

2025 상반기 ICT Internship

---

# ICT 인턴십 프로젝트 리뷰

2025.03.03. ~ 2025.06.30. (4개월)

---

김차미	2025.06.24
-----	------------

---



# 인턴십 기간 및 목표

2025 상반기 ICT 인턴십 수행

“ 2025.03.03. ~ 2025.06.30. (4개월) ”



## 실무 경험을 쌓아보자!

실무에서 실제 데이터를 만지고  
하나의 프로젝트를 다양한 사람들과  
협업하며 실무 역량을 배워보고 싶다!



## 실제 문제를 해결해보자!

실제 문제를 발견하고,  
이를 최적화하여 문제를 해결하는 경험을  
통해 성취감을 얻고 싶다!



## 앞으로 어떤 공부를 해야 할까?

학점만 잘 받는 공부는  
실무에 필요한 역량을 키울 수 없다.  
앞으로 어떻게 공부를 해야 할까?

## Content

# 목차

## | Part 1

### Domain-Knowledge 획득 및 Research 수행

- Genetic Algorithm 및 TSP(Traveling Salesman Problem) 탐색
- H-TSP 리뷰와 데이터셋 변경 실험
- DiffEvo를 활용한 TSP task 해결
- OpenES, TabPFN 논문 리뷰

## | Part 2

### 1.5D CSP Project

- 1D CSP(Cutting Stock Problem) 이해
- 1.5D CSP 문제 재정의 및 2-Stage 방법론 탐색



## Part 1

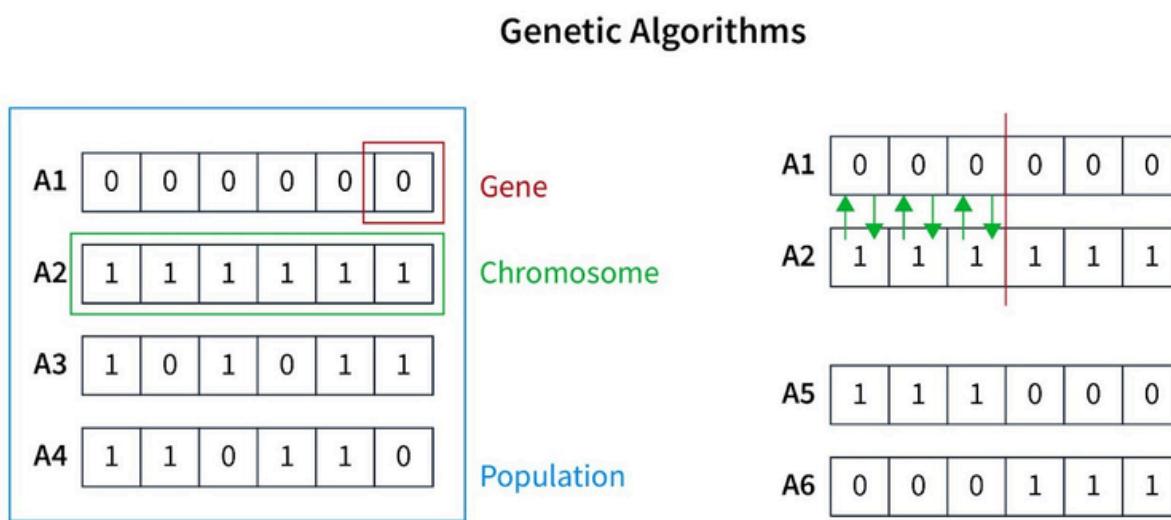
# Domain-Knowledge 획득 및 Research 수행

- Genetic Algorithm 및 TSP(Traveling Salesman Problem) 탐색
- H-TSP 리뷰와 데이터셋 변경 실험
- DiffEvo를 활용한 TSP task 해결
- OpenES, TabPFN 논문 리뷰

# Genetic Algorithm 및 TSP(Traveling Salesman Problem) 탐색

## Genetic Algorithm에 대한 이해

- A search heuristic inspired by the process of natural selection



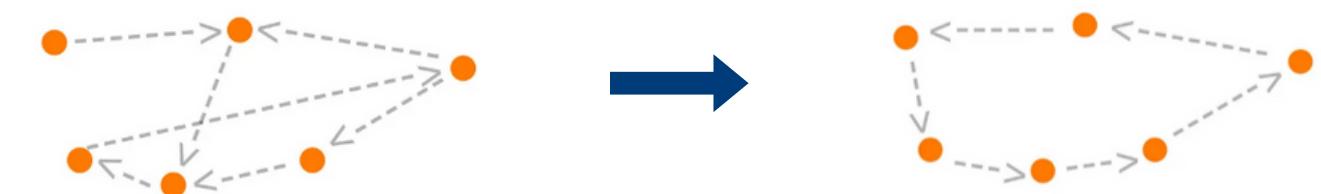
- GA 관련 리뷰 논문인 'A review on genetic algorithm: past, present, and future' 논문을 읽고 GA에 대해 이해

- ML이나 DL 같이 학습 데이터의 제약을 크게 받지 않음
- 복잡하고 넓은 탐색 공간을 GA 연산(Crossover, Mutation, ...)을 통해 효과적으로 탐색
- 문제 적용의 유연성을 가지고 있음

## TSP 탐색 및 이해

- TSP: Travelling Salesman Problem

→ 도시와 도시 간의 거리가 주어졌을 때 모든 도시를 정확히 한 번씩 방문하는 최단 거리를 찾는 문제 (다시 시작점으로 돌아와야 함)



- 택배 배송 경로 최적화: 하루에 수십~수백 개의 주소에 빠르게 방문하려면 최적의 경로가 중요
- 차량 경로 계획: 예를 들어 셔틀 버스의 효율적인 시간표 및 경로 구성
- 제조 및 생산 공정 최적화

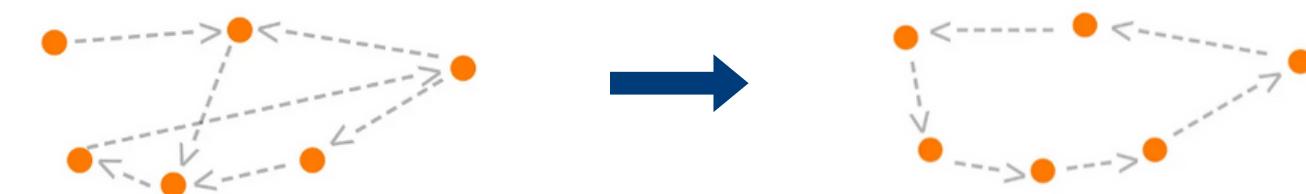
- 변형 문제

- Generalized TSP (GTSP): 노드가 그룹으로 묶여있는 TSP
- Prize Collecting TSP: 반드시 모든 도시를 방문할 필요 없이 보상을 모으면서 최적의 경로를 찾는 문제
- Multiple TSP: m명의 외판원이 하나의 노드에서 시작해서 다시 돌아오는 TSP 변형 문제

# Genetic Algorithm 및 TSP(Traveling Salesman Problem) 탐색



## TSP 탐색 및 이해

- TSP: Travelling Salesman Problem
  - 도시와 도시 간의 거리가 주어졌을 때 모든 도시를 정확히 한 번씩 방문하는 최단 거리를 찾는 문제 (다시 시작점으로 돌아와야 함)
- 
- 택배 배송 경로 최적화: 하루에 수십~수백 개의 주소에 빠르게 방문하려면 최적의 경로가 중요
- 차량 경로 계획: 예를 들어 셔틀 버스의 효율적인 시간표 및 경로 구성
- 제조 및 생산 공정 최적화
- 변형 문제
  - Generalized TSP (GTSP): 노드가 그룹으로 묶여있는 TSP
  - Prize Collecting TSP: 반드시 모든 도시를 방문할 필요 없이 보상을 모으면서 최적의 경로를 찾는 문제
  - Multiple TSP:  $m$ 명의 외판원이 하나의 노드에서 시작해서 다시 돌아오는 TSP 변형 문제

## H-TSP: Hierarchically Solving the Large-Scale Travelling Salesman Problem

### 기존 TSP 해결 방법론들의 문제점과 한계

- 기존 해를 반복적으로 개선하는 방법
  - 초기 해를 랜덤 혹은 휴리스틱 방법을 이용해 구한 후 학습된 모델이 기존 해를 지속적으로 개선하는 방식
  - 스케일이 큰 TSP 학습과 추론에 시간이 많이 소요됨
  - 초기 해 품질이 전체 성능에 큰 영향을 미침
- 처음부터 하나씩 경로를 만들어나가는 구성적 방법
  - 출발 도시에서 시작해 매 단계마다 다음에 방문할 도시를 결정하는 방식
  - 스케일이 작은 TSP에서는 좋은 성능을 보이지만 스케일이 큰 TSP로 확장하기 어려움
  - 탐색 공간이 너무 커서 학습 시 메모리, 계산량 증가

---

TSP 문제를 여러 개의 작은 open-loop TSP 하위 문제로 나눈 후, 이를 순차적으로 해결하여 전체 해를 구성하는 Divide-and-Conquer 방식의 계층적 접근법

#### [1단계 - 상위 레벨 모델]

PPO를 활용해 방문해야 할 모든 도시 중 작은 부분 집합을 선택해 하위 레벨 모델에게 전달

#### [2단계 - 하위 레벨 모델]

Transformer 기반 모델로 강화학습 훈련을 통해 선택된 도시만 포함하는 작은 열린 경로 TSP를 해결하고 상위 레벨 모델로 반환

- 상태: 현재 Context
- 행동: 다음에 방문할 노드 선택
- 보상: 전체 경로 길이의 음의 값

→ 기존 부분 경로에 병합되고 이러한 단계는 모든 도시를 방문하고 실행 가능한 해를 얻을 때까지 반복됨

---

#### Algorithm 1: Hierarchical TSP Algorithm

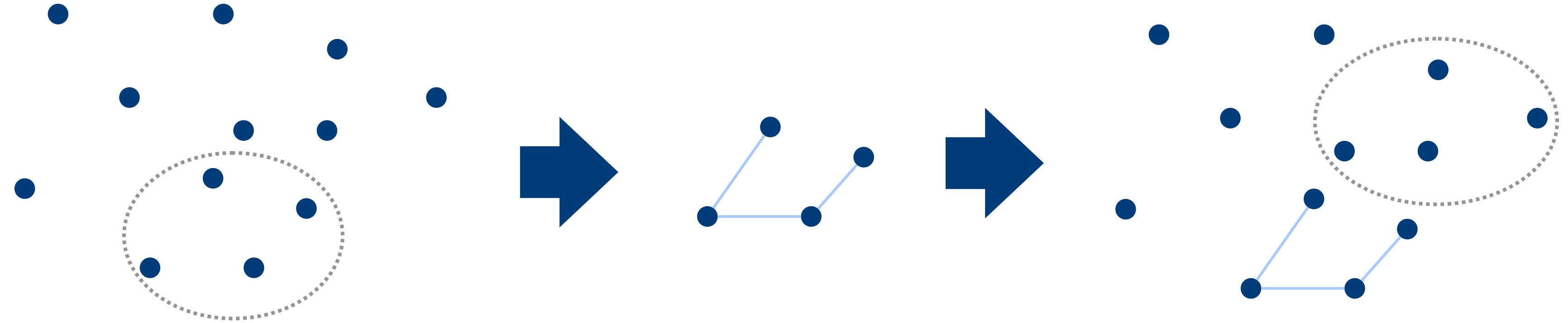
---

```
Input: TSP instance  $V = \{v_1, v_2, \dots, v_N\}$ , initial solution  $\tau_{init} = \{v_d\}$ 
Output: Solution route  $\tau = \{\tau_1, \tau_2, \dots, \tau_N\}$ 
1  $\tau \leftarrow \tau_{init}$ , the nearest node  $v$  of  $v_d$  ;
2 while  $len(\tau) < N$  do
3    $SubProb \leftarrow GenerateSubProb(V, \tau)$  ;
4    $SubSol \leftarrow SolveSubProb(SubProb)$  ;
5    $\tau \leftarrow MergeSubSol(SubSol, \tau)$  ;
6 end
7 return  $\tau$ 
```

---

## H-TSP: Hierarchically Solving the Large-Scale Travelling Salesman Problem

TSP 문제를 여러 개의 작은 open-loop TSP 하위 문제로 나눈 후, 이를 순차적으로 해결하여 전체 해를 구성하는 Divide-and-Conquer 방식의 계층적 접근법

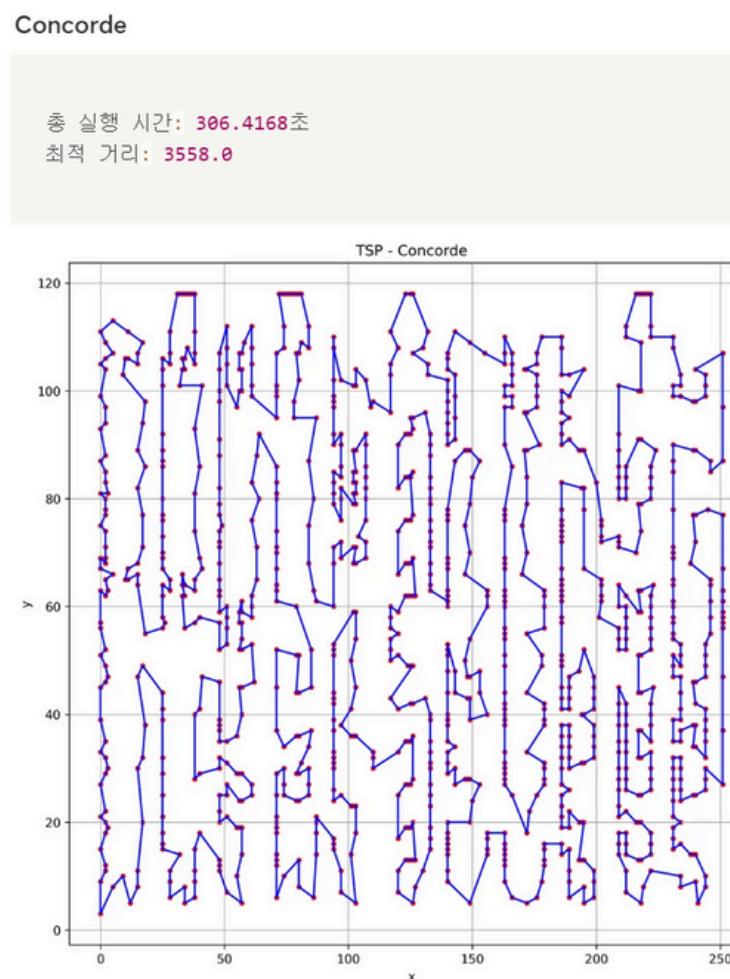
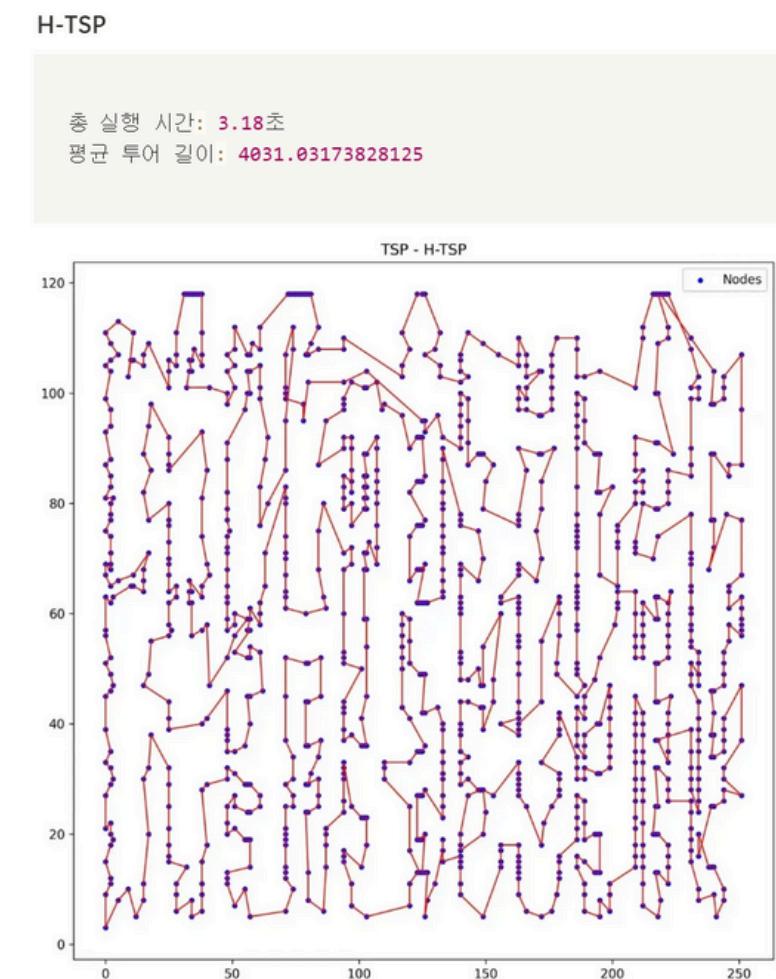


# H-TSP 리뷰와 데이터셋 변경 실험

## H-TSP 레포 확인 및 데이터셋 적용

### Concorde 와 비교

- ?Concorde: 여러 알고리즘을 통합한 TSP 최적화 패키지
- 1083 points를 가진 TSPLIB 데이터셋으로 변경



→ 성능을 어떻게 더 올릴 수 있을까?

```
line 491, in get_fragment_knn  
assert predict_coord.min() >= 0  
AssertionError
```

하이퍼파라미터 실험 중 코드 오류 발견

improvement\_step 값 설정 시 AssertionError 발생



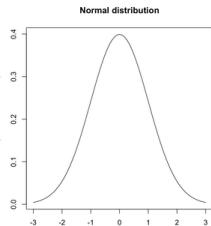
### 분석 결과 도출된 원인

- ① 개선 단계에서 랜덤으로 예측한 행동을 도출할 때 [-1, 1] 범위를 넘음
- ② start\_city 변수 할당 X

## H-TSP 디버깅

### AssertionError 해결

- improvement\_step: 경로 최적화 후 추가적으로 Random 좌표를 찍어 부분 경로를 생성해 추가적인 개선을 가능하게 하는 하이퍼 파라미터
- 논문에서는 전체 노드들의 좌표 값을  $[0,1]$ 로 가정
  - BUT 상위 레벨 모델이 도출하는 노드 부분 집합은 tanh를 사용하기 때문에  $[-1, 1]$  범위
  - So, 하위 레벨로 넘어갈 때  $[0,1]$  범위로 바꿔주어야 함
- BUT improvement\_step을 설정했을 때 실행되는 개선 단계에서 부분 집합에 들어갈 랜덤한 좌표를 뽑을 때 torch.randn 사용  
 $\rightarrow [0,1]$  이외의 값이 생성됨



### 해결 방안

⇒ torch.randn 를 torch.rand로 변환

→ torch.randn: 평균이 0이고 분산이 1인 정규분포에서 난수로 채워진 텐서 반환하는 함수

→ torch.rand: 균일 분포에서 임의의 숫자로 채워진 텐서를 반환 (범위 =  $[0, 1]$ )

### UnboundLocalError: local variable 'start\_city' referenced before assignment 발생

- start\_city: 상위 레벨 모델이 부분 집합을 생성하고 하위 레벨 모델에서 부분 경로를 생성할 때 이미 방문된 노드  
→ 하위 레벨 모델이 부분 경로를 생성할 때 어디에 새로운 경로가 이어질지에 대한 Context가 필요함

### 해결 방안

⇒ 랜덤으로 정해진 노드의 부분 집합에서 가장 가까운 방문된 노드를 할당

```
else: ## 전체 노드가 이미 전체 경로에 다 추가된 상태
    # refine a old fragment
    assert not self._improvement_done ## 개선 단계가 완료되었으면 assert문 실행
    ## nearest_old_city = 상위모델이 예측한 좌표 주변 가장 가까운 방문된 노드
    ## -> 원래는 방문되지 않은 노드를 선택하지만 개선 단계에서는 방문되지 않은 노드가 없기 때문에
    nearest_old_city = self.state.get_nearest_old_city_idx(predict_coord)
    new_cities = []
    ## 수정 - 추가
    ## start_city라는 기준점 또한 방문된 노드의 가장 가까운 방문된 노드로 설정
    start_city = self.state.get_nearest_old_city_idx(
        self.x[nearest_old_city]
    )
```

# H-TSP 리뷰와 데이터셋 변경 실험

## H-TSP 디버깅 후 성능 향상 실험

### 에러 해결 후 성능 향상 실험

- 실험을 진행했을 때도 성능이 높아지지 않음

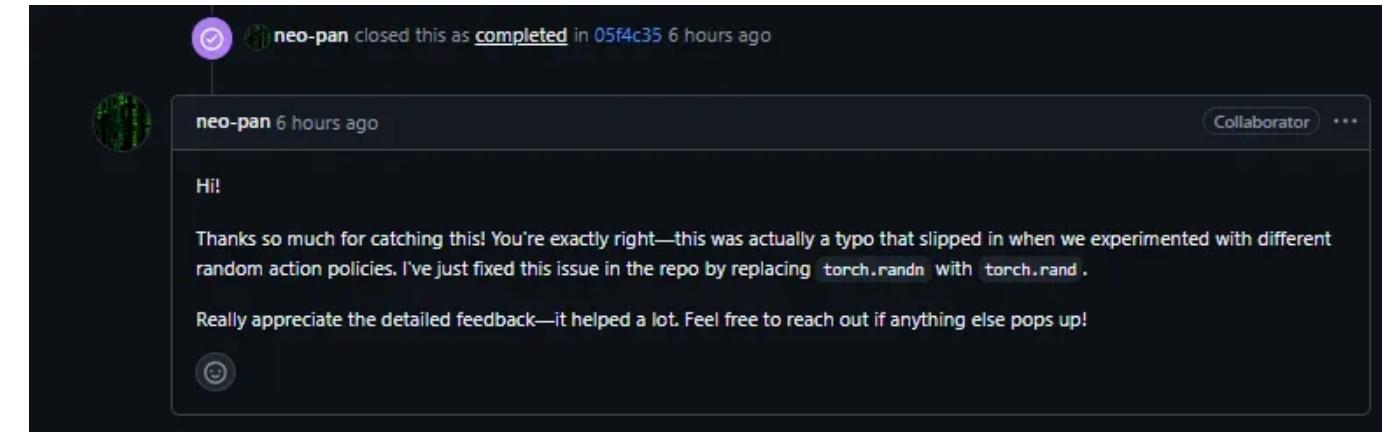
노드 개수	1000	2000	5000	10000				
improvement_step	time	최적 투어 길이	time	최적 투어 길이	time	최적 투어 길이	time	최적 투어 길이
default = 0	36.0374259948 7305	24.6607758104 80118	12.6352026462 55493	34.8758168220 52	25.2438640594 48242	55.0073347091 6748	44.3704833984 375	77.7509098052 9785
1	39.5228528976 4404	24.7577117234 46846	13.6389198303 22266	34.9813668727 87476	25.6070592403 41187	55.0791707038 8794	46.5537979602 8137	77.7844409942 627
5	56.8095860481 2622	24.9959048628 80707	15.5441026687 62207	35.2150440216 06445	33.6117324829 10156	55.3237504959 10645	47.2450163364 4104	78.0057406425 4761
...								
10	78.6941232681 2744	25.1278322935 10437	20.4447467327 11792	35.5545351505 27954	51.2170214653 01514	55.6623492240 90576	50.9711918830 8716	78.2378683090 21
100	473.217006683 3496	25.4086171239 6145	x	x	x	x	x	x
1000	4707.97184467 3157	25.8501151055 09758	x	x	x	x	x	x

원인 분석: 그렇다면 왜 성능이 높아지지 않는 걸까?

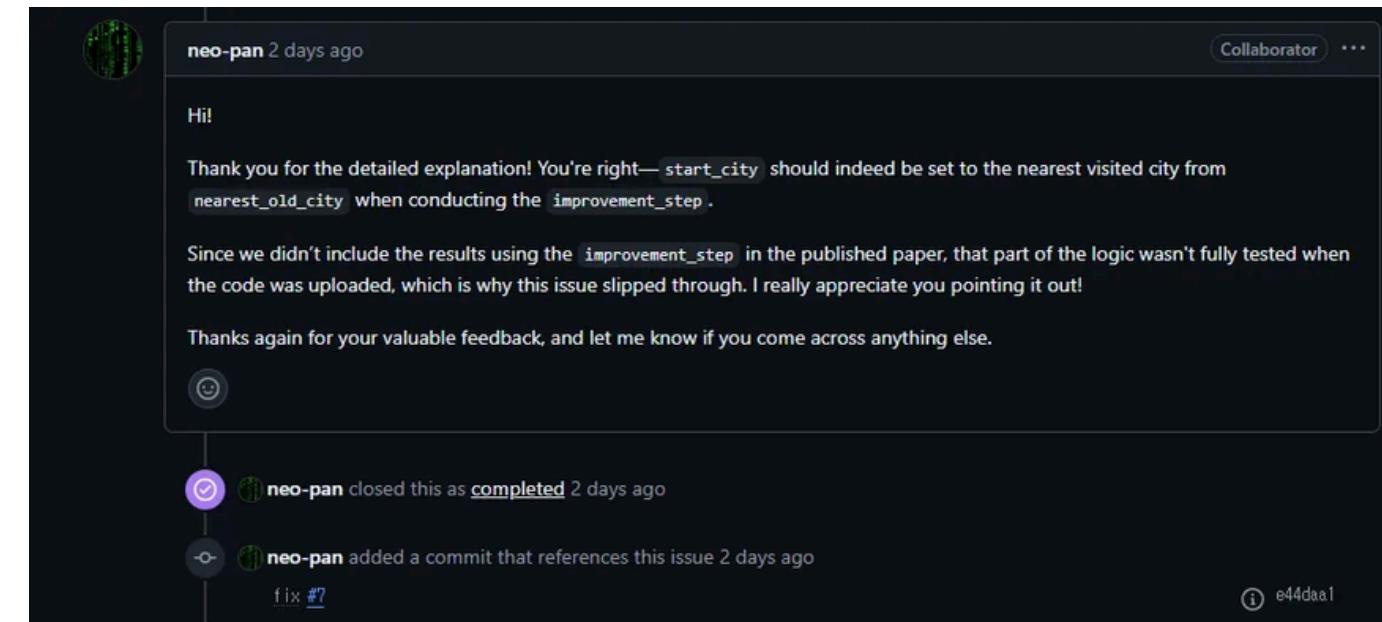
- random\_action()
- 개선 코드의 부재

### GitHub에 Issue 생성

1. Error when setting improvement\_step



2. UnboundLocalError: start\_city variable referenced before assignment in h\_tsp.py



# DiffEvo를 활용한 TSP task 해결

## Diffusion Models are Evolutionary Algorithms

Diffusion 관점에서 Evolutionary Algorithm을 수학적으로 재구성한 논문

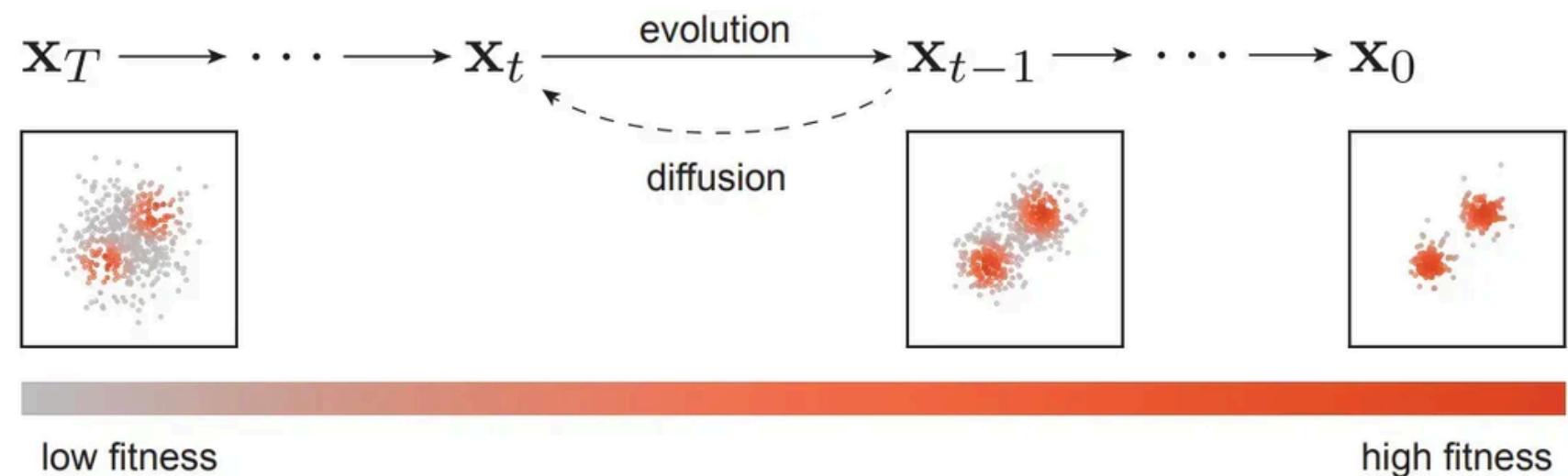
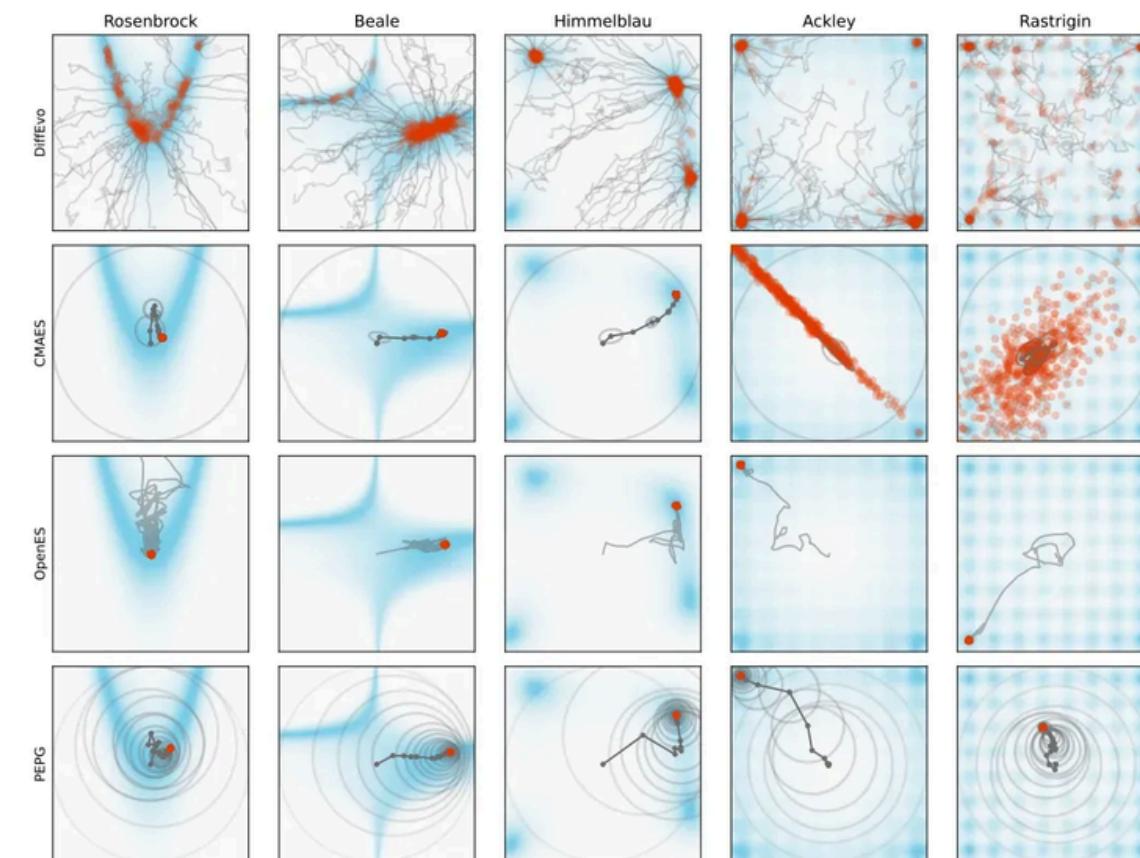


Figure 1: Evolution processes can be viewed as the inverse process of diffusion, where higher fitness populations (red points) have higher final probability density. The initially unstructured parameters are iteratively refined towards high-fitness regions in parameter space.

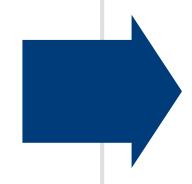


[Diffusion 과정]

노이즈가 가득한 데이터에서 점차적으로 노이즈를 제거하면서 타겟 분포를 추론

$$\mathbf{x}_t = \sqrt{\alpha_t} \mathbf{x}_0 + \sqrt{1 - \alpha_t} \boldsymbol{\epsilon},$$

$$\mathbf{x}_{t-1} = \sqrt{\alpha_{t-1}} \hat{\mathbf{x}}_0 + \sqrt{1 - \alpha_{t-1} - \sigma_t^2} \hat{\boldsymbol{\epsilon}} + \sigma_t \mathbf{w}.$$



[생물이 진화하는 과정]

다양한 생물들이 환경에 적응해 점차 적합한, 생존에 유리한 형질들만 생존

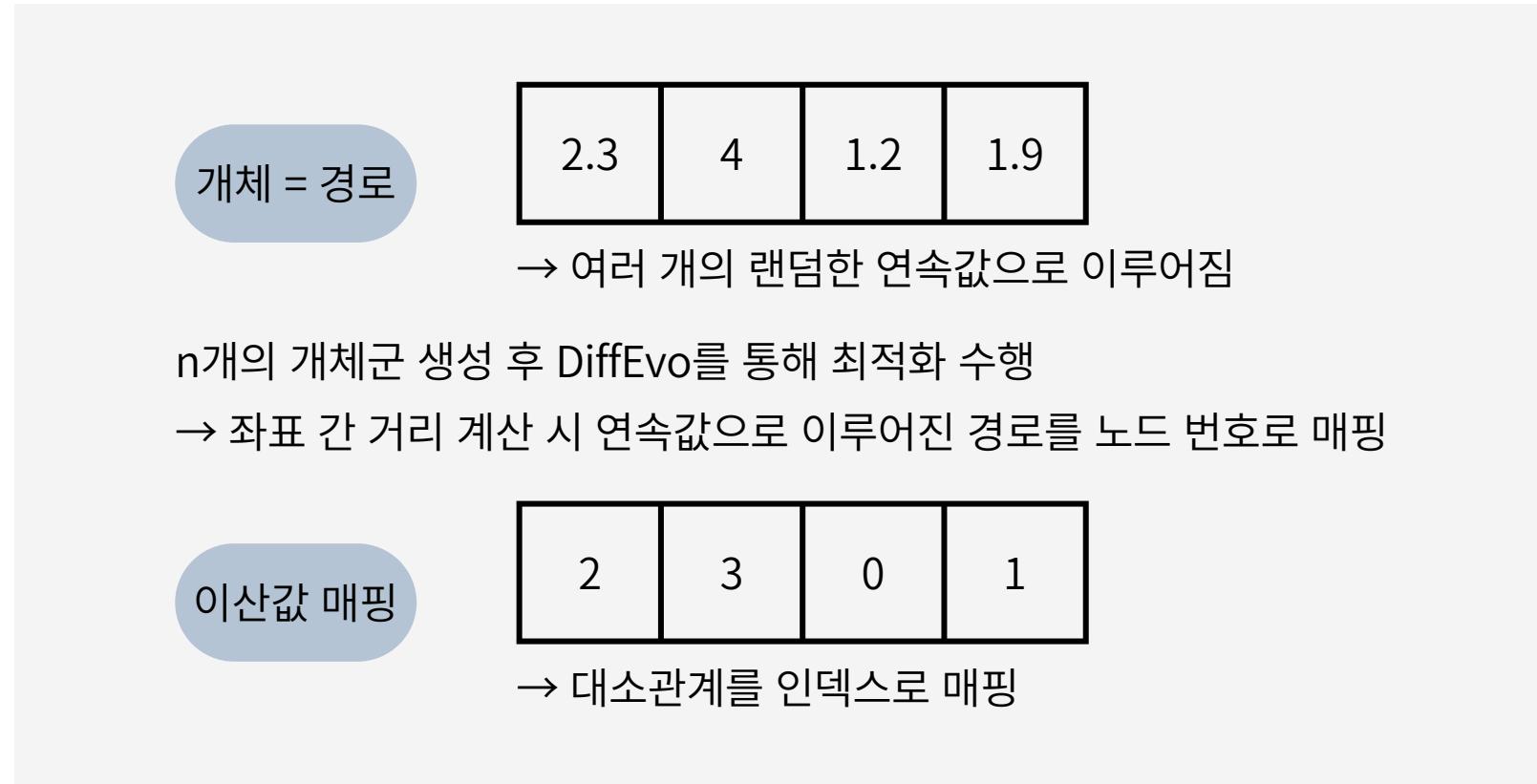
# DiffEvo를 활용한 TSP task 해결

## 1. Fitness 함수 정의

- TSP: 경로가 짧을 수록 좋음 (최소화해야 함)
  - DiffEvo: 적합도 함수가 클수록 좋음 (최대화해야 함)
- 경로 길이에 역수 OR 음수를 취한 것을 fitness로 활용

## 2. 개체 구성

- TSP: 번호가 적힌 노드들의 집합으로 이루어짐
- DiffEvo: 하나의 개체가 연속적인 숫자들로 이루어져 있고, 연속적으로 변화함



**<Diffusion Evolution>**

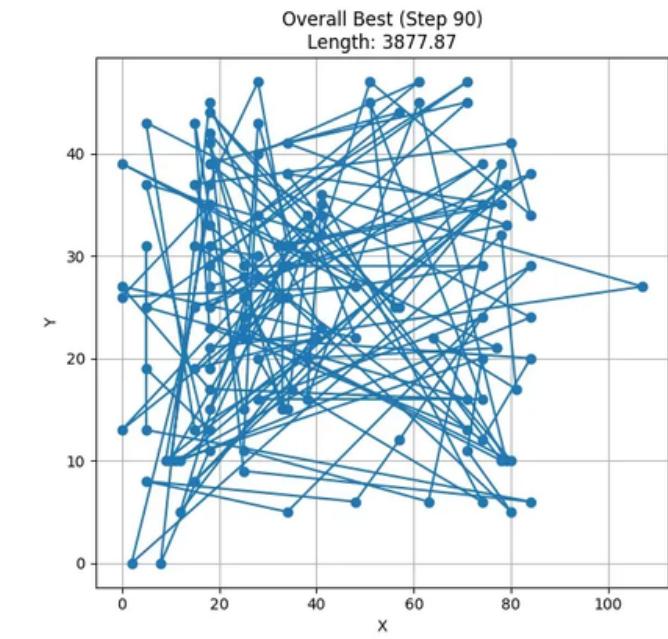
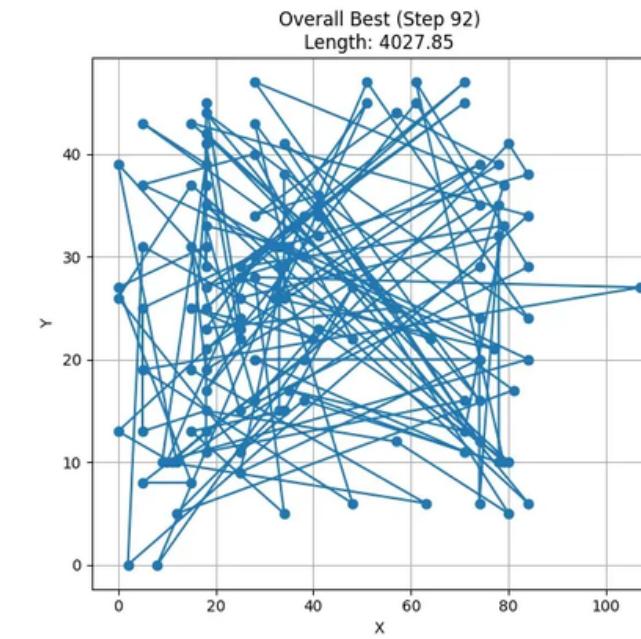
1. 초기화  
 $[x_t^1, x_t^2, x_t^3, \dots, x_t^N] \leftarrow \mathcal{N}(0, I^{N \times D})$   $\rightarrow N: \text{번화지 개수}, D: \text{좌표 차원 개수} (= \text{노드 개수})$   
 time step = T 일때 1번행 :  $[n_1, n_2, n_3, \dots, n_T]$
2. for  $t \in [T, T-1, \dots, 2]$  do
3.     for  $\forall i \in [1, 2, \dots, N]$  do  
 $\hat{x}_t^i = \mathcal{N}(x_t^i)$   $\rightarrow Q_i = f(\hat{x}_t^i) = x_t^i \text{의 적합도}$
4.      $Z = \sum_{j=1}^N Q_j \mathcal{N}(x_t^j)$   $\rightarrow \hat{x}_t^j = \frac{Q_j}{Z} x_t^j$
5.      $\hat{x}_t^i = \frac{1}{Z} \sum_{j=1}^N Q_j \mathcal{N}(x_t^j)$   $\rightarrow \hat{x}_t^i = \frac{Q_i}{Z} x_t^i + \frac{\sum_{j \neq i} Q_j}{Z} x_t^j$
6.      $\hat{x}_t^i = \frac{1}{Z} \sum_{j=1}^N Q_j \mathcal{N}(x_t^j)$   $\rightarrow \hat{x}_t^i = \frac{Q_i}{Z} x_t^i + \frac{\sum_{j \neq i} Q_j}{Z} x_t^j$
7.      $W = \mathcal{N}(0, I^D)$   $\rightarrow$  멤버 (mutation의 확률)
8.      $x_{t+1}^i = \hat{x}_t^i + \sqrt{1 - \alpha_t} \cdot \hat{x}_t^i + \sqrt{1 - \alpha_t} \cdot \frac{(x_t^i - \bar{x}_t^i)}{\sqrt{1 - \alpha_t}} + \alpha_t W$

(1)  $D = 4$ ,  $N = 2$  일 때

- i) 개체가 노드 번호로 이루어져 있을 경우 (최초화 =  $x_0 = [0, 1, 2, 3]$ )  
 $\hat{x}_0^1 = [0, 2, 3, 1] \rightarrow$  적합도: 4 / 적합도: 0.25  
 $\hat{x}_0^2 = [3, 1, 0, 2] \rightarrow$  적합도: 2 / 적합도: 0.5
- (2)  $\hat{x}_0 = \frac{1}{2} \sum_{j=1}^2 Q_j \mathcal{N}(\hat{x}_0^1; \sqrt{d} \hat{x}_0^1, 1 - \alpha_t) \hat{x}_0^2$   
 $= \frac{1}{2} ([\Delta, \Delta, \Delta, \Delta] + [\square, \square, \square, \square])$   
 $= [0.5, 0.5, 0.5, 0.5] \rightarrow$  적합도: 0.25
- $\hat{x}_{t+1}^1 = \sqrt{\alpha_t} \hat{x}_0 + \sqrt{1 - \alpha_t} \frac{(\hat{x}_0^1 - \bar{x}_0^1)}{\sqrt{1 - \alpha_t}} + \alpha_t W$   
 $= [2.6, 1, 1.2, 2.3]$
- ② 비슷한 노드 번호로 대체되는 경로는?  $\rightarrow$  멤버 기준  
 규칙이 있음!!!
- ③ 비슷한 노드 번호로 대체되는 경로는?  $\rightarrow$  멤버 기준  
 규칙이 있음!!!
- 2) 비슷한 노드 번호로 대체되는 경로는?  $\rightarrow$  멤버 기준  
 규칙이 있음!!!
- 3) 비슷한 노드 번호로 대체되는 경로는?  $\rightarrow$  멤버 기준  
 규칙이 있음!!!

$\rightarrow$  (2)  $[2.6, 1, 1.2, 2.3] \rightarrow [0, 1, 0, 2]$

## 3. 결과



- 전혀 최적화되지 않음...



# DiffEvo를 활용한 TSP task 해결

## 최적화되지 않는 문제 해결 시도

### ISSUE 1: $x_0$ 의 추정값을 구할 때 발생하는 문제

#### ISSUES

##### 1. $x_0$ 의 추정값을 구할 때의 문제

- 문제 상황: fitness가 높은 쪽으로 가지 않고 경로끼리 섞이는 느낌이 아님
- $x_0$ 의 추정값을 구할 때 가우시안 확률을 활용
- 이때 가우시안 확률의 평균은  $\mu = \text{self.x} * (\text{self.alpha} ** 0.5)$ 로 사용 (원본 데이터가  $x$ 일 때  $x$ 나 다른 경로가 관찰될 확률을 구하기 위함)
  - a. 연속값의 경우:  $\sqrt{\alpha_t}x \neq x \rightarrow$  단조 증가이기 때문에 순서가 바뀌지 않음
  - b. 연속값을 argsort한 경우:  $\sqrt{\alpha_t}x = x \rightarrow$  단조 증가이기 때문에 순서가 바뀌지 않음
- 가우시안 확률의 평균  $\mu$ 와  $x_t$ 가 같아지게 되고  $p_{\text{diffusion}}$  (= 각 원본 샘플일 때  $x_t$ 가 관찰될 확률)이 자기 자신일 때만 1이 되고 나머지는 다 0이 됨
  - 왜 0이 나올까? 다른 것들이 원본 샘플일 때  $x_t$ 가 관찰될 확률이 아예 없는 게 아닌데!
  - $\rightarrow \text{dist} = (x_t - \mu)$ 가 너무 커서 0이 나올 수 밖에 없음
- 해결 방안: max 값으로 나눠주어 정규화
- 예시



$$\hat{x}_0 = \frac{1}{2} \sum_{j=1}^n Q_t N(x_t^i; \sqrt{x_t^i} x_t^i, 1-x_t^i) x_t^i$$

설명:  $\hat{x}_0$ 는  $x_t^i$ 에 대한 확률 분포의 평균입니다.  $Q_t$ 는 확률 분포의 평균입니다.  $\sqrt{x_t^i} x_t^i$ 는 확률 분포의 표준 편차입니다.

- $x_0$ 의 추정값을 구하는 과정에서  $x^1$ 일 때  $x^1$ 만고 다른 경로들이 영향을 못 미치게 됨  
 $\rightarrow$  결국 자기 경로에 대해서만 영향을 받게 됨

## Chromosome을 어떻게 표현해도

## 연속값을 노드 번호로 매팅하는 것에서부터 문제가 발생한다!

### ISSUE 2: 최적화 성능이 좋지 않고, Fitness가 우상향하지 않는 문제

#### 2. fitness가 우상향하지 않음

##### • 원인

- fitness의 크기
  - 기존 코드에서의 fitness는 0~100
  - fitness의 크기에 따라 결과가 달라지지 않음
- 실험 (fitness 변경)
  - 경로의 역수
  - 경로의 역수 \* 100



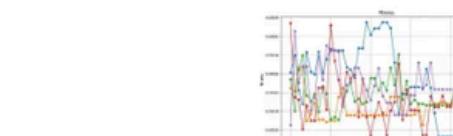
#### 3. $\alpha$ 값 실험 (스케줄러를 달리했을 때)

##### ▼ 실험

- DDIMScheduler
  - 개체군 5, 10번 반복 (최적 경로): 351.19
  - 개체군 5, 10번 반복 (최적 경로): 337.14
- DDIMSchedulerCosine
  - 개체군 5, 10번 반복 (최적 경로): 363.65
  - 개체군 5, 50번 반복 (최적 경로): 317.67



- b. 개체군 5, 50번 반복 (최적 경로): 363.65
- b. 개체군 5, 50번 반복 (최적 경로): 317.67



- 실험 결과 요약
  - $\alpha$ 값이 애매할 때 fitness가 좋음 (하지만 모든 경로에서 그렇다고 볼 수 없음)
  - 스케줄러에 따른 차이가 크지 않지만 DDIMSchedulerCosine이 더 좋은 fitness를 보임

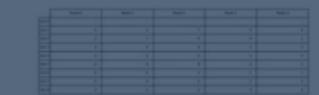
#### 2. noise 크기

- 너무 커서 수렴이 안 될 것
  - noise 정의: 랜덤 노이즈의 기중치인  $\sigma_t$ 의 크기 제어 변수
  - ▶ 논문 Appendix에서는 대부분의 실험에서 noise = 1로 설정해 사용했다고 언급

- 실험 ( $\text{random\_seed} = 0$ , 노드 수 10개, 개체군 크기 5개,  $\text{num\_steps} = 10$ )
  - + 데이터셋: 노드 10개인 데이터셋

##### 1. noise = 0.1 (기본값)

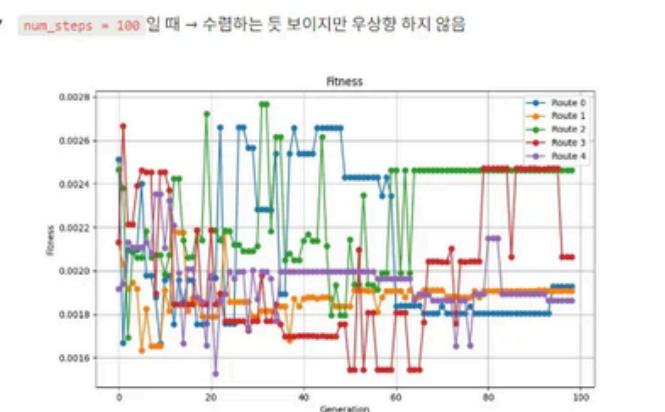
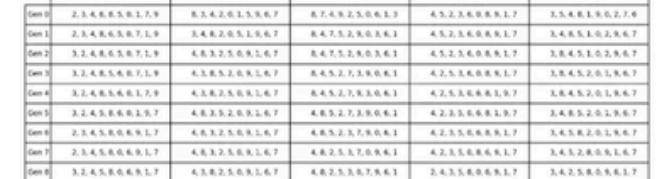
- 적합도
  - 수렴하지 않음
  - 경로에 따라 다르지만 초반엔 7~8개, 후반엔 2~4개



#### ▼ 경로 순서 자세히 보기

- DDIMScheduler
  - 개체군 5, 10번 반복 (최적 경로): 351.19
  - 개체군 5, 10번 반복 (최적 경로): 337.14

#### 3. num\_steps = 100 일 때 → 수렴하는 듯 보이지만 우상향 하지 않음



## DiffEvo를 활용한 TSP task 해결

---

결론: 연속값을 노드 번호로 매핑하는 것에서부터 문제가 발생한다!

	Route 0	Route 1	Route 2	Route 3	Route 4
Gen 0	-	-	-	-	-
Gen 1	0	2	2	2	2
Gen 2	2	2	3	0	2
Gen 3	4	0	0	4	4
Gen 4	4	2	2	4	2
Gen 5	4	0	0	4	0
Gen 6	2	2	0	0	2
Gen 7	0	0	0	0	0
Gen 8	0	0	0	0	0

	Route 0	Route 1	Route 2	Route 3	Route 4
Gen 0	7, 9, 1, 0, 2, 6, 3, 8, 5, 4	7, 6, 5, 2, 0, 1, 8, 9, 4, 3	1, 5, 2, 6, 7, 4, 0, 8, 9, 3	1, 2, 7, 6, 9, 5, 3, 8, 0, 4	4, 8, 5, 1, 7, 9, 0, 2, 6, 3
Gen 1	7, 9, 1, 0, 2, 6, 3, 8, 5, 4	7, 6, 5, 2, 1, 0, 8, 9, 4, 3	1, 5, 6, 2, 7, 4, 0, 8, 9, 3	1, 7, 2, 6, 9, 5, 3, 8, 0, 4	5, 8, 4, 1, 7, 9, 0, 2, 6, 3
Gen 2	7, 9, 1, 0, 6, 2, 3, 8, 5, 4	7, 6, 5, 1, 2, 0, 8, 9, 4, 3	1, 5, 7, 6, 2, 4, 0, 8, 9, 3	1, 7, 2, 6, 9, 5, 3, 8, 0, 4	5, 8, 4, 7, 1, 9, 0, 2, 6, 3
Gen 3	7, 1, 9, 0, 6, 2, 5, 8, 3, 4	7, 6, 5, 1, 2, 0, 8, 9, 4, 3	1, 5, 7, 6, 2, 4, 0, 8, 9, 3	1, 7, 2, 6, 5, 9, 8, 3, 0, 4	5, 7, 8, 1, 4, 9, 0, 2, 6, 3
Gen 4	7, 1, 9, 2, 6, 0, 5, 8, 4, 3	7, 6, 5, 1, 2, 8, 0, 9, 4, 3	1, 7, 5, 6, 2, 4, 0, 8, 9, 3	7, 1, 2, 6, 5, 9, 8, 0, 3, 4	5, 7, 1, 8, 4, 9, 0, 2, 6, 3
Gen 5	7, 1, 9, 6, 5, 2, 0, 8, 4, 3	7, 6, 5, 1, 2, 8, 0, 9, 4, 3	1, 7, 5, 6, 2, 4, 0, 8, 9, 3	7, 1, 5, 6, 2, 9, 8, 0, 4, 3	5, 7, 1, 8, 4, 9, 0, 2, 6, 3
Gen 6	7, 1, 5, 6, 9, 2, 0, 8, 4, 3	7, 6, 5, 1, 2, 8, 9, 0, 4, 3	1, 7, 5, 6, 2, 4, 0, 8, 9, 3	7, 1, 5, 6, 2, 9, 8, 0, 4, 3	7, 5, 1, 8, 4, 9, 0, 2, 6, 3
Gen 7	7, 1, 5, 6, 9, 2, 0, 8, 4, 3	7, 6, 5, 1, 2, 8, 9, 0, 4, 3	1, 7, 5, 6, 2, 4, 0, 8, 9, 3	7, 1, 5, 6, 2, 9, 8, 0, 4, 3	7, 5, 1, 8, 4, 9, 0, 2, 6, 3
Gen 8	7, 1, 5, 6, 9, 2, 0, 8, 4, 3	7, 6, 5, 1, 2, 8, 9, 0, 4, 3	1, 7, 5, 6, 2, 4, 0, 8, 9, 3	7, 1, 5, 6, 2, 9, 8, 0, 4, 3	7, 5, 1, 8, 4, 9, 0, 2, 6, 3

### 1. DiffEvo가 최적화 방향성을 잡지 못한다.

- 실제로 경로가 어떻게 변화하는지 뽑아봤을 때 기준 없이 노드 번호가 마구잡이로 변화하는 것을 볼 수 있음 형식

### 2. 연속값과 이산값의 차이에서 오는 한계

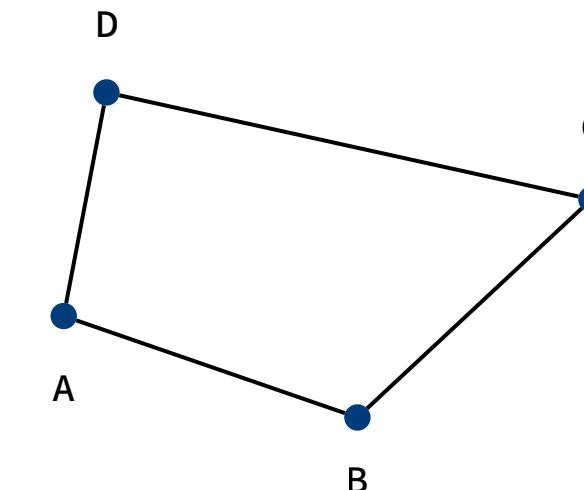
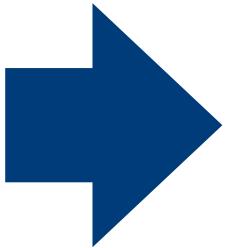
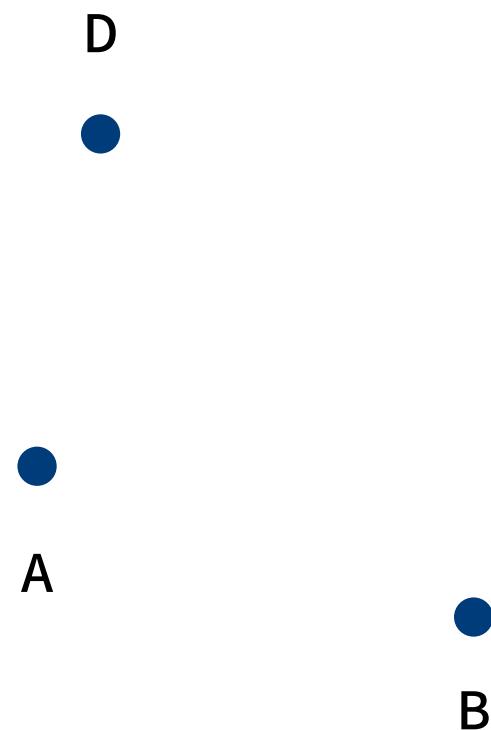
- 연속값에서 차이가 크지 않지만 대소 관계를 따져 정수값으로 매핑한다고 했을 때 작은 차이로 경로가 바뀜
- 실험 중 최적화 후반부 노이즈가 거의 줄어드는 시점, 즉 경로가 많이 바뀌면 안 되는 상황에서 노드가 9개, 10개씩 바뀌기도 함

### 3. 가우시안 확률을 구할 때의 한계

- 원본 데이터가  $x_t^i$ 일 때 멀면 확률을 작게, 가까우면 확률을 크게 주어  $x_0$ 을 추정할 때 가중치를 넣어줌
- 연속값이면 평균을 원본 데이터  $\sqrt{\alpha_t}x_t^i$ 로 뒀을 때  $\sqrt{\alpha_t}x_t^i \neq x_t^i$ 이지만
  - 좌표 거리가 먼지 가까운지 정보를 넣어주려고 노드 번호로 매핑해주면  $\sqrt{\alpha_t}x_t^i = x_t^i$  (단조 증가라 대소 관계가 변하지 않음) 가 되어 원본 데이터가 자기 경로일 때 자기 자신이 관찰될 확률이 1이 되어  $x_0$ 의 추정값이 잘못 구현됨

## DiffEvo를 활용한 TSP task 해결

### 추후 피드백



개체 = 경로

0.7	0.08	0.64	0.85	0.3	0.57
$\overline{AB}$	$\overline{AC}$	$\overline{AD}$	$\overline{BC}$	$\overline{BD}$	$\overline{CD}$

노드 순서가 아닌 엣지 연결 여부를 Chromosome으로 정의

→ TSP의 순환 특성은 노드 순서보다

어떤 연결선을 선택할지가 더 중요



Part 2

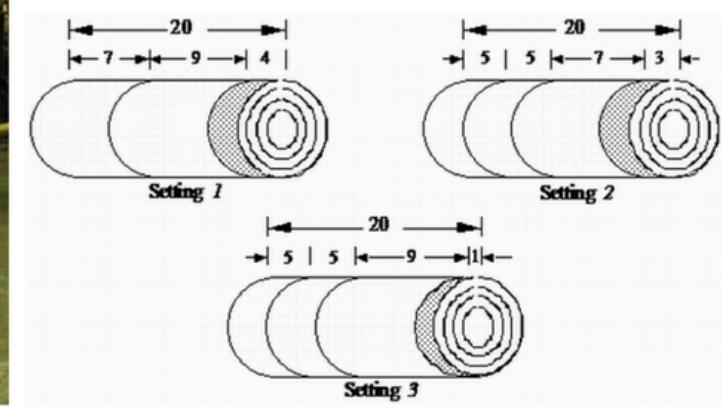
# 1.5D CSP Project

- 1D CSP(Cutting Stock Problem) 이해
- 1.5D CSP 문제 재정의 및 2-Stage 방법론 탐색

# 1D CSP(Cutting Stock Problem) 이해

## Cutting Stock Problem

- 큰 Material을 여러 개의 Product로 자른다고 할 때 Trim loss와 Material 사용량을 최소화하는 문제
- 최악의 경우: 하나의 Material에서 하나의 Product를 자르는 경우



## CSP 해결 방법론

### 1. Integer programming formulation

- 각 Product에 번호가 정해져있고 해당 Product를 특정 Material에서 자를 건지 여부와 해당 Material을 사용할지 여부를 이진으로 나타냄

$x_{ij} \in \{0, 1\}$ : 조각  $j$ 를 raw  $i$ 에서 자를지 여부

$y_i \in \{0, 1\}$ : raw  $i$ 를 사용할지 여부

- 제약 조건을 지키면서 사용하는 Material 수를 최소화시키고 해를 찾음
- 단점
  - 변수가 너무 많아서 비효율적
  - 같은 종류의 Product를 여러 개의 Material에 넣는다고 할 때 같은 결과가 여러 방식으로 표현될 수 있어 중복된 경우를 탐색하게 됨  
→ Material1에서  $s_1$ 을 2번 자르는 것은 Material 2에서  $s_1$ 을 2번 자르는 것과 동일

### 2. New Integer programming formulation

- 하나의 Material에서 자를 수 있는 경우의 수를 Pattern이라고 놓고 해당 패턴을 사용해 최적화  
→ 중복된 경우를 탐색하지 않아도 되어 변수가 줄어듦

$$W = 10, s_1 = 5, s_2 = 3, s_3 = 2$$

$$\begin{bmatrix} 2 \\ 0 \\ 0 \end{bmatrix} x_1 + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} x_2 + \dots + \begin{bmatrix} 0 \\ 2 \\ 2 \end{bmatrix} x_7 \geq \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

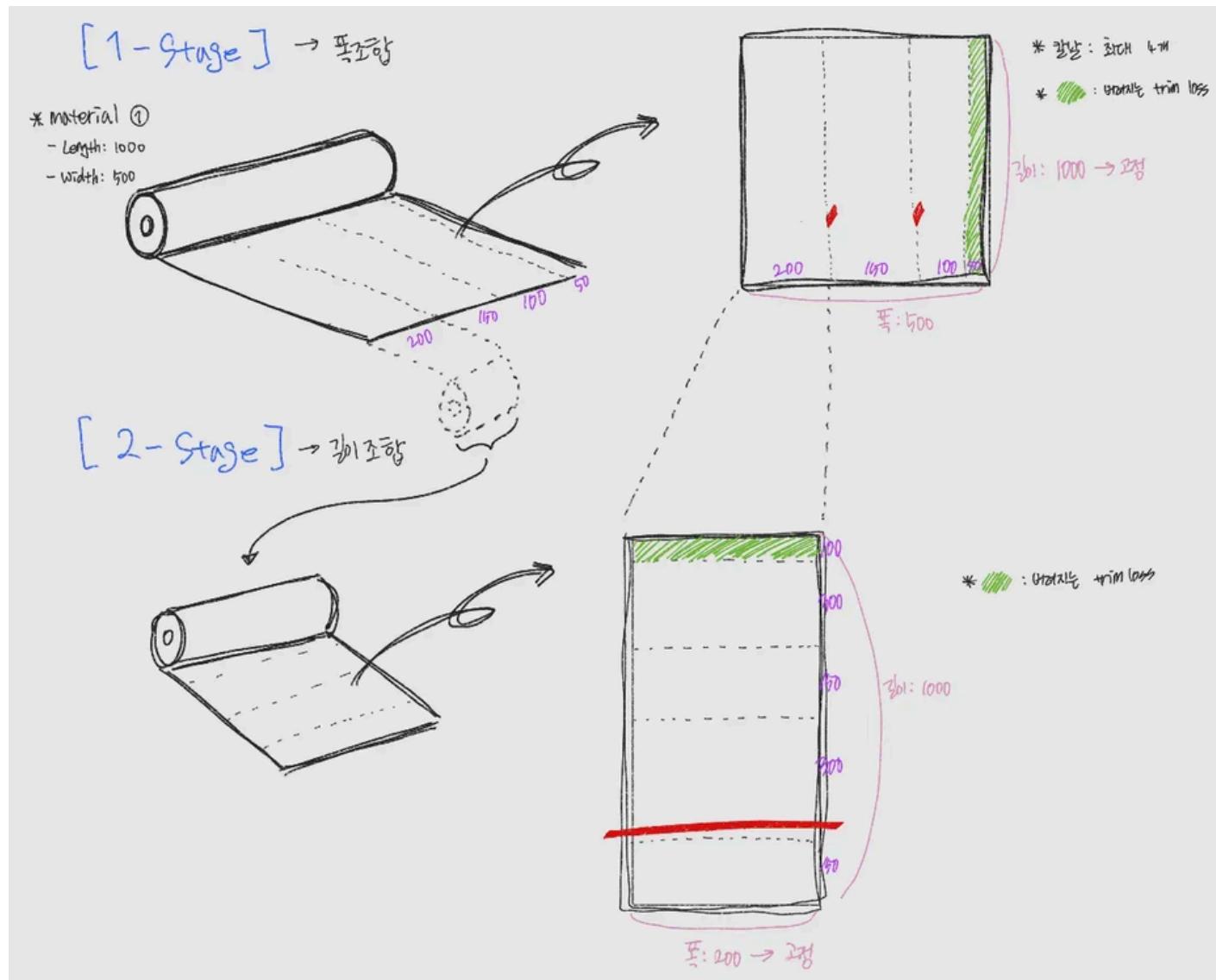
- 단점
  - 모든 가능한 패턴을 미리 만들어놓고 계산해야 하기 때문에 패턴이 너무 많아짐

### 3. Column Generation for the Cutting Stock Problem

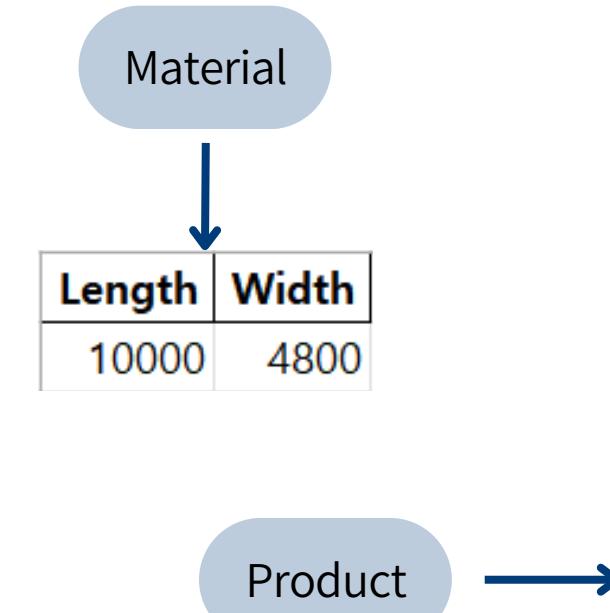
- 처음엔 조금의 패턴만 가지고 시작하고 필요한 새로운 패턴만 하나씩 찾아서 추가해가면서 푸는 방식

## 1.5D CSP 문제 재정의 및 2-Stage 방법론 탐색

### 2-Stage 1.5D CSP 문제 재정의



### 데이터셋 생성



product id	Length	Width	Demands
1	834	863	387
2	930	1337	46
3	894	1576	15
4	816	642	153
5	799	642	222
6	909	786	206
7	828	1576	273
8	788	1595	382
9	875	1595	236
10	851	633	340
11	1060	671	228
12	1005	863	9
13	1084	1595	166
14	631	1576	319
15	653	642	214
16	1019	1595	117
17	657	642	386
18	758	1595	344

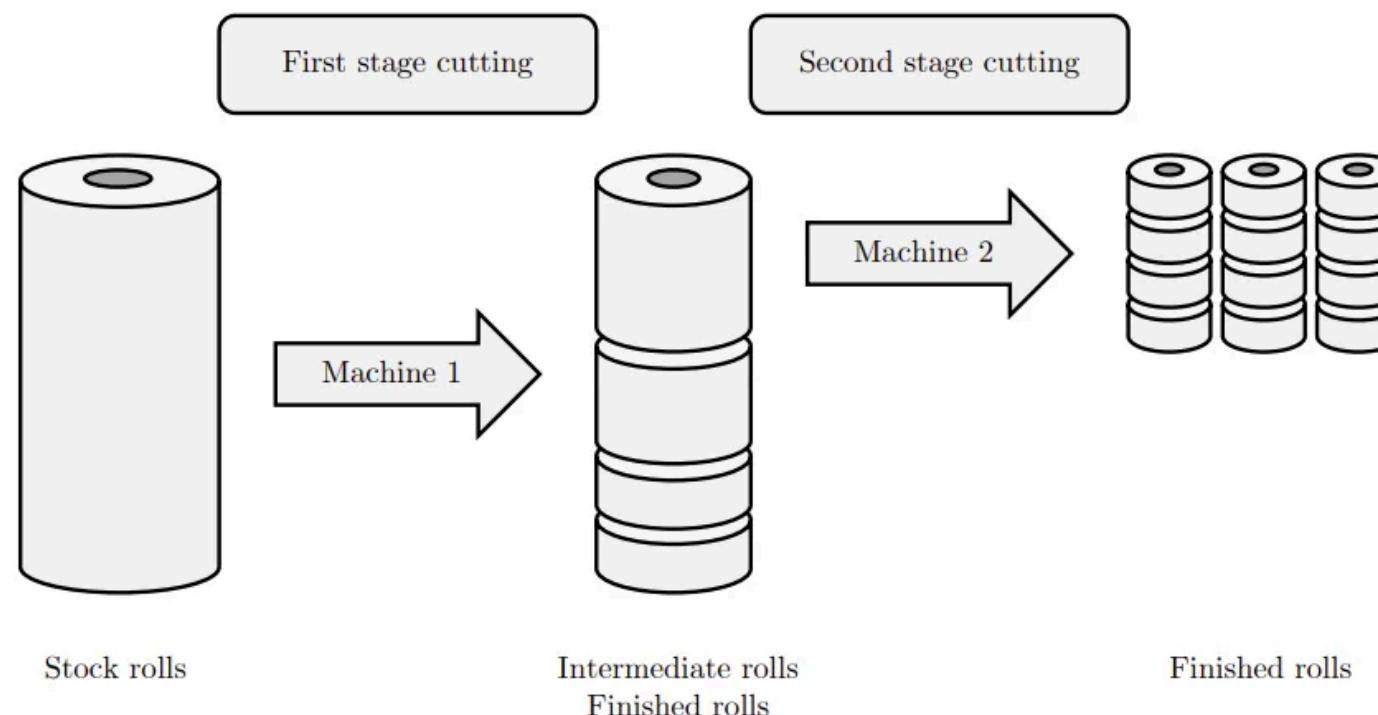
- 폭 조합 후 길이 조합을 진행
- 2D CSP와의 차이점: Length와 Width를 동시에 고려하는 2D CSP와 다르게 Width 와 Length를 따로 따로 고려하는 방법

- 기존 한솔 제지 솔루션을 구축할 때 샘플 데이터 (천안 공장 지폭 데이터) 존재  
→ 해당 데이터에서 시트지 데이터들을 뽑아서 데이터 범위를 설정  
→ 실제 데이터셋 분포를 반영한 데이터셋을 베이스라인으로 사용

## 1.5D CSP 문제 재정의 및 2-Stage 방법론 탐색

### 참고한 선행연구: Algorithmic Advances for 1.5 D Two-Stage Cutting Stock Problem

- 2-Stage에 걸쳐 폭을 2번 절단
  - 1-Stage에서 잘리는 것 : 중간롤 → Product를 1단계에서 자를 수 없음
  - 2-Stage에서 잘리는 것: 중간롤에서 잘린 Product
- 2-Stage 제약 행렬을 사용해 어떤 중간롤에서 어떤 2-Stage 패턴이 잘리는지 표현  
→ Linking 제약 (연결 제약) 이라고 표현



**Figure 1.** 1.5D Two-Stage Slitting process

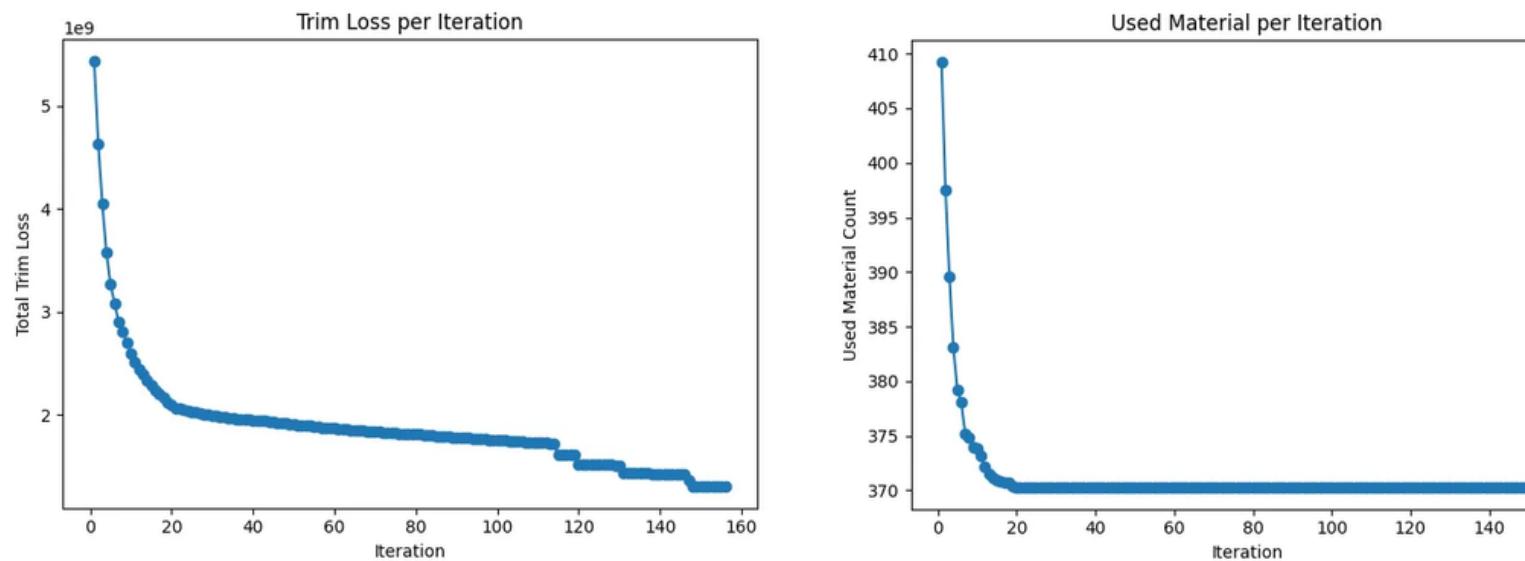
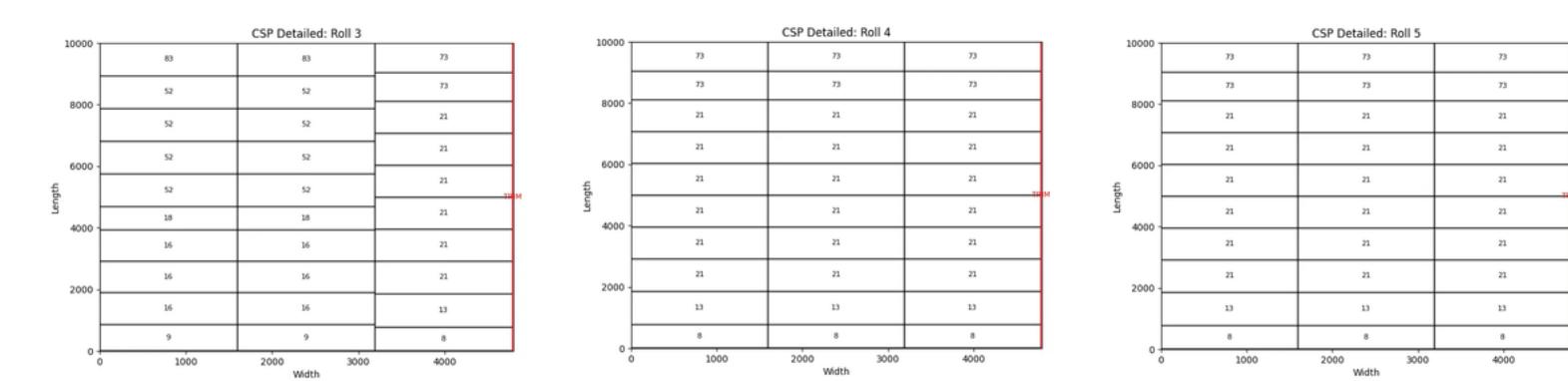
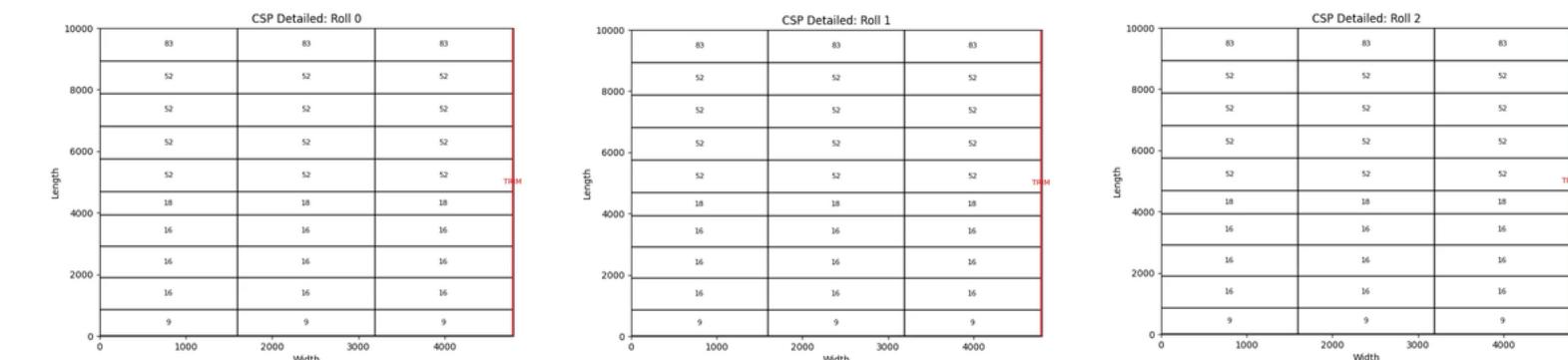
1. 최소한의 패턴으로 초기화한 후 RSRMP 계산 (Solver 사용)
2. 모든 가능한 중간 롤 길이 중 하나를 선택  
→ 1S-CG-PSP: 음의 reduced cost를 갖는 새로운 1단계 패턴을 탐색
3. 중간 롤 중 하나를 선택해 해당 중간 롤로 만들 수 있는 product 패턴 탐색  
→ 음의 reduced cost를 갖는 새로운 2단계 패턴을 탐색  
~~~~~ 반복 ~~~~
4. 기존 정해진 중간 롤 조합에서 더 이상 개선이 안 된다고 판단  
→ RCG-PSP: 새로운 중간 롤을 생성

# 1.5D CSP 문제 재정의 및 2-Stage 방법론 탐색

## Solver를 활용한 방법론 구현 (Python PuLP 사용)

- CBC Solver 활용  
→ Branch-and-bound 방식을 사용해 정수 및 선형 제약 조건을 만족하는 최적해를 찾는 Solver
- 단점
  - a. 계산 시간 ↑
  - b. 반복적인 Solver 호출
  - c. 데이터가 크면 최적해를 보장할 수 없음

----- 10000초 초과 ➔ 중단  
전체 사용한 material 수: 377.0  
총 trim loss: 1279770461.84  
총 소요 시간: 5시간 18분  
총 iter: 156



## 1.5D CSP 문제 재정의 및 2-Stage 방법론 탐색

### CBC Solver 하이퍼파라미터 실험

- 데이터셋을 조금 작게 만들어서 하이퍼 파라미터 튜닝 진행  
→ 드라마틱한 변화 X

| Aa 이름                            | trim loss     | used material | 하이퍼 파라미터                                                                                                | 소요 시간  |
|----------------------------------|---------------|---------------|---------------------------------------------------------------------------------------------------------|--------|
| 베이스라인 + (strong = 15)            | 603145418.87  | 46            | cuts=None gapRel=0.1 gapAbs=None maxNodes=None options = []<br>presolve=None warmStart=True strong=15   | 409.84 |
| 베이스라인 + (strong = 10)            | 708356233.3   | 48            | cuts=None gapRel=0.1 gapAbs=None maxNodes=None options = []<br>presolve=None warmStart=True strong=10   | 412.14 |
| 베이스라인 + (strong = 5)             | 760961640.52  | 49            | cuts=None gapRel=0.1 gapAbs=None maxNodes=None options = []<br>presolve=None warmStart=True strong=5    | 408.72 |
| 베이스라인 + (strong = 20)            | 813567047.73  | 50            | cuts=None gapRel=0.1 gapAbs=None maxNodes=None options = []<br>presolve=None warmStart=True strong = 20 | 408.40 |
| 베이스라인                            | 1003969477.79 | 54            | gapAbs=None presolve=None cuts=None strong=None<br>maxNodes=None options = [] gapRel=0.1 warmStart=True | 410.54 |
| 베이스라인 + presolve 활성화             | 1003969477.79 | 54            | cuts=None gapRel=0.1 gapAbs=None maxNodes=None options = []<br>strong=None warmStart=True presolve=True | 407.89 |
| 베이스라인 + (maxNodes=5000)          | 1003969477.79 | 54            | cuts=None gapRel=0.1 gapAbs=None options = [] presolve=None<br>strong=None warmStart=True maxNodes=5000 | 190.93 |
| 베이스라인 + cuts 활성화                 | 1198193862.08 | 58            | gapAbs=None presolve=None strong=None maxNodes=None<br>options = [] gapRel=0.1 warmStart=True cuts=True | 413.82 |
| 베이스라인 + (strong = 15) + cuts 활성화 | 1211653248.33 | 58            | gapRel=0.1 gapAbs=None maxNodes=None options = []<br>presolve=None warmStart=True strong=15 cuts=True   | 411.26 |



Column Generation으로 만들어진 패턴을 몇 개 쓸지  
GA를 활용해 최적화시켜보자!

## 1.5D CSP 문제 재정의 및 2-Stage 방법론 탐색

### 참고한 선행연구: Solving multi-depot electric vehicle scheduling problem by column generation and genetic algorithm

- GA와 CG를 결합한 MD-EVSP (multi-depot electric vehicle scheduling problem) 해결 기법을 제안  
→ 최적화 Solver인 CPLEX를 이용해 Column Generation을 진행하고 컬럼들의 집합에서 최종 해를 구성할 컬럼들을 선택하기 위해 GA 사용
- Chromosome 표현  
→ CG로 만들어진 모든 칼럼들에 번호를 붙이고 해당 경로를 포함할지 말지 이진으로 표현

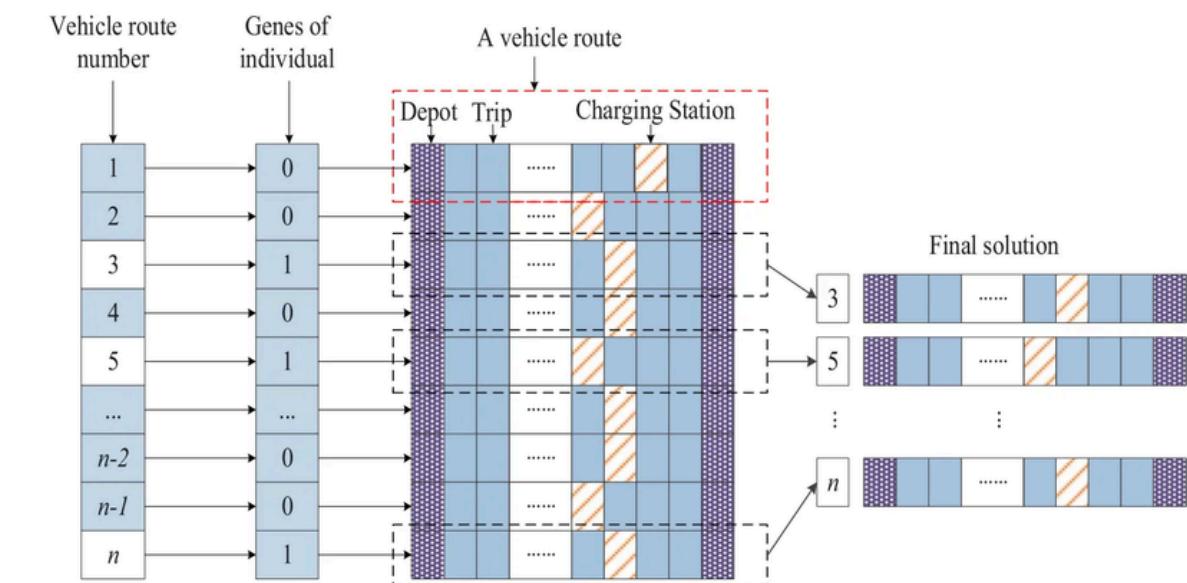


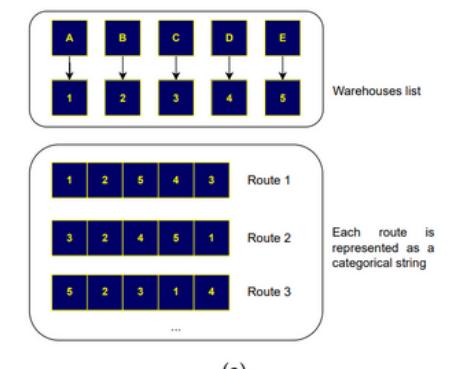
Fig. 4. The solution coding scheme.

### 2-Stage GA 선행연구

- An Efficient Two-Stage Genetic Algorithm for Flexible Job-Shop Scheduling
  - 생산/제조 공정에서 각 작업을 어떤 순서, 어떤 기계에서 처리할지를 동시에 결정하는 스케줄링 최적화 문제
  - 1단계에서 작업 순서를 빠르게 최적화하고 2단계에서는 그걸 초기해로 받아 작업 순서와 기계 선택을 함께 최적화
- Two-Stage Genetic Algorithm for Optimization Logistics Network for Groupage Delivery
  - 지역 창고 위치 선정과 차량 경로를 동시에 최적화하여 물류 네트워크를 최적화하는 문제
  - 1단계에서는 후보 도시들 중 어디에 지역 창고를 둘지 선택하고 2단계에서는 선택된 창고 위치 + 고객 위치와 수요량을 파악해 배송 경로를 최적화

|                 |       |       |       |       |       |       |
|-----------------|-------|-------|-------|-------|-------|-------|
| 2,1             | 3,1   | 1,1   | 2,2   | 2,3   | 1,2   | 3,2   |
| Stage 1 (j,o)   |       |       |       |       |       |       |
| 2,1,3           | 3,1,1 | 1,1,2 | 2,2,1 | 2,3,3 | 1,2,2 | 3,2,1 |
| Stage 2 (j,o,m) |       |       |       |       |       |       |

Fig. 1. Solution representation example for stage 1 and 2

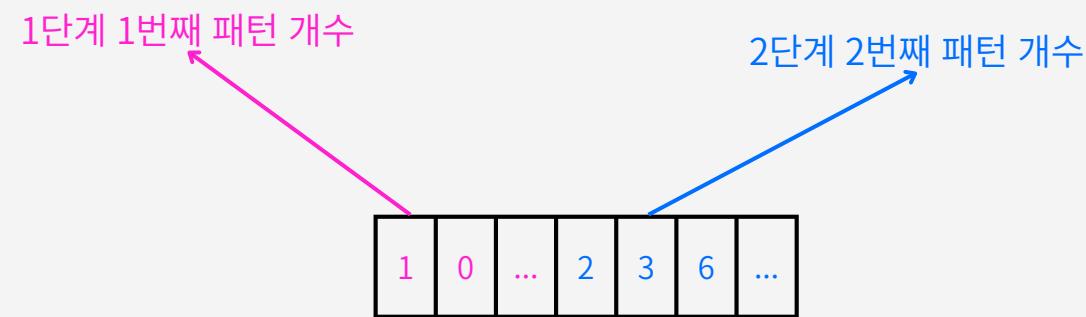


# 1.5D CSP 문제 재정의 및 2-Stage 방법론 탐색

## 아이디어 적용 및 코드 구현

### Chromosome

```
# J = 1단계 패턴 개수, R = 2단계 패턴 개수  
# x1_j: 1단계 패턴 j개 사용 횟수  
# x2_r: 2단계 r개 사용 횟수  
  
chromosome = [ x1_1, x1_2, ..., x1_J,      x2_1, x2_2, ..., x2_R ]  
              ← Stage1 →          ← Stage2 →
```



\* Column Generation에서 사용했던 연결 제약을 통해 어떤 2단계 패턴이 어떤 중간률에서 잘리는지 정보를 제공한다.

### Chromosome Initialization

- LP 해들을 가장 성능이 좋은 해부터 개체로 만들어서 GA 초기 개체로 사용
- 만약 Population이 LP 해의 크기보다 크다면 Random 초기화
  - 1단계 Genes: 0 (min) ~ 패턴에 포함된 폭 중 폭에 필요한 총 길이를 Material 길이로 나눈 값에 기반 (max)
  - 2단계 Genes: 0 (min) ~ 폭이 같은 중간률의 생산량 안에서 Random 분배

### Fitness

Solver에서의 Objective값과 동일하게 진행

→ trim loss + penalty\_weight × 사용된 Material 개수

(penalty\_weight = Material 한 장을 사용했을 때 발생하는 평균 trim loss)

### 예시: Material 1

| Product a | Product c | Product z | ” |
|-----------|-----------|-----------|---|
| Product a | Product c | Product z | ” |
| Product b | Product d | Product x | ” |
| Product a | Product c | Product z | ” |

폭 A                          폭 B                          폭 C                          폭 C

1단계 패턴 = [A, B, C, C]  
2단계 패턴 = [a, b, a, a],  
[c, c, d, c],  
[z, x, z, z],  
[z, x, z, z]

# 1.5D CSP 문제 재정의 및 2-Stage 방법론 탐색

## 아이디어 적용 및 코드 구현

### | Selection

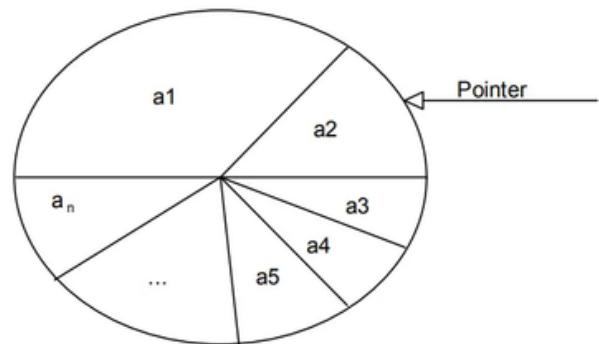


Fig. 2. Roulette wheel selection [22]

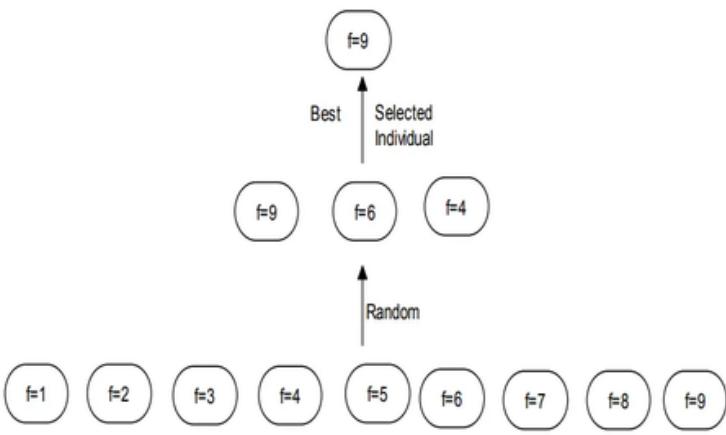


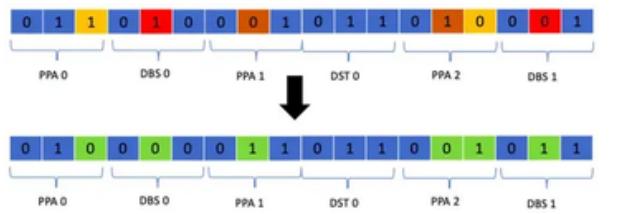
Fig. 3. Tournament Selection Mechanism [15]

### | Mutation

- 1단계 Mutation
  - 가장 비효율적인 1단계 패턴의 gene 값을  $-1$
  - 가장 효율적인 1단계 패턴의 gene 값을  $+1$
- 2단계 Mutation:  
폭 종류 중 랜덤으로 하나 선택해 해당 폭의 중간룰을 사용하는 2단계 패턴 중...
  - 가장 비효율적인 2단계 패턴의 gene 값을  $-1$
  - 가장 효율적인 2단계 패턴의 gene 값을  $+1$

### | Crossover

- 1단계 genes: One-Point Crossover
- 2단계 genes: Two-Point Crossover  
→ 여기선 two-point의 범위가 동일 폭으로 제한됨 (즉, 동일한 폭을 가지는 2 단계 패턴 그룹 내에서 랜덤한 두 지점을 골라 그 사이 구간만 Crossover)
- 참고한 선행연구



→ 한 학생이 클래스에 배정되면 특정 확률에 따라 해당 클래스를 동일한 유형의 다른 클래스와 서로 바꿈 (Mutation)

### | Repair

1. 수요 제약 만족  
→ 수요 제약을 만족하지 않는 해 = 미생산된 Product를 가지고 있는 해
  - 수요가 부족한 Product를 찾고 해당 Product를 생산하는 2단계 패턴 중 가장 효율적인 패턴을 선택해 생산량을 부족한 만큼 증가시킴
2. 연결 제약 만족  
→ 연결 제약을 만족하지 않는 해 = 생산된 중간룰보다 절단되는 중간룰이 많은 해
  - 연결 제약을 위반하는 경우 해당 폭의 중간룰을 생산하는 1단계 패턴 중 가장 효율적인 패턴을 선택해 필요한 만큼 증가시킴

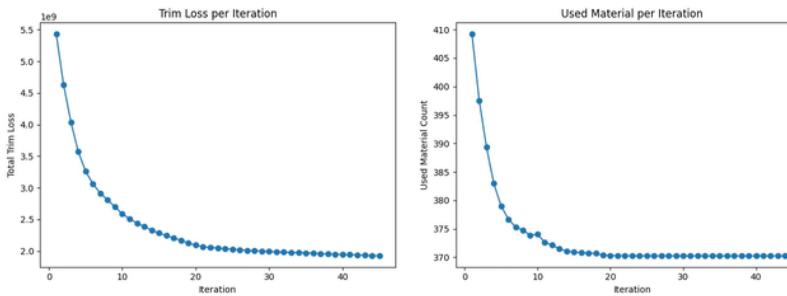
# 1.5D CSP 문제 재정의 및 2-Stage 방법론 탐색

## GA로 수행한 결과

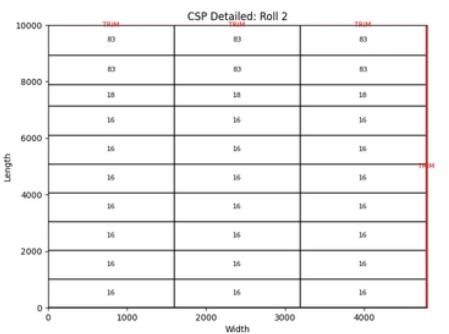
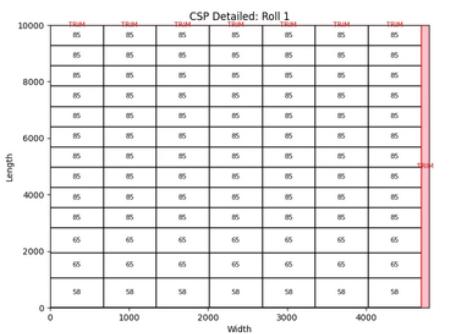
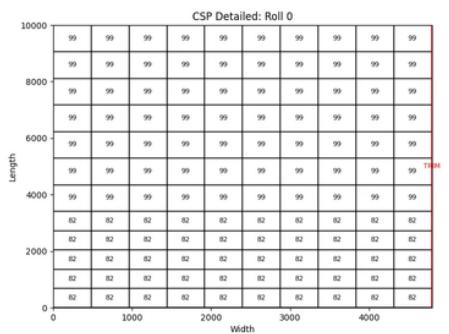
pop\_size = 100, generartion = 100

### 수요 충족만 하면 됨 (과생산 최소화)

최종 objective: 1979693293.64  
전체 사용한 material 수: 381  
전체 Trim Loss: 17227264  
총 과생산량: 401  
전체 수요 대비 과생산율: 1.95%



- ▶ GA 소요 시간: 48.92초
- ▶ 총 소요시간: 149.78초

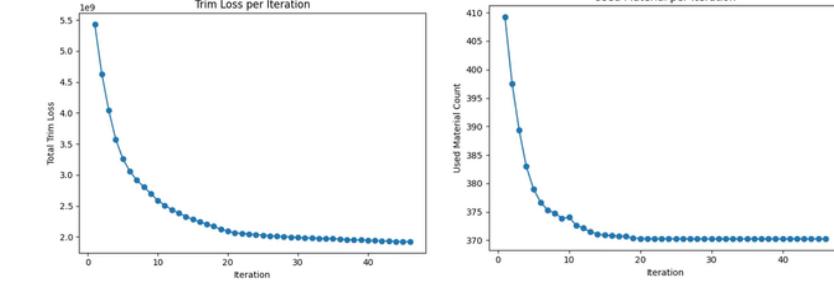


## Solver로 수행한 결과

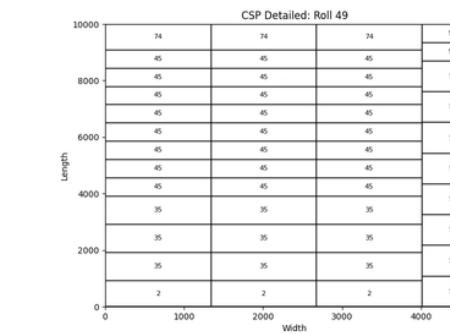
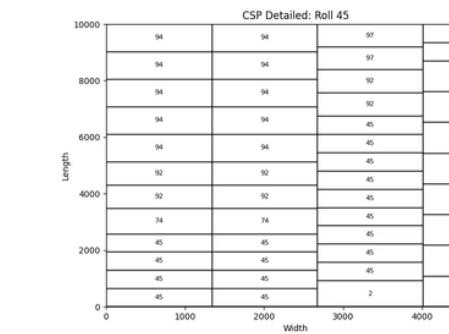
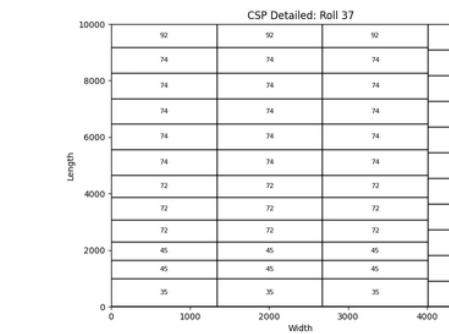
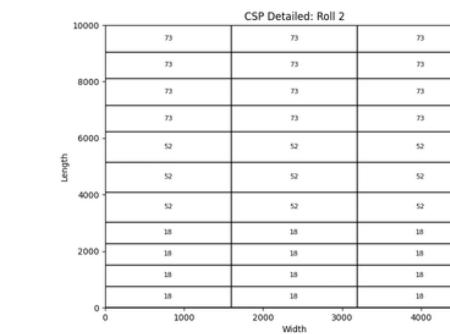
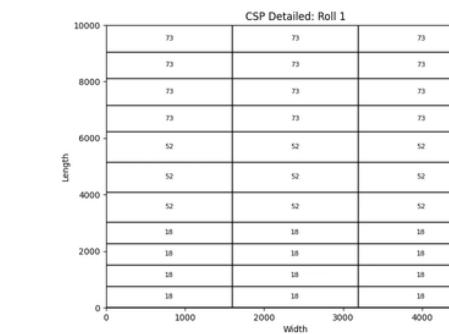
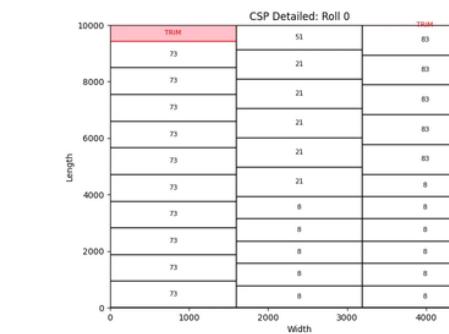
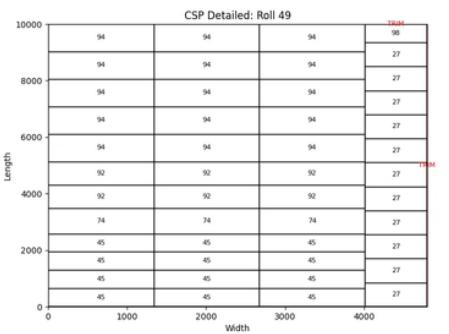
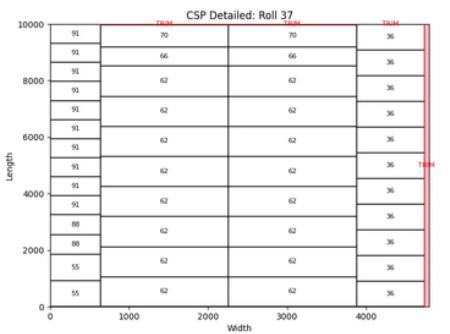
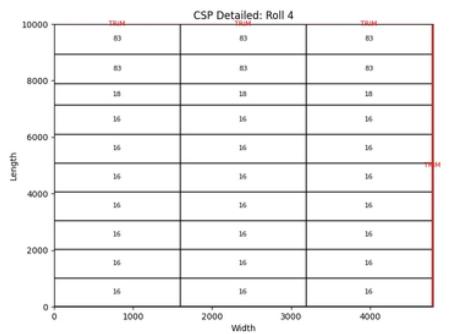
Early stopping 기준: LP 100초, ILP 100초

### 수요 충족만 하면 됨 (과생산 최소화)

최종 objective: 1993991747.94  
전체 사용한 material 수: 384  
전체 Trim Loss: 21008360  
총 과생산량: 961  
전체 수요 대비 과생산율: 4.67%



- ▶ ILP 소요 시간: 102.27초
- ▶ 총 소요시간: 205.56초



## 1.5D CSP 문제 재정의 및 2-Stage 방법론 탐색

### GA로 수행한 결과

최종 objective: 1979693293.64

전체 사용한 material 수: 381

전체 Trim Loss: 17227264

총 과생산량: 401

전체 수요 대비 과생산율: 1.95%

▶ GA 소요 시간: 48.92초

▶ 총 소요시간: 149.78초

### Solver로 수행한 결과

최종 objective: 1993991747.94

전체 사용한 material 수: 384

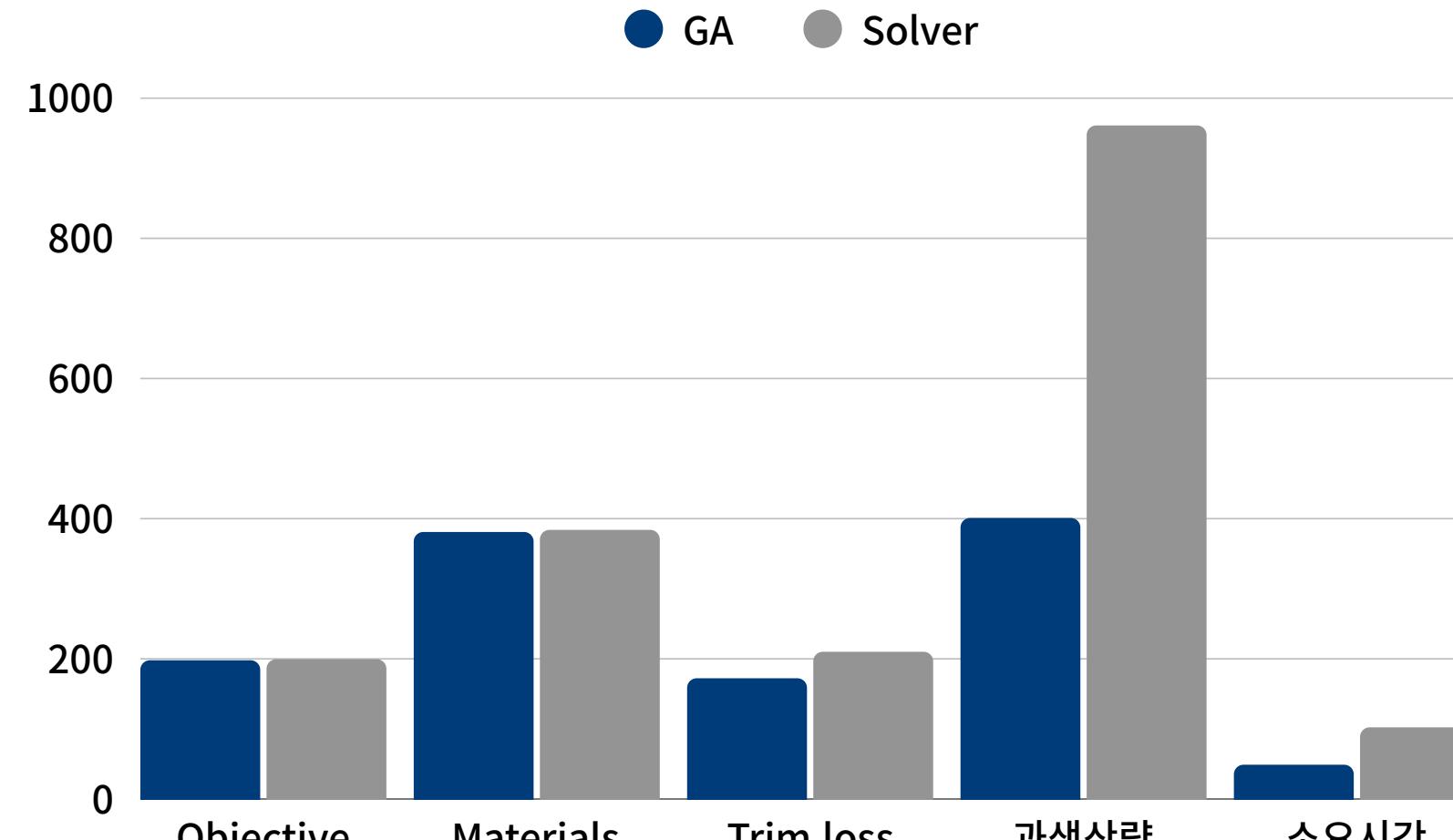
전체 Trim Loss: 21008360

총 과생산량: 961

전체 수요 대비 과생산율: 4.67%

▶ ILP 소요 시간: 102.27초

▶ 총 소요시간: 205.56초

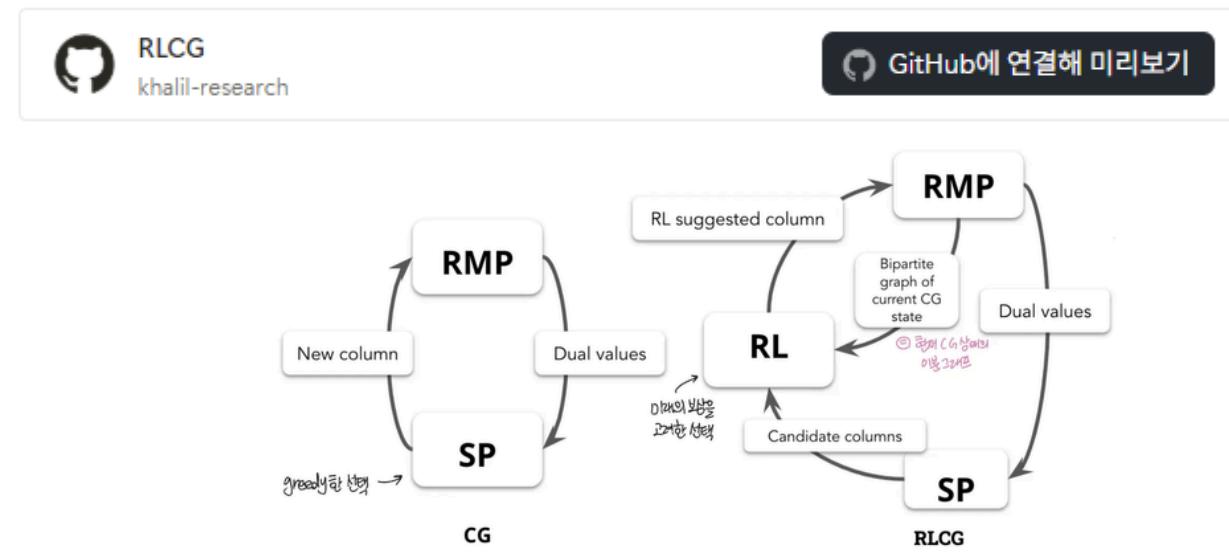


\*시각화를 위해 너무 큰 Objective 과 Trim loss는 일정 비율로 나눠줌

## 1.5D CSP 문제 재정의 및 2-Stage 방법론 탐색

### 개선 방안

- Column Generation은 여전히 Solver를 사용  
→ CG를 solver로 하지 말고 RLCG로 대체



#### RLCG 방법론 요약

1. 초기 전체 패턴 중 일부 간단한 패턴만 선택해 RMP 해결
2. 이를 바탕으로 SP 업데이트
3. DQN으로 현재 RMP와 SP가 생성한 후보 컬럼을 입력 받고  
각 후보 컬럼의 Q-value를 계산해 가장 좋은 컬럼을 선택
4. RMP 다시 계산 후 SP 업데이트
5. CG가 수렴될 때까지 반복  
(= 목표 함수를 개선할 수 있는 음의 reduced cost를 가진 컬럼이 없을 때  
= CG 알고리즘의 반복 과정이 더 이상 개선되지 않는 상태에 도달했을 때)

- 더 많은 제약 사항 반영 가능하게끔 변경
  - 칼날 수 제한
  - 칼날 변경 횟수 최소화
  - Material 종류 여러 개일 때의 최적화
  - ... 등등

- Multi-objective 반영
  - 기존 방법론에 반영되어 있던 Multi-objective
  - 실무에서 원하는 것: 칼날 변경 횟수 최소화
  - 경영진들이 원하는 것: 사용한 Material 수 최소화

# 감사합니다

---

