.8	10.4	7.0	7.1	15.6	13.3	4.0	2.3	**2.1**	7.9	5.9	**3.0**	11.0	3.9
	Avg	27.4	19.5	20.1	18.0	27.1	14.9	13.4	17.2	15.0	14.5	14.8	20.5	10.5	8.4	**7.8**	12.2	11.3	7.4	8.4	**2.0**
Demirkol's benchmark	20 × 15	27.9	27.5	30.7	24.2	39.0	-	18.0	17.2	22.3	23.8	12.5	29.3	19.4	12.0	**11.3**	-	-	-	5.3	**1.8**
	20 × 20	33.2	26.8	26.3	21.3	37.7	-	19.4	18.8	18.9	21.6	13.5	27.1	16.0	13.5	**12.3**	-	-	-	4.7	**1.9**
	30 × 15*	31.2	31.9	36.9	26.5	42.0	-	21.8	20.7	26.8	30.7	18.2	34.0	16.5	14.4	**14.0**	-	-	-	14.2	**2.5**
	30 × 20*	34.4	32.1	32.3	28.5	39.7	-	25.7	23.3	26.1	28.3	17.3	33.6	20.2	17.1	**15.8**	-	-	-	16.7	**4.4**
	40 × 15*	25.3	27.0	35.8	24.7	35.6	-	17.5	17.9	23.2	30.2	14.4	31.5	17.6	11.7	**10.9**	-	-	-	16.3	**4.1**
	40 × 20*	33.8	32.3	35.9	29.4	39.6	-	22.2	24.5	27.6	31.8	20.3	35.8	25.6	16.0	**14.8**	-	-	-	22.5	**4.6**
	50 × 15*	24.3	27.6	34.9	23.0	36.5	-	15.7	17.7	24.1	31.0	18.2	32.7	21.7	11.2	**10.6**	-	-	-	14.9	**3.8**
	50 × 20*	30.0	30.3	36.8	30.2	39.5	-	22.4	23.5	26.8	32.7	22.8	36.1	15.2	15.8	**15.0**	-	-	-	22.5	**4.8**
	Avg	30.0	29.4	33.7	26.0	38.7	-	20.3	20.4	24.5	28.8	17.2	32.5	19.0	14.0	**13.1**	-	-	-	14.6	**3.5**

각 벤치마크 인스턴스에 대한 성능은 Percentage Gap (PG)으로 측정

$PG = 100 \cdot (\frac{C_{alg}}{C_{ub}-1})$

C_{alg} 는 알고리즘이 생성한 makespan

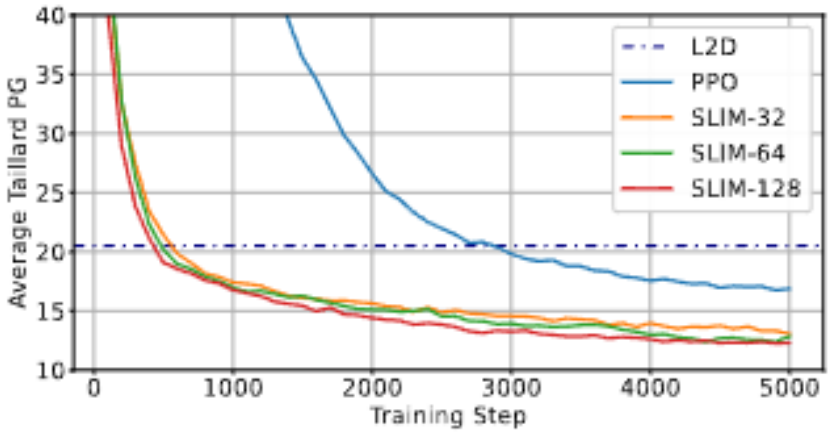
C_{ub} 는 해당 인스턴스의 최적 또는 가장 잘 알려진 makespan

Ⅱ Results

다양한 알고리즘의 평균 PG(Percentage Gap) 성능을 비교한 결과

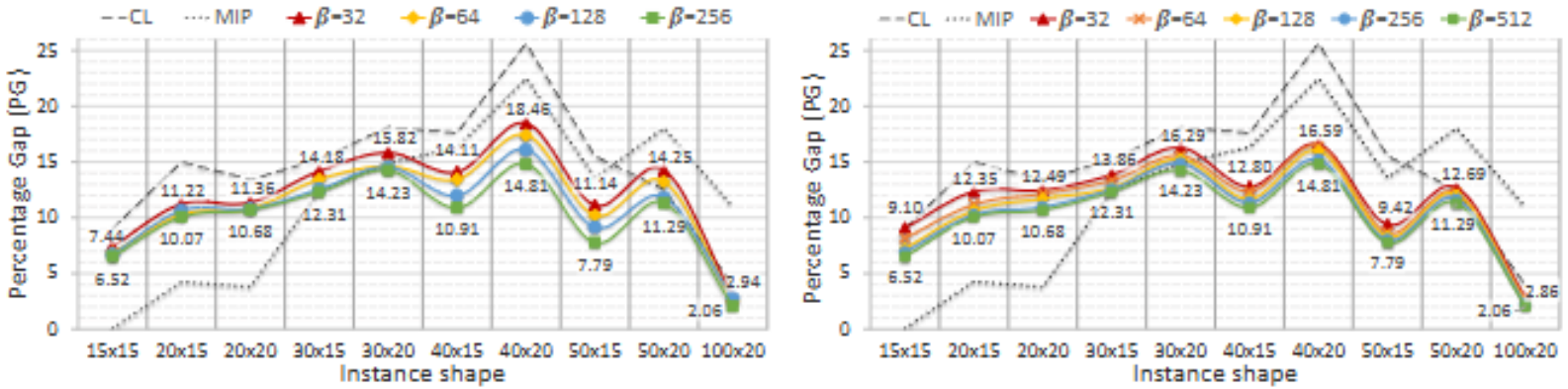
Benchmark	RL		Self-Labeling		
	CL_{UCL}	CL	CL_{SL}	FNN_{SL}	GM
Taillard	17.3	10.5	10.7	9.5	8.4
Demirkol	-	19.0	20.9	14.9	14.0

생성 모델(GM)을 다양한 방식으로 훈련했을 때의 검증 성능 변화



Ⅱ Results

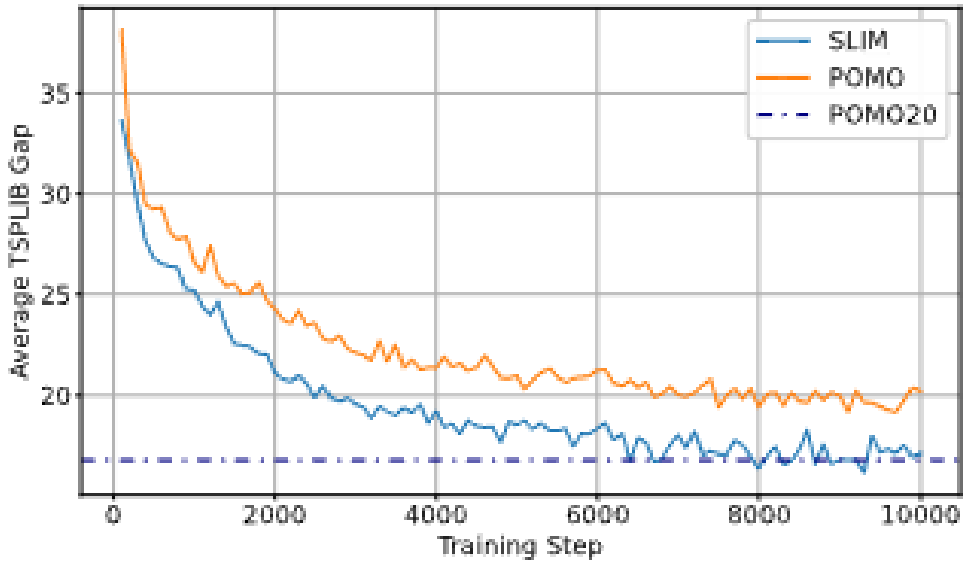
Generative Model(GM)의 성능을 다양한 조건에서 비교



β: 훈련 과정에서 생성 모델이 각 훈련 인스턴스에 대해 병렬적으로 샘플링하는 솔루션의 개수
β 값을 32, 64, 128, 256으로 다르게 설정

Ⅱ Results

TSP에 대한 SLIM (Self-Labeling Improvement Method)과 POMO (Policy Optimization with Multiple Optima) 두 가지 학습 방법의 성능을 비교



Ⅱ Conclusion

- JSP를 위한 고품질 솔루션을 신속하게 병렬로 생성할 수 있는 효율적인 인코더-디코더 아키텍처를 제안
- 단순함에도 불구하고, 이 방법론은 JSP의 많은 구축(constructive) 및 학습(learning) 알고리즘보다 뛰어난 성능을 보였음
- 최적화 정보나 Markov Decision Process의 정확한 공식화가 필요 없다는 점에서 가치 있는 훈련 전략이며, 특히 비전통적인 목적 함수를 가진 조합 문제에 유용