Multi-Action Self-Improvement for Neural Combinatorial Optimization

26. 01. 02

노정빈

I Introduction

❖ **조합 최적화(Combinatorial Optimization, CO)란?**

- 가능한 수많은 조합 중 가장 좋은 최적의 조합을 찾는 문제
- 예: TSP, JSSP, VRP 등
- 문제의 크기가 커질수록 컴퓨터로도 답을 찾기 매우 어려운 NP-hard 문제

❖ **신경 조합 최적화(Neural Combinatorial Optimization, NCO)**

- 딥러닝을 사용하여 CO 문제를 해결하는 분야
- 강화 학습 기반 NCO: 시행착오를 통해 스스로 문제 해결 전략을 찾아내는 방식

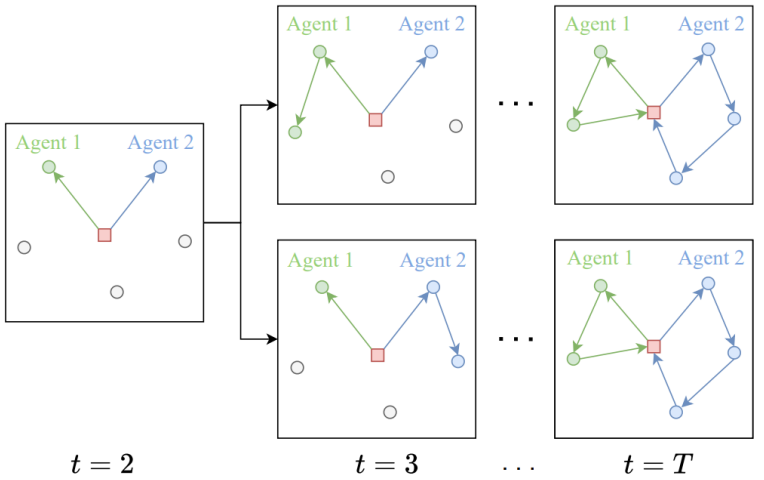
❖ **Self-Improvement**

- 강화 학습의 한 형태로, 최근 NCO 분야에서 주목받는 패러다임
- 스스로 솔루션을 생성하고, 그 중 가장 좋은 솔루션을 모범 답안으로 삼아 자신을 개선하는 방식
- 여러 후보 솔루션 생성 → 가장 좋은 솔루션(모범 답안) 선택 → 모범 답안으로부터 학습 → 반복을 통해 성능 개선

II Limitations & Solution

❖ 기존 self-improvement 방식의 한계점

- ✓ 높은 훈련 비용
  - 하나의 모범 답안을 얻기 위해 수많은 후보 솔루션을 생성해야 함
- ✓ Multi-agent 문제의 “순열 대칭성(permutation symmetries)”을 활용하지 못함
  - Agent: 차량 경로 문제에서는 차량, 스케줄링 문제에서는 기계를 의미
  - 순열 대칭성: 서로 다른 행동 순서가 동일한 솔루션을 만들어내는 것
  - 항상 단일의 최적 솔루션을 선택한 뒤 다시 다음 행동을 선택하기 때문에 순서를 강제하고, 나머지 대칭적인 선택을 오류로 취급
  - 이는 샘플 효율성을 감소시키고 일반화 성능을 저하시킴



❖ MACSIM: Multi-action Self-improvement Method

- ✓ 핵심 아이디어
  - 대칭성을 활용하도록 기존 self-improvement를 확장
- ✓ 해결책
  - 각 결정 단계에서 모든 에이전트에 대한 작업 할당을 동시에 예측 → 솔루션 생성 단계 감소 및 속도 가속화
  - 에이전트-작업 할당을 순서에 무관한 집합으로 간주하여 학습 → 대칭성 활용

Ⅲ Methodology: 1) Multi-agent Policy

❖ Multi-agent Policy

✓ 역할

- 현재 상태(s_t)로부터 모든 에이전트-작업 할당에 대한 로짓(logit) 행렬 L을 생성
- L의 각 항목 $L_{m,v}$는 에이전트 m이 작업 v를 할당 받을 '적합성 점수'를 나타냄

✓ 구조

- Input Embedding: 이분 그래프(bipartite graph) $G = \{V, M, \mathcal{E}, w\}$를 임베딩
- Encoder: 여러 계층의 attention을 통해 최종 임베딩 생성
- Decoder: 최종 임베딩을 사용하여 로짓 행렬 계산

V: 작업 집합

M: 에이전트 집합

$\mathcal{E}$: 엣지(edge). 가능한 에이전트-작업 할당

w: 엣지의 가중치(비용)

❖ Input Embedding

- 주어진 이분 그래프로부터 각 에이전트와 작업의 특징을 추출하여 차원 d의 공통 임베딩 공간으로 투영
- 에이전트 임베딩: $H_M^0 \in \mathbb{R}^{M \times d}$
- 작업 임베딩:$H_V^0 \in \mathbb{R}^{N \times d}$

Ⅲ Methodology: 1) Multi-agent Policy

❖ Encoder

- 다층 트랜스포머(Multi-Layer Transformer) 구조 기반
- 여러 계층의 self-attention과 cross-attention
- 이를 통해 에이전트 및 작업 임베딩 간의 복합적인 상호작용과 관계를 학습하여 최종 임베딩 생성
- Cross-attention: 에이전트 및 작업 임베딩 서로 간의 관계를 파악
  - ① 에이전트를 쿼리(Q), 작업을 키(K)로 사용하여 초기 어텐션 점수(attention score) 행렬 A를 계산

$$A = \frac{QK^T}{\sqrt{d_k}}, \qquad Q = W^Q H_M^{l-1}, \qquad K = W^K H_V^{l-1}$$

  - ② 어텐션 점수와 가중치 행렬(E)을 결합하여 다중 은닉층 인공신경망(Multi-Layer Perceptron, MLP)을 통해 에이전트-작업의 최종 호환성 점수를 생성

$$A_{M\to V} = \mathrm{MLP}_M([A||E]), \qquad A_{V\to M} = \mathrm{MLP}_V([A^T||E^T])$$

  - ③ 이 점수들을 활용하여 에이전트 임베딩 H'_M 및 작업 임베딩 H'_V으로 업데이트
  - ④ 스킵 연결(skip connections)과 계층 정규화(layer normalization) 적용 후 self-attention layer로 전달
- Self-attention: 업데이트된 임베딩 H'_M와 H'_V의 각 내부 관계를 파악하여 현재 계층 l에 대한 최종 임베딩 H_M^l 및 H_V^l를 생성

Methodology: 1) Multi-agent Policy

❖ Decoder

- 인코더에서 생성된 최종 임베딩을 사용하여 행동 선택을 위한 로짓 행렬(L) 계산
- multiple-pointer 메커니즘을 사용

$$L = c \cdot \tanh\left(\frac{QK^T}{\sqrt{d}}\right), \qquad Q = W_{Dec}^{Q} H_M, \qquad K = W_{Dec}^{K}\left(H_v || h^{skip}\right)$$

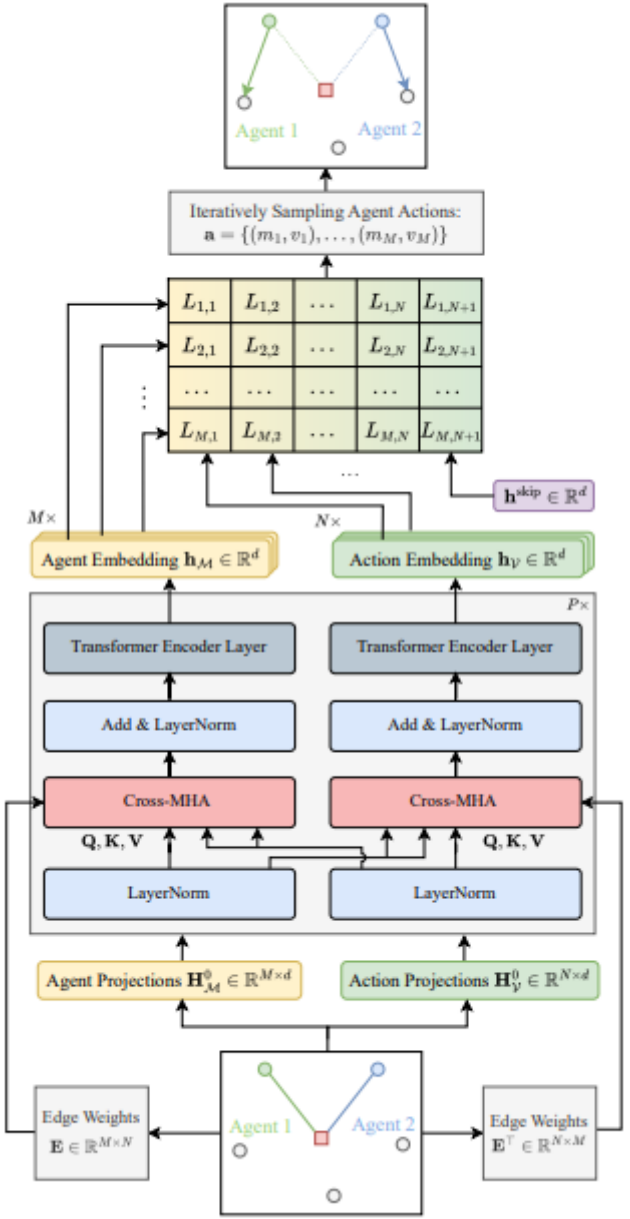
- c: 탐색을 강화하기 위한 스케일 파라미터
- h^{skip}: 스킵 토큰

$L_{1,1}$	$L_{1,2}$	$\cdots$	$L_{1,N}$	$L_{1,N+1}$
$L_{2,1}$	$L_{2,2}$	$\cdots$	$L_{2,N}$	$L_{2,N+1}$
$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$
$L_{M,1}$	$L_{M,2}$	$\cdots$	$L_{M,N}$	$L_{M,N+1}$

<로짓 행렬>

Ⅲ Methodology: 1) Multi-agent Policy

❖ Policy 전체 구조



Ⅲ Methodology: 2) Multi-action Generation

❖ Multi-action Generation

✓ 역할

- 로짓 행렬을 바탕으로, 유효하고 충돌 없는 전체 에이전트-작업 할당을 구체적으로 생성
- 이 과정은 솔루션 생성 속도를 가속화함

✓ 과정

① Policy에서 생성된 로짓 행렬을 입력으로 받음

$$L \in \mathbb{R}^{M \times N}$$

② 자동 회귀 샘플링(Autoregressive Sampling)

- 각 단계(k)에서 현재 사용 가능한(아직 할당되지 않은) 에이전트-작업 쌍을 순차적으로 하나씩 샘플링
- 샘플링된 에이전트와 작업은 다음 샘플링($k+1$)에서 제외(비복원 추출) → 다른 에이전트가 같은 작업을 선택하는 충돌 방지
- 이 과정은 확률의 연쇄 법칙(chain rule)에 따라 표현됨

$$P(a_t|L) = \prod_{k=1}^{M} P(m_{t,k}, v_{t,k}|a_t^{<k}, L)$$

a_t: t 시점에서의 전체 에이전트-작업 쌍의 집합

a_t^k: t 시점에서 k번째 에이전트에 대한 에이전트-작업 쌍

③ 에이전트-작업 쌍이 샘플링되면, 해당 에이전트와 작업은 다음 단계에서 선택되지 않도록 로짓 행렬에서 $-\infty$로 마스킹

Ⅲ Methodology: 2) Multi-action Generation

❖ Multi-action Generation 알고리즘

Algorithm 1 Joint Action Sampling from $\mathbf{L}$

Require: Score matrix $\mathbf{L} \in \mathbb{R}^{M \times N}$, feasible pairs $\mathcal{E}$
Ensure: $\mathbf{a} = [(m_1, v_1), \ldots, (m_M, v_M)]$

$\mathbf{a} \leftarrow [\,]$ ▷ Initialize empty sequence
for $k = 1$ to M **do**
  $P(m, v) \leftarrow \underset{(m,v)\in\mathcal{E}}{\text{softmax}}(L_{m,v})$ ▷ Normalize
  $(m_k, v_k) \sim P$ ▷ Sample from joint dist.
  $\mathbf{a} \leftarrow \mathbf{a} \,\|\, (m_k, v_k)$ ▷ Append
  $\mathbf{L}_{m_k,:} \leftarrow -\infty$; $\mathbf{L}_{:,v_k} \leftarrow -\infty$ ▷ Mask
end for
return $\mathbf{a}$

softmax: $P\left(a_t^k = (m_k, v_k) \middle| a_t^k, L\right) = \dfrac{\exp(L_{m_k,v_k})}{\sum_{(m',v')\in\mathcal{E}_k} \exp(L_{m',v'})}$

Ⅲ Methodology: 3) Skip Token

❖ Skip Token

✓ 역할

- 모든 에이전트에게 작업을 강제 할당하는 것이 최적이 아닐 수 있음
- 이를 해결하기 위해 스킵 토큰(skip token)이라고 불리는 '더미(dummy) 행동'을 도입
- 현재 솔루션을 수정하지 않고 다음 결정 단계까지 기다릴 수 있는 유연성 제공

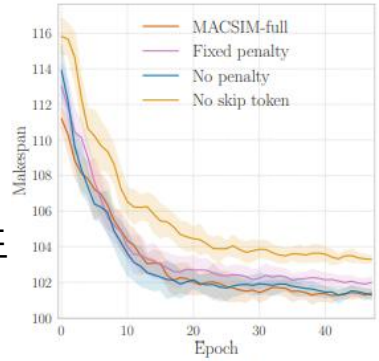
✓ 구현

- 모든 작업 임베딩(H_v) 집합에 추가되는 학습 가능한 임베딩으로 구현

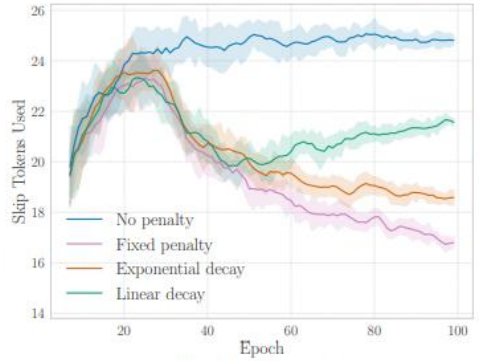
$$h^{skip} \in \mathbb{R}^d$$

- 스킵 토큰 사용에 대해 '페널티(Penalty)'가 부과되어 불필요한 스킵 토큰 사용을 방지
- 페널티는 훈련 과정에서 점진적으로 감소(annealing)함
- 훈련 초기에는 스킵 토큰을 유연하게 사용하여 탐색
- 훈련 후반에는 페널티를 통한 정규화 효과를 유지하며
  최종적으로는 스킵을 적게 사용하면서도 고품질의 답을 찾도록 유도

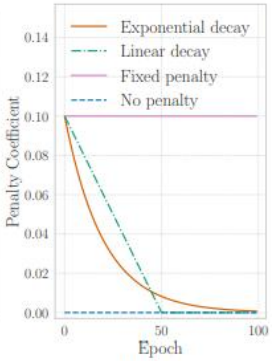
FJSP				
$N \times M$	10×5		15×10	
Metric	Obj.	Time	Obj.	Time
MACSIM-full (*s.*)	**97.64**	0.86	**145.95**	6.19
w/o AR-sampling fixed (*s.*)	98.97	0.76	155.13	3.97
random (*s.*)	98.66	0,78	150.72	4.15
w/o skip-token (*s.*)	98.69	**0.75**	159.65	**3.89**



(a) FJSP 10×5



(b) FFSP 50×4



(c) FFSP 50×4

Ⅲ Methodology: 4) Policy Learning

❖ Policy Learning

✓ 역할

- Multi-agent 정책의 매개변수 θ를 업데이트하여, 솔루션을 생성하는 능력 향상

✓ 훈련 방식: 2단계 self-improvement 프레임워크

① 모범 답안 생성

- 현재 가장 성능이 좋은 정책 네트워크 π_θ^*를 사용하여 각 문제 인스턴스(x)에 대해서 여러($\beta \gg 1$) 개의 후보 솔루션 생성
- 스킵 토큰 사용에 대한 페널티를 목적 함수 값에 더한 후 가장 좋은(비용 최소) 최적해(τ^*) 선택
- τ^*에 해당하는 상태-행동 쌍 $((s_1, a_1^*), (s_2, a_2^*), \ldots, (s_T, a_T^*))$을 훈련 데이터셋에 추가

② 모방(Imitation) 학습

- 정책 네트워크(π_θ)는 모범 답안에 대한 모방 학습을 통해 업데이트

Ⅲ Methodology: 4) Policy Learning

❖ 손실 함수(Loss Function)

✓ 기존 손실 함수의 문제점

- 단일 행동 기반($\mathcal{L}_{LA}$, $\mathcal{L}_{ML}$): 단일 에이전트-작업 쌍을 예측하고 대칭성을 무시하여 기울기 충돌(conflicting gradient) 및 학습 신호 소실(vanishing gradient) 문제 발생

✓ 해결책: Cross-Entropy(CE) 기반 손실 함수

- 모범 답안의 각 에이전트-작업 쌍을 독립적인 지도 학습 인스턴스로 간주
- 에이전트 순서에 관계없이 각 에이전트의 선택을 평가하며, 동시에 여러 정답을 선택하는 것에 대해 페널티를 부과하지 않음

$$\mathcal{L}_{CE} = -\sum_{k=1}^{M} \log P(v_k|m_k) = -\sum_{k=1}^{M} \log \frac{\exp(L_{m_k,v_k})}{\sum_{v' \in V} \exp(L_{m_k,v'})}$$

- 본 논문에서는 $\mathcal{L}_{CE}$가 대칭성을 활용하는 목적 함수에 대한 계산 가능한 상한선임을 증명함
- 정책의 신뢰도가 높아짐에 따라 상한이 좁혀지며, 후반 훈련에서 이상적인 목적 함수에 근사

	FJSP		**FFSP**	
$N \times M$	10×5	20×5	20×12	50×12
Metric	Obj.	Obj.	Obj.	Obj.
$\mathcal{L}_{\text{SA}}$ ($s.$)	98.81	189.18	24.29	49.58
$\mathcal{L}_{\text{ML}}$ ($s.$)	98.13	188.97	24.15	49.02
$\mathcal{L}_{\text{PL}}$ ($s.$)	97.99	188.81	24.08	47.90
$\mathcal{L}_{\text{CE}}$ ($s.$)	**97.64**	**188.10**	**23.96**	**47.74**

IV Experiments

❖ **실험 문제**

- 유연 잡샵 스케줄링 문제(Flexible Job Shop Problem, FJSP)
- 유연 플로우샵 스케줄링 문제(Flexible Flow Shop Problem, FFSP)
- 최소-최대 이종 용량 제한 차량 경로 문제(Min-Max Heterogeneous Capacitated Vehicle Routing Problem, HCVRP)

❖ **비교 대상**

- 상용 최적화 솔버: OR-Tools, Gurobi
- 기존 휴리스틱
- 최신 NCO baseline: SLIM, PARCO 등

IV Experiments

❖ 실험 결과

FJSP

$N \times M$	10×5			20×5			15×10		
Metric	Obj.	Gap	Time (s.)	Obj.	Gap	Time (s.)	Obj.	Gap	Time (s.)
OR-Tools	**96.32**	0.00%	1597	188.15	0.03%	1800	**143.53**	0.00%	1724
FIFO	119.4	23.96%	0.16	216.08	14.88%	0.32	184.55	28.58%	0.51
MOR	115.38	19.79%	0.16	214.16	13.85%	0.32	173.15	20.64%	0.51
MWKR	113.23	17.56%	0.16	209.78	11.53%	0.32	171.25	19.31%	0.50
PPO (*g.*)	111.67	15.94%	0.45	211.22	12.29%	1.43	166.92	16.30%	1.35
DANIEL (*g.*)	106.71	10.79%	0.45	197.56	5.03%	0.94	161.28	12.37%	1.43
SLIM (*g.*)	103.85	7.82%	0.91	194.37	3.33%	1.18	154.32	7.62%	2.74
MACSIM (*g.*)	102.21	6.12%	0.44	191.08	1.58%	0.76	149.84	4.40%	1.32
PPO (*s.*)	105.59	9.62%	1.11	207.53	10.33%	2.36	160.86	12.07%	6.42
DANIEL (*s.*)	101.67	5.55%	0.74	192.78	2.49%	1.87	153.22	6.75%	6.10
SLIM (*s.*)	98.74	2.51%	2.32	189.08	0.52%	6.91	149.02	3.82%	20.08
MACSIM (*s.*)	97.64	1.37%	0.86	**188.10**	0.00%	2.28	145.95	1.69%	6.19

$N \times M$	20×10			30×10			40×10		
Metric	Obj.	Gap	Time (s.)	Obj.	Gap	Time (s.)	Obj.	Gap	Time (s.)
OR-Tools	195.98	3.14%	1800	**274.67**	0.00%	1800	365.96	0.08%	1800
PPO (*g.*)	215.78	13.56%	1.91	312.59	13.81%	2.86	416.18	13.81%	3.82
DANIEL (*g.*)	198.50	4.46%	1.85	281.49	2.48%	2.76	371.45	1.58%	3.81
SLIM (*g.*)	195.89	3.09%	3.11	281.87	2.62%	4.57	374.13	2.31%	6.03
MACSIM (*g.*)	192.15	1.12%	1.19	276.01	0.49%	1.71	365.87	0.05%	2.27
PPO (*s.*)	214.81	13.05%	6.23	308.55	12.33%	12.79	410.76	12.33%	24.54
DANIEL (*s.*)	193.91	2.05%	6.35	279.20	1.65%	12.37	370.08	1.21%	21.09
SLIM (*s.*)	194.19	2.19%	28.15	281.42	2.46%	69.97	373.70	2.20%	139.30
MACSIM (*s.*)	**190.02**	0.00%	6.79	275.48	0.29%	14.12	**365.67**	0.00%	27.13

IV Experiments

❖ 실험 결과

FFSP

$N \times M_i \times S$	$20 \times 4 \times 3$			$50 \times 4 \times 3$			$100 \times 4 \times 3$		
Metric	Obj.	Gap	Time (s.)	Obj.	Gap	Time (s.)	Obj.	Gap	Time (s.)
Gurobi (600s)	31.61	31.93%	600	-	-	600	-	-	600
Shortest Job First	31.27	30.51%	0.13	56.94	19.27%	0.37	99.27	13.82%	0.62
Genetic Algorithm	31.15	30.01%	21.05	56.92	19.23%	44.82	99.25	13.79%	89.20
Particle Swarm	29.10	21.45%	46.17	55.10	15.42%	82.46	97.30	11.56%	154
MatNet (*g.*)	27.26	13.77%	1.22	51.52	7.92%	2.17	91.58	5.00%	4.97
PolyNet (*g.*)	26.71	11.48%	1.69	51.01	6.85%	2.45	91.22	4.59%	5.21
PARCO (*g.*)	26.31	9.81%	0.26	51.19	7.23%	0.52	91.29	4.67%	0.89
SLIM (*g.*)	26.18	9.27%	0.86	50.01	4.75%	3.36	91.97	5.45%	5.14
MACSIM (*g.*)	25.75	7.47%	0.28	49.36	3.39%	0.43	89.88	1.86%	0.96
MatNet (*s.*)	25.43	6.14%	3.88	49.68	4.06%	8.91	89.72	2.87%	18.00
PolyNet (*s.*)	24.98	4.26%	5.04	49.23	3.12%	9.24	89.21	2.28%	19.29
PARCO (*s.*)	24.78	3.42%	0.99	49.27	3.20%	1.97	89.46	2.57%	4.04
SLIM (*s.*)	24.19	0.96%	1.55	48.13	0.82%	10.21	89.50	2.61%	19.01
MACSIM (*s.*)	**23.96**	0.00%	0.49	**47.74**	0.00%	0.91	**87.22**	0.00%	3.60

HCVRP

$N \times M$	60×3			80×3			100×3		
Metric	Obj.	Gap	Time (s.)	Obj.	Gap	Time (s.)	Obj.	Gap	Time (s.)
SISRs	**6.57**	0.00%	478	**8.52**	0.00%	750	**10.29**	0.00%	1084
Genetic Algorithm	9.21	40.18%	411	12.32	44.60%	612	15.33	48.98%	845
Simulated Annealing	7.04	7.15%	382	9.17	7.63%	561	11.13	8.16%	765
ET (*g.*)	7.58	15.37%	0.28	9.76	14.55%	0.38	11.74	14.09%	0.45
DRL_{Li} (*g.*)	7.43	13.09%	0.34	9.64	13.15%	0.46	11.44	11.18%	0.58
2D-Ptr (*g.*)	7.20	9.59%	0.2	9.24	8.45%	0.27	11.12	8.07%	0.31
SLIM (*g.*)	7.19	9.44%	0.63	9.25	8.57%	0.87	11.10	7.87%	1.04
MACSIM (*g.*)	7.15	8.83%	0.35	9.15	7.39%	0.43	11.02	7.09%	0.78
ET (*s.*)	7.14	8.68%	0.52	9.19	7.86%	0.66	11.20	8.84%	1.02
DRL_{Li} (*s.*)	6.97	6.09%	0.73	9.10	6.81%	1.1	10.90	5.93%	1.48
2D-Ptr (*s.*)	6.82	3.81%	0.32	8.85	3.87%	0.44	10.71	4.08%	0.55
SLIM (*s.*)	6.88	4.75%	2.40	8.92	4.69%	3.31	10.81	5.05%	4.09
MACSIM (*s.*)	6.76	2.89%	1.65	8.78	3.05%	2.29	10.67	3.69%	2.76

IV Experiments

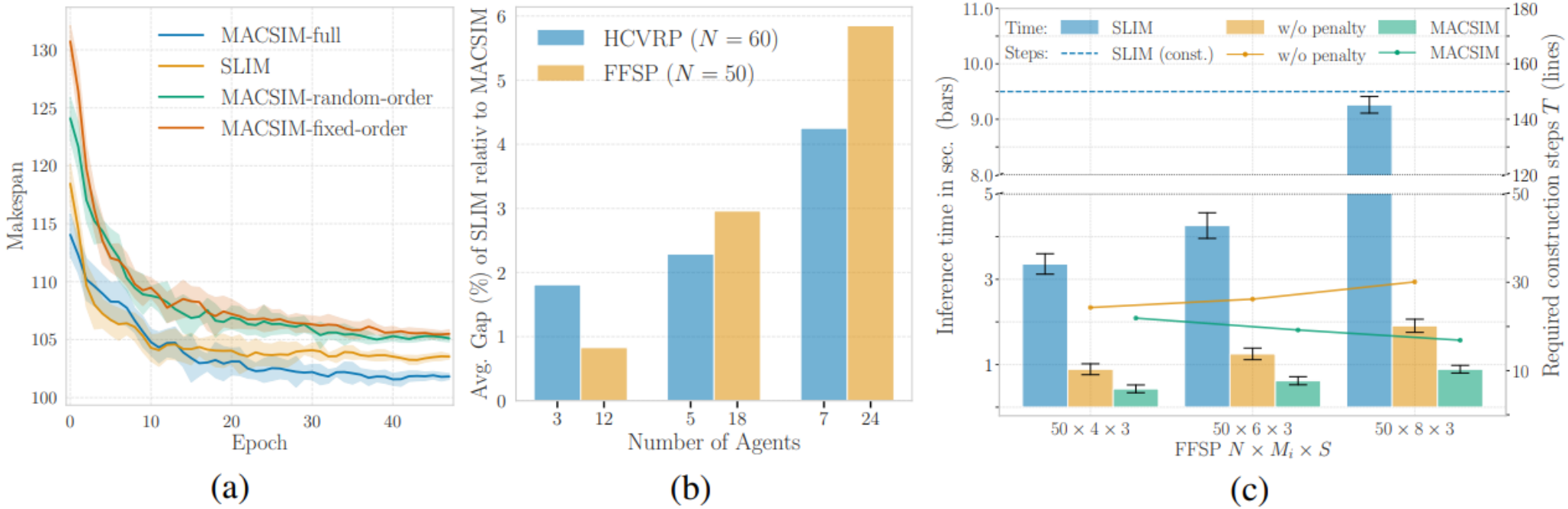
❖ 실험 결과

✓ (b)

- agent 수가 증가할수록 SLIM 대비 MACSIM의 성능 우위가 더욱 명확 → multi-agent의 대칭성을 효과적으로 활용하기 때문

✓ (c)

- 기존 SLIM 대비 추론 시간 단축
- Multi-agent 정책으로 인해 솔루션 구성 단계 감소 → 에이전트 수가 증가함에 따라 더욱 크게 감소



V Conclusion

❖ 결론

- MACSIM은 Multi-agent 조합 최적화 문제를 위한 새로운 self-improvement 프레임워크 제안
- 한 번에 모든 에이전트-작업 쌍의 집합을 샘플링하고 학습함으로써 Multi-agent 문제의 구조적 대칭성 활용
- Skip token 도입으로 유연한 솔루션 구성 가능
- 솔루션 품질 향상, 솔루션 생성 시간 감소, 뛰어난 일반화 능력 입증

❖ 한계점 및 향후 연구

- 단계별 인코딩에 의존하기 때문에 일부 경량 신경망 솔버보다 생성 속도가 느릴 수 있음
- 더 효율적인 정책 아키텍처를 개발하여 생성 지연 시간을 더욱 줄이는 데 중점을 둘 것

MACSIM for NCO (Part II): Solving HCVRP

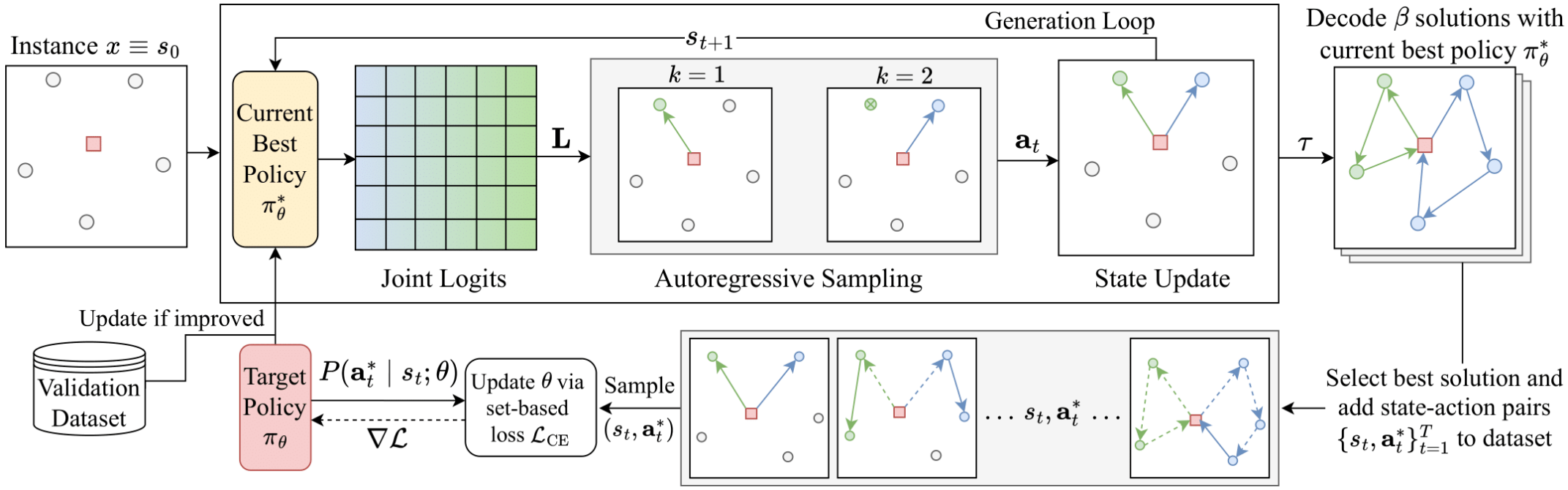
26. 01. 08

노정빈

I Background

❖ MACSIM 개요

- 매 스텝(t)마다 로짓 행렬을 1번 생성 (t: 동시에 결정하는 의사결정 시점)
- 그 안에서 모든 에이전트-작업 쌍을 충돌 없이 순차적으로 선택
- 상태 갱신 후 다음 스텝 반복



Ⅱ 문제 설정

❖ 최소-최대 이종 용량 제한 차량 경로 문제(Min-Max Heterogeneous Capacitated Vehicle Routing Problem, HCVRP)

✓ 노드(고객, 작업)

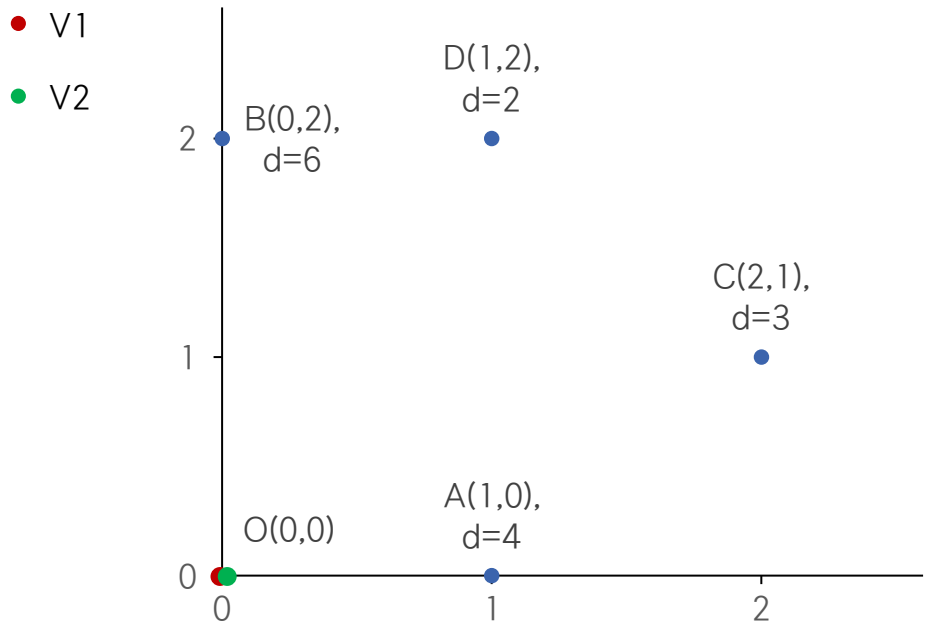
- Depot O: (0, 0)
- A: (1, 0), demand=4
- B: (0, 2), demand=6
- C: (2, 1), demand=3
- D: (1, 2), demand=2

✓ 차량(에이전트)

- V1: capacity=10, speed=1.0
- V2: capacity=6, speed=0.8

✓ 목표

- 모든 고객 방문 완료
- 각 차량의 총 이동시간 중 최대값 최소화

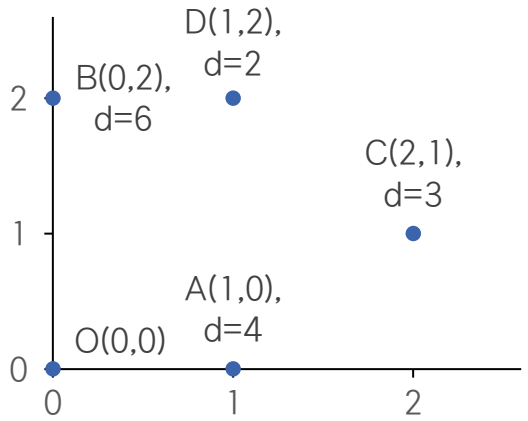


Ⅲ Multi-agent Policy

❖ Policy 입력 – 현재 상태

✓ 노드 정보

- 좌표: depot 포함 모든 노드의 위치
- 수요(demand): depot은 0으로 설정
- 방문 여부(Visited)



✓ 차량 정보

- 용량
- 속도
- 현재 상태: 현재 위치 좌표, 현재 누적 이동거리, 사용한 용량

✓ 제약(Mask)

- 선택 가능 여부(depot, skip token 포함) → 갈 수 있으면 True

<t=0에서 Mask>

	A	B	C	D	Depot	Skip
V1	T	T	T	T	T	T
V2	T	T	T	T	T	T

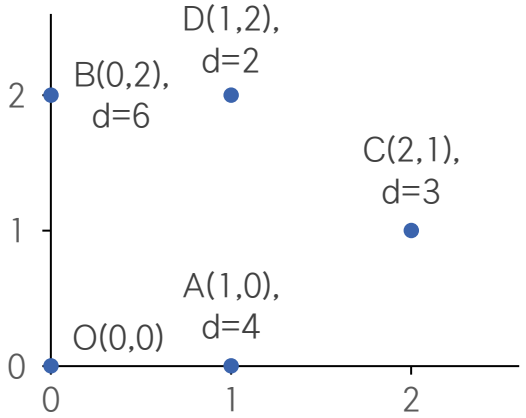
Ⅲ Multi-agent Policy

❖ Input Embedding: 노드/차량의 특징을 d차원의 벡터로 변환

✓ 노드 특징

- Depot: [x, y, 0]
- 고객 : [x, y, demand]

→ 초기 노드 임베딩



✓ 차량 특징

- 현재 위치 좌표
- Depot과의 거리
- 현재 누적 이동거리
- 속도
- 용량
- 남은 용량 = 용량-사용한 용량
- 진행률 = $\frac{\text{남은 노드}}{\text{전체 노드}}$

→ 초기 차량 임베딩

Ⅲ Multi-agent Policy

❖ Encoder

✓ MatNet(Matrix Networks) 인코더 VS AM(Attention Model) 인코더

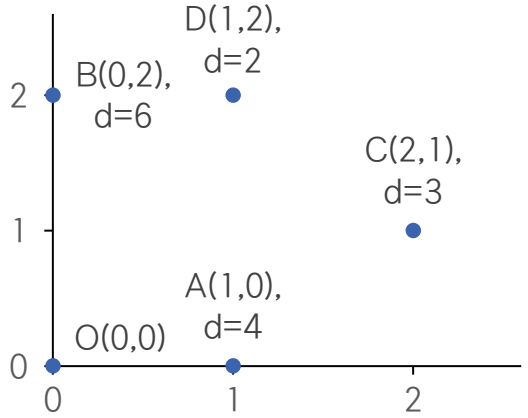
- MatNet: cross-attn + self-attn
- AM: self-attn, cross-attn은 디코더에서 수행

✓ 스케줄링 문제(FFSP/FJSP)

- 기계-작업 간의 비용이 '정적'이므로 인코더에서 한 번에 처리
- MatNet의 cross-attn에서 고정 비용 행렬을 활용해 기계↔작업 관계 학습

✓ 차량경로 문제(HCVRP)

- 차량-노드 간의 비용이 '동적'이므로 매 스텝(t)마다 달라짐 → 상대적으로 가벼운 AM 인코더 사용
- AM 인코더는 self-attn으로 노드/차량의 기본 임베딩만 생성
- 디코더에서 현재 상태(현재 위치, 사용한 용량 등)를 반영하여 다음 행동 결정



<t=0에서 가중치 행렬(E)>

	A	B	C	D
V1	1.000	2.000	2.236	2.236
V2	1.250	2.500	2.795	2.795

※ HCVRP에서 가중치 행렬은 현재 위치 기준 이동시간 (= $\frac{\text{거리}}{\text{속도}}$)

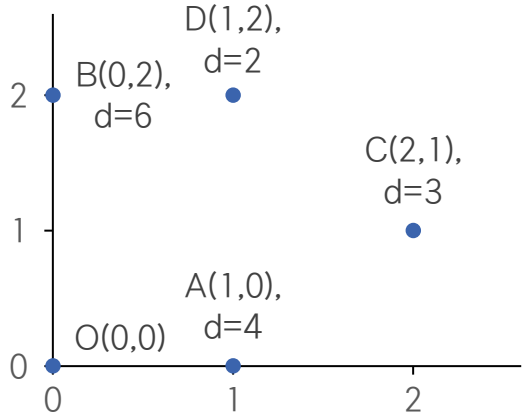
Ⅲ Multi-agent Policy

❖ Decoder (Multiple-pointer Mechanism)

- Query = 차량 임베딩
- Key = 노드 임베딩 + skip
- 인코더에서 생성된 최종 임베딩과 가중치 행렬을 사용하여 로짓 행렬(L) 계산
- $L_{m,v}$: 차량 m이 노드 v로 가는 점수

<t=0에서 가중치 행렬(E)>

	A	B	C	D
V1	1.000	2.000	2.236	2.236
V2	1.250	2.500	2.795	2.795



$$L = c \cdot \tanh\left(\frac{QK^T}{\sqrt{d}}\right), \qquad Q = W_{Dec}^{Q} H_M, \qquad K = W_{Dec}^{K}\left(H_v || h^{skip}\right)$$

<t=0에서 로짓 행렬 예시>

	A	B	C	D	Depot	Skip
V1	1.6	0.7	0.5	0.4	-0.2	-0.6
V2	0.9	1.2	0.3	0.2	-0.2	-0.6

IV Multi-action Generation

❖ Greedy

✓ 로짓 행렬에서 점수가 높은 순으로 차량-노드 쌍 선택

❖ Sampling

✓ 로짓 행렬에 Mask 적용 → softmax → 확률 분포(뽑힐 확률)

✓ 모든 차량-노드 쌍 선택

① 차량-노드(고객+skip) 쌍 1개 뽑기

② 로짓 행렬에서 선택된 차량(행), 노드(열) 제거(마스킹)

③ 차량 수만큼 반복

→ 샘플링 할 때는 인코더 및 디코더 반복X

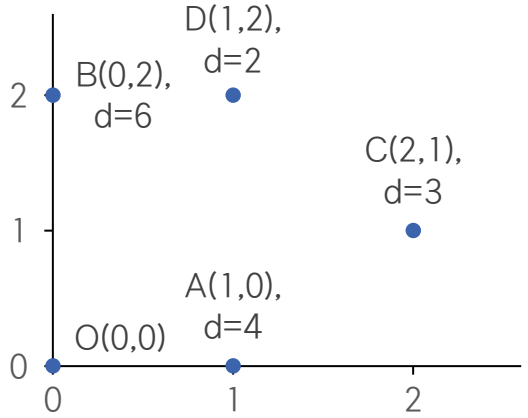
✓ 샘플 수

- 샘플링은 랜덤성 존재 → 여러 번(K) 뽑아 그중 가장 좋은 해(min-max) 선택
- K = 시도 횟수 x 변형 횟수 x 전략 개수 = 128 x 8 x 1 = 1024

→ 샘플링 여러 번 + best 선택으로 최적해에 근사할 수 있음

<t=0에서 로짓 행렬>

	A	B	C	D	Depot	Skip
V1	1.6	0.7	0.5	0.4	-0.2	-0.6
V2	0.9	1.2	0.3	0.2	-0.2	-0.6



<t=0에서 샘플링 과정>

① 확률 분포에 의해 (V2-B) 뽑힘

② V2 행, B 열 마스킹

③ (V1-A) 뽑힘

④ 1번 샘플링 {(V1-A), (V2-B)} 생성

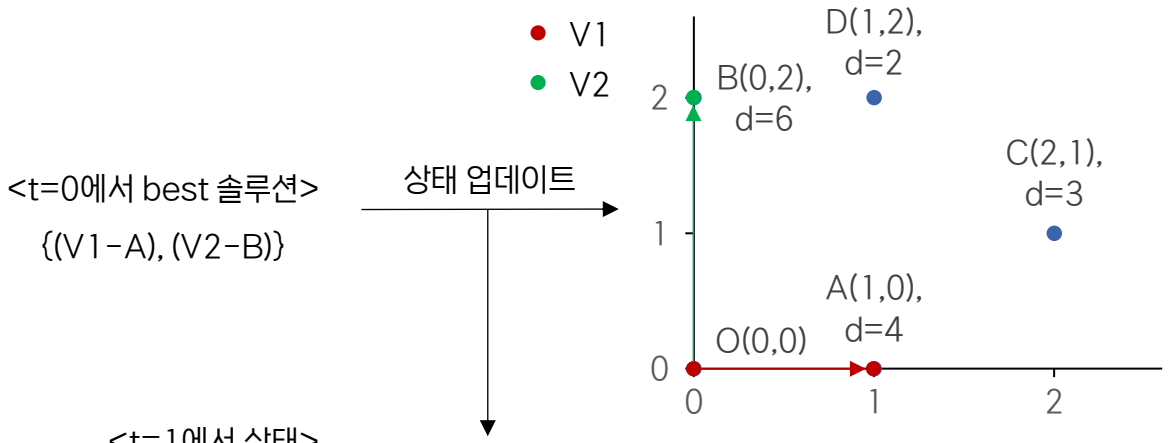
⑤ 여러 번 샘플링 후 best 선택

(1번 샘플링이 best라고 가정)

V Update

❖ 상태 업데이트

- ✓ 노드
  - 방문 여부
- ✓ 차량
  - 현재 위치 좌표
  - 현재 누적 이동거리
  - 사용한 용량
- ✓ Mask
  - 선택된 노드 및 차량 False
  - (차량의 남은 용량 < 노드의 수요) 이면 False



<t=1에서 상태>

- 방문 여부: A, B 방문 완료
- V1: 현재 위치=A, 누적 이동거리=1, 사용한 용량=4 → 남은 용량=10-4=6
- V2: 현재 위치=B, 누적 이동거리=2, 사용한 용량=6 → 남은 용량=6-6=0
- Mask: A, B는 방문 완료 / V2의 남은 용량=0 이므로 False

	A	B	C	D	Depot	Skip
V1	F	F	T	T	T	T
V2	F	F	F	F	T	T

❖ 반복

- ✓ 다음 스텝(t+1): 새로운 상태로 (임베딩 → 인코더 → 디코더)를 재실행하여 샘플링
- ✓ 모든 노드를 방문할 때까지 반복

VI Result

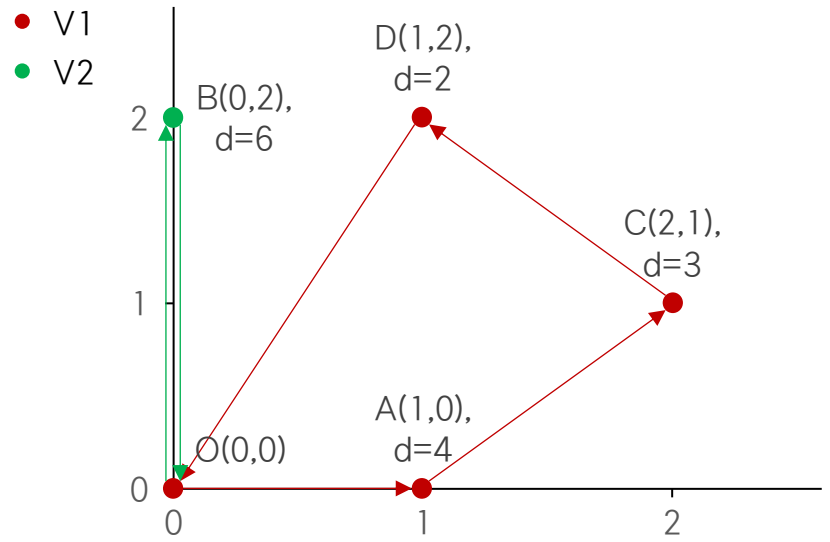
❖ 결과

✓ 차량 경로 문제

- 최종해 = 차량별 경로 (depot → … → depot)
- 차량별 시간 = $\frac{\text{총 이동거리}}{\text{속도}}$
- 목적: 차량별 시간 중 가장 큰 것을 최소화

✓ 스케줄링 문제

- 최종해 = 기계별 작업 공정
- 목적: makespan 최소화



<최종해>

- V1: O→A(4)→C(3)→D(2)→O, 총 이동거리 = 6.0645

  → 이동시간 = $\frac{6.0645}{1.0}$ = 6.0645

- V2: O→B(6)→O, 총 이동거리 = 4.0000

  → 이동시간 = $\frac{4.0000}{0.8}$ = 5.0000