Travel Route Planner using GA and solving TSP

Group 12

2019145081 Taejun Kang

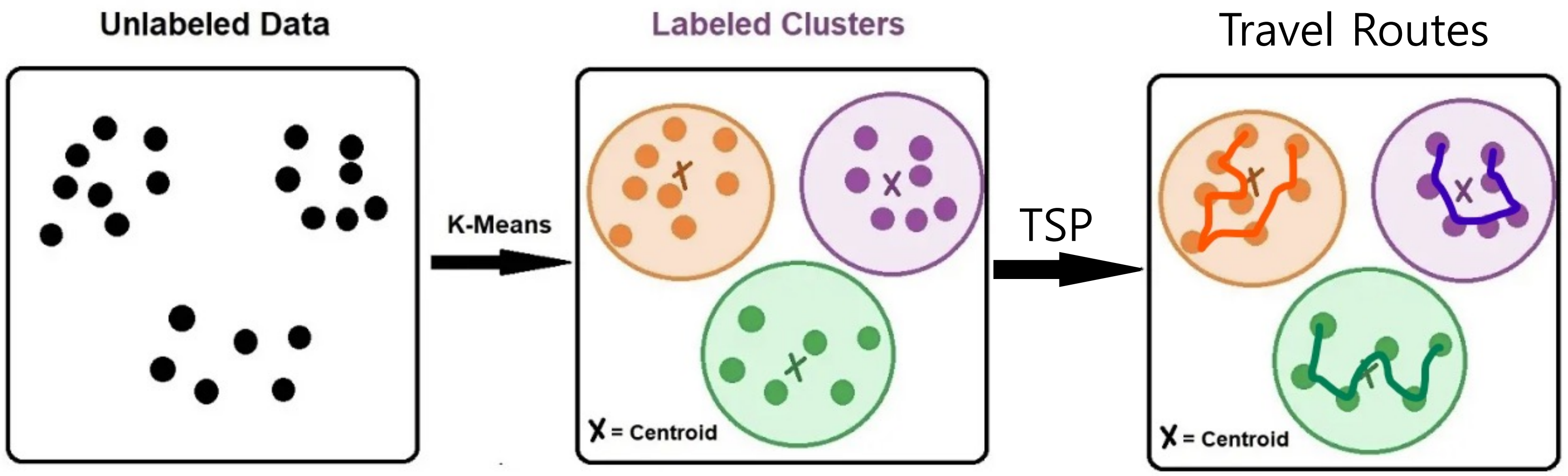
2020147049 Hyundong Kim

Contents

Contribution of the Source Paper

A Development of Travel Itinerary Planning Application using Traveling Salesman Problem and K-Means Clustering Approach. - Rani, S., Kholidah, K. N., & Huda, S. N.

- Algorithm for making travel itinerary using TSP and k-means clustering technique is proposed.

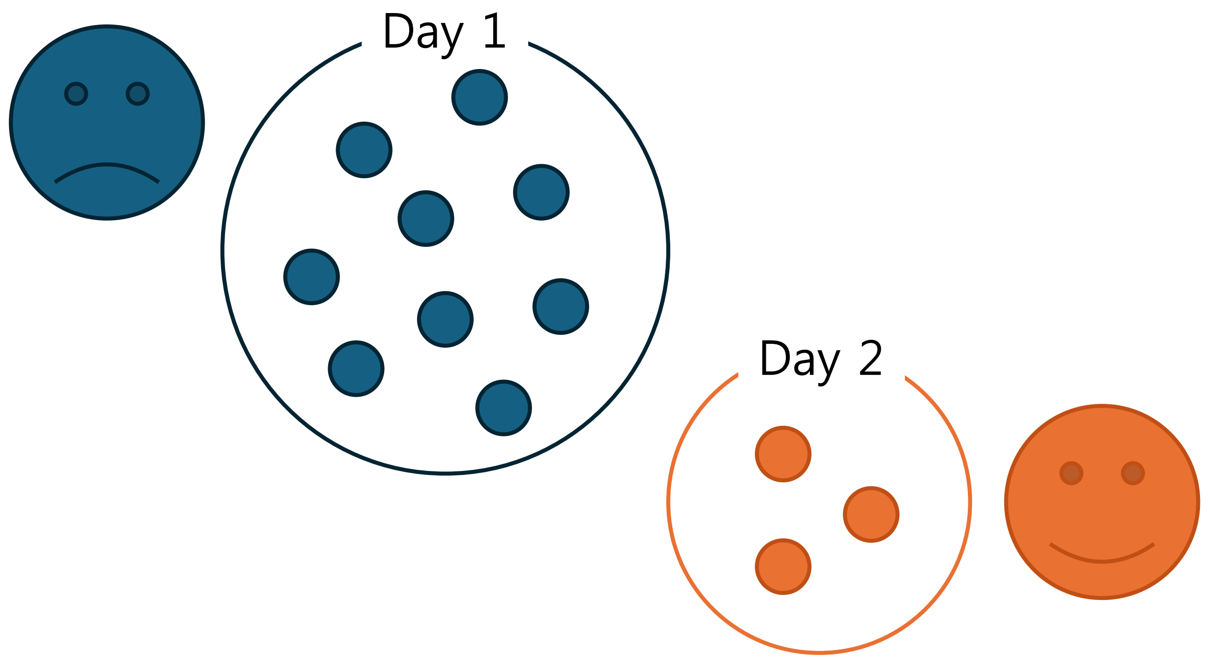


https://www.ejable.com/tech-corner/ai-machine-learning-and-deep-learning/k-means-clustering/

Limitation of the Source Paper

- Clusters are formed based only on the points' distance.
- Without considering the size of clusters

➢Some cluster may contain too many points for daily travel.



https://www.ejable.com/tech-corner/ai-machine-learning-and-deep-learning/k-means-clustering/

Extension

- To overcome the limitation of the original paper,
  - We changed the clustering method from K-means clustering to GA
  - We made the fitness function that considers both total travel time and the deviation between travel time for each cluster(day).

- Fitness Function of GA

  → $\alpha \times$ Total Time of Travel + $(1-\alpha) \times$ Deviation between Travel Time Each Day

Methodology

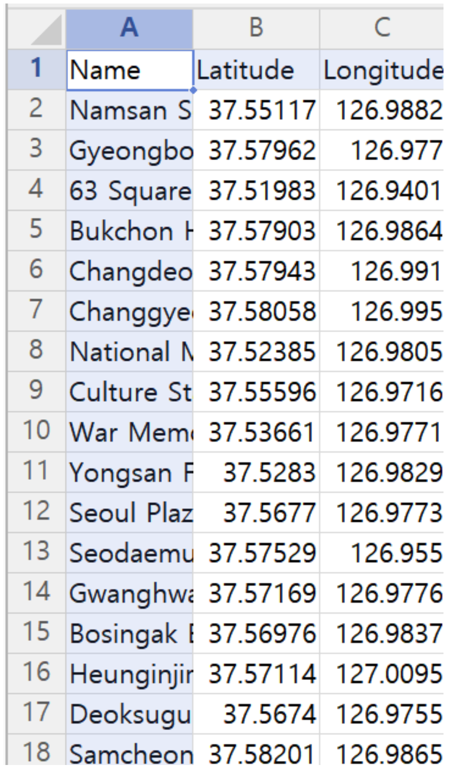
1. Make the initial solution for GA by assigning travel sites to random clusters(days).
2. Set the fitness function as below:

   $\alpha\times$ Total Time of Travel + $(1-\alpha)$ $\times$ Deviation between Travel Time Each Day

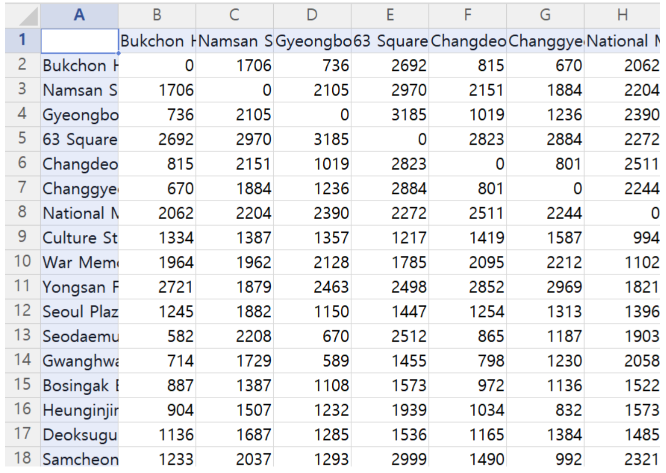
   (Time to travel each cluster is calculated with OR-Tools by solving the TSP)
3. With the fitness function, run GA to find the best cluster combination (which day to visit each sites).
4. Print the result with the travel time of each days.

Data

- Longitude and Latitude of 34 Tourist Attractions (via Google Map)
- Time Duration by Public Transportation between 34 Tourist Attractions (via Google Map)
- List of Attractions
  - 'Bukchon Hanok Village', 'Namsan Seoul Tower', 'Gyeongbokgung Palace', '63 Square', 'Changdeokgung Palace', 'Changgyeonggung Palace', 'National Museum of Korea', 'Culture Station Seoul 284', 'War Memorial of Korea', 'Yongsan Family Park', 'Seoul Plaza', 'Seodaemun Prison History Museum', 'Gwanghwamun Square', 'Bosingak Belfry', 'Heunginjimun Gate', 'Deoksugung Palace', 'Samcheong-dong Alley', 'Bangi-dong Baekje Tombs', 'Myeongdong Cathedral', 'Hwangudan Altar', 'The Blue House', 'National Assembly Building', 'National Library of Korea', 'Bukak Skyway Octagonal Pavilion', 'Banpo Bridge Night View', 'Lotte World Tower', 'Sungnyemun Gate', 'Seoullo 7017', 'Deoksugung Stonewall Walkway', 'Dongdaemun Design Plaza (DDP)', 'Starfield Library', 'Namdaemun Market Tourist Information Center', 'Ikseon-dong Hanok Street'



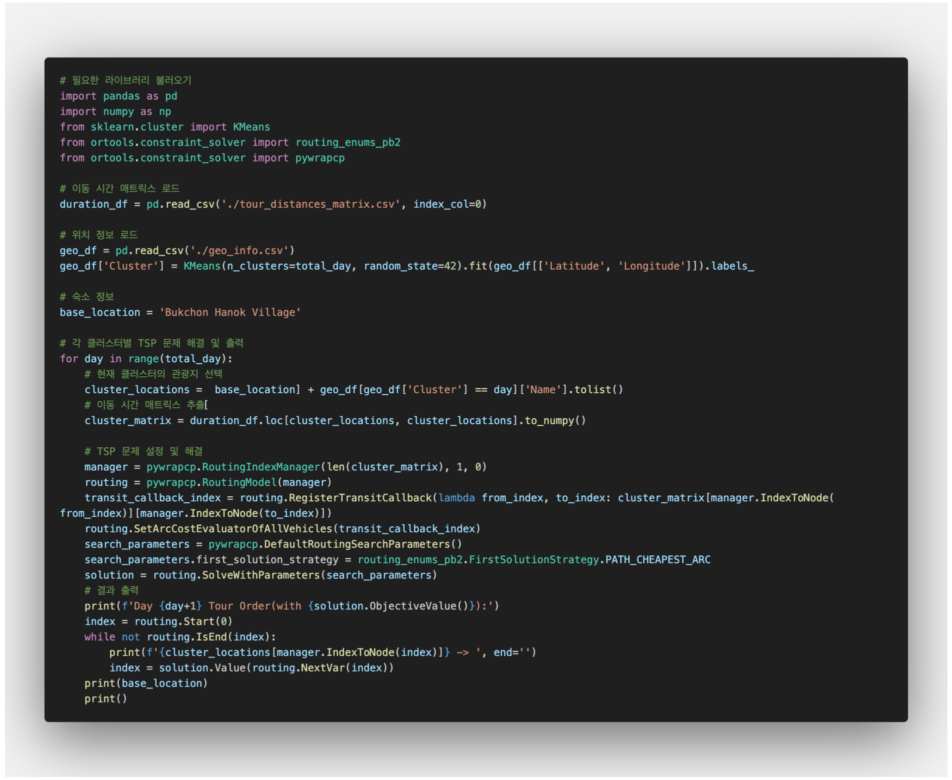
	A	B	C
1	Name	Latitude	Longitude
2	Namsan S	37.55117	126.9882
3	Gyeongbo	37.57962	126.977
4	63 Square	37.51983	126.9401
5	Bukchon H	37.57903	126.9864
6	Changdeo	37.57943	126.991
7	Changgye	37.58058	126.995
8	National M	37.52385	126.9805
9	Culture St	37.55596	126.9716
10	War Mem	37.53661	126.9771
11	Yongsan F	37.5283	126.9829
12	Seoul Plaz	37.5677	126.9773
13	Seodaemu	37.57529	126.955
14	Gwanghwa	37.57169	126.9776
15	Bosingak E	37.56976	126.9837
16	Heunginjir	37.57114	127.0095
17	Deoksugu	37.5674	126.9755
18	Samcheon	37.58201	126.9865



	A	B	C	D	E	F	G	H
1		Bukchon H	Namsan S	Gyeongbo	63 Square	Changdeo	Changgye	National M
2	Bukchon H	0	1706	736	2692	815	670	2062
3	Namsan S	1706	0	2105	2970	2151	1884	2204
4	Gyeongbo	736	2105	0	3185	1019	1236	2390
5	63 Square	2692	2970	3185	0	2823	2884	2272
6	Changdeo	815	2151	1019	2823	0	801	2511
7	Changgye	670	1884	1236	2884	801	0	2244
8	National M	2062	2204	2390	2272	2511	2244	0
9	Culture St	1334	1387	1357	1217	1419	1587	994
10	War Mem	1964	1962	2128	1785	2095	2212	1102
11	Yongsan F	2721	1879	2463	2498	2852	2969	1821
12	Seoul Plaz	1245	1882	1150	1447	1254	1313	1396
13	Seodaemu	582	2208	670	2512	865	1187	1903
14	Gwanghwa	714	1729	589	1455	798	1230	2058
15	Bosingak E	887	1387	1108	1573	972	1136	1522
16	Heunginjir	904	1507	1232	1939	1034	832	1573
17	Deoksugu	1136	1687	1285	1536	1165	1384	1485
18	Samcheon	1233	2037	1293	2999	1490	992	2321

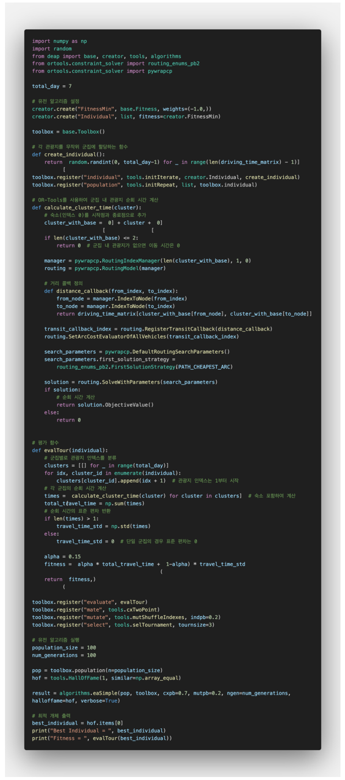
Code

Source Paper



```python
# 필요한 라이브러리 불러오기
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp

# 이동 시간 매트릭스 로드
duration_df = pd.read_csv('./tour_distances_matrix.csv', index_col=0)

# 위치 정보 로드
geo_df = pd.read_csv('./geo_info.csv')
geo_df['Cluster'] = KMeans(n_clusters=total_day, random_state=42).fit(geo_df[['Latitude', 'Longitude']]).labels_

# 숙소 정보
base_location = 'Bukchon Hanok Village'

# 각 클러스터별 TSP 문제 해결 및 출력
for day in range(total_day):
    # 현재 클러스터의 관광지 선택
    cluster_locations =  base_location] + geo_df[geo_df['Cluster'] == day]['Name'].tolist()
    # 이동 시간 매트릭스 추출[
    cluster_matrix = duration_df.loc[cluster_locations, cluster_locations].to_numpy()

    # TSP 문제 설정 및 해결
    manager = pywrapcp.RoutingIndexManager(len(cluster_matrix), 1, 0)
    routing = pywrapcp.RoutingModel(manager)
    transit_callback_index = routing.RegisterTransitCallback(lambda from_index, to_index: cluster_matrix[manager.IndexToNode(
from_index)][manager.IndexToNode(to_index)])
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy = routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC
    solution = routing.SolveWithParameters(search_parameters)
    # 결과 출력
    print(f'Day {day+1} Tour Order(with {solution.ObjectiveValue()}):')
    index = routing.Start(0)
    while not routing.IsEnd(index):
        print(f'{cluster_locations[manager.IndexToNode(index)]} -> ', end='')
        index = solution.Value(routing.NextVar(index))
    print(base_location)
    print()
```

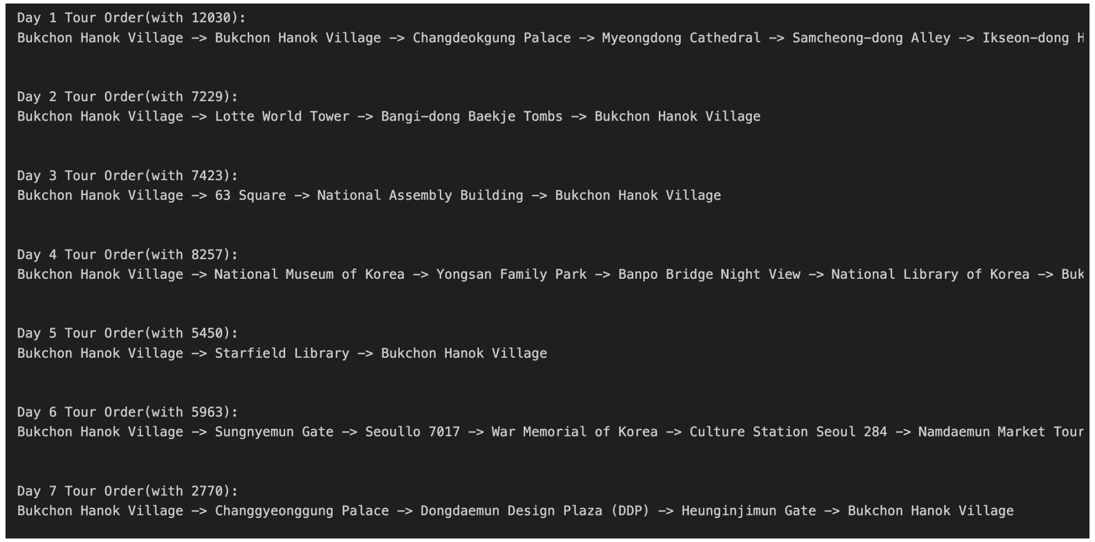
Suggestion



```python
import numpy as np
import random
from deap import base, creator, tools, algorithms
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp

total_day = 7

# 유전 알고리즘 설정
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()

# 각 관광지를 무작위 군집에 할당하는 함수
def create_individual():
    return  random.randint(0, total_day-1) for _ in range(len(driving_time_matrix) - 1)]
         [
toolbox.register("individual", tools.initIterate, creator.Individual, create_individual)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# OR-Tools를 사용하여 군집 내 관광지 순회 시간 계산
def calculate_cluster_time(cluster):
    # 숙소(인덱스 0)를 시작점과 종료점으로 추가
    cluster_with_base =  0] + cluster +  0]
                        [              [
    if len(cluster_with_base) <= 2:
        return 0  # 군집 내 관광지가 없으면 이동 시간은 0

    manager = pywrapcp.RoutingIndexManager(len(cluster_with_base), 1, 0)
    routing = pywrapcp.RoutingModel(manager)

    # 거리 콜백 정의
    def distance_callback(from_index, to_index):
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return driving_time_matrix[cluster_with_base[from_node], cluster_with_base[to_node]]

    transit_callback_index = routing.RegisterTransitCallback(distance_callback)
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy =
        routing_enums_pb2.FirstSolutionStrategy(PATH_CHEAPEST_ARC)

    solution = routing.SolveWithParameters(search_parameters)
    if solution:
        # 순회 시간 계산
        return solution.ObjectiveValue()
    else:
        return 0


# 평가 함수
def evalTour(individual):
    # 군집별로 관광지 인덱스를 분류
    clusters = [[] for _ in range(total_day)]
    for idx, cluster_id in enumerate(individual):
        clusters[cluster_id].append(idx + 1)  # 관광지 인덱스는 1부터 시작
    # 각 군집의 순회 시간 계산
    times =  calculate_cluster_time(cluster) for cluster in clusters]  # 숙소 포함하여 계산
    total_travel_time = np.sum(times)
    # 순회 시간의 표준 편차 반환
    if len(times) > 1:
        travel_time_std = np.std(times)
    else:
        travel_time_std = 0  # 단일 군집의 경우 표준 편차는 0

    alpha = 0.15
    fitness =  alpha * total_travel_time +  1-alpha) * travel_time_std
                                         (
    return  fitness,)
          (

toolbox.register("evaluate", evalTour)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)

# 유전 알고리즘 실행
population_size = 100
num_generations = 100

pop = toolbox.population(n=population_size)
hof = tools.HallOfFame(1, similar=np.array_equal)

result = algorithms.eaSimple(pop, toolbox, cxpb=0.7, mutpb=0.2, ngen=num_generations,
halloffame=hof, verbose=True)

# 최적 개체 출력
best_individual = hof.items[0]
print("Best Individual = ", best_individual)
print("Fitness = ", evalTour(best_individual))
```

Result Comparison

Source Paper (K-Means)

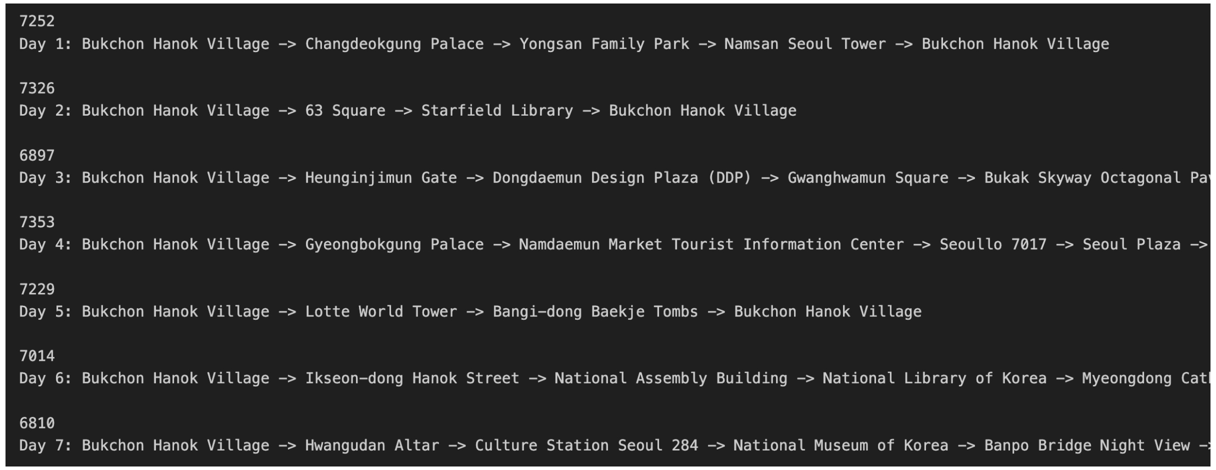


```
Day 1 Tour Order(with 12030):
Bukchon Hanok Village -> Bukchon Hanok Village -> Changdeokgung Palace -> Myeongdong Cathedral -> Samcheong-dong Alley -> Ikseon-dong H


Day 2 Tour Order(with 7229):
Bukchon Hanok Village -> Lotte World Tower -> Bangi-dong Baekje Tombs -> Bukchon Hanok Village


Day 3 Tour Order(with 7423):
Bukchon Hanok Village -> 63 Square -> National Assembly Building -> Bukchon Hanok Village


Day 4 Tour Order(with 8257):
Bukchon Hanok Village -> National Museum of Korea -> Yongsan Family Park -> Banpo Bridge Night View -> National Library of Korea -> Buk


Day 5 Tour Order(with 5450):
Bukchon Hanok Village -> Starfield Library -> Bukchon Hanok Village


Day 6 Tour Order(with 5963):
Bukchon Hanok Village -> Sungnyemun Gate -> Seoullo 7017 -> War Memorial of Korea -> Culture Station Seoul 284 -> Namdaemun Market Tour


Day 7 Tour Order(with 2770):
Bukchon Hanok Village -> Changgyeonggung Palace -> Dongdaemun Design Plaza (DDP) -> Heunginjimun Gate -> Bukchon Hanok Village
```

K-means only considers physical coordinates
(not the real traveling time),
and doesn't consider the size of clusters
It results to high fluctuation between days.
Day 7 has 46 minutes(3 sites)
Day 1 has 5 hours(15 sites) for travel time only
→ Too little time for sight-seeing

Suggestion (GA)

Best Individual = [0, 3, 1, 0, 3, 6, 6, 6, 0, 3, 3, 2, 3, 2, 3, 3, 4, 5, 6, 2, 5, 5, 2, 6, 4, 6, 3, 3, 2, 1, 3, 5]



```
7252
Day 1: Bukchon Hanok Village -> Changdeokgung Palace -> Yongsan Family Park -> Namsan Seoul Tower -> Bukchon Hanok Village

7326
Day 2: Bukchon Hanok Village -> 63 Square -> Starfield Library -> Bukchon Hanok Village

6897
Day 3: Bukchon Hanok Village -> Heunginjimun Gate -> Dongdaemun Design Plaza (DDP) -> Gwanghwamun Square -> Bukak Skyway Octagonal Pa

7353
Day 4: Bukchon Hanok Village -> Gyeongbokgung Palace -> Namdaemun Market Tourist Information Center -> Seoullo 7017 -> Seoul Plaza ->

7229
Day 5: Bukchon Hanok Village -> Lotte World Tower -> Bangi-dong Baekje Tombs -> Bukchon Hanok Village

7014
Day 6: Bukchon Hanok Village -> Ikseon-dong Hanok Street -> National Assembly Building -> National Library of Korea -> Myeongdong Cat

6810
Day 7: Bukchon Hanok Village -> Hwangudan Altar -> Culture Station Seoul 284 -> National Museum of Korea -> Banpo Bridge Night View -
```

Using GA and real public transportation data,
it results in practical travel time
with reasonable cluster sizes
It results to low fluctuation between days.
Day 7 has 1hour 53min(6 sites)
Day 4 has 2hour 2min(10 sites) for travel time
→ Enough time for sight-seeing

Conclusion

- We found out that clustering using GA gave better tour plan than the source paper.
- As we used GA for clustering the sites, as the problem grows, we might have to consider how to tune the hyper parameters.

Future Work

- This methodology only considered transportation time to move between sites. In the future work, we can add sight-seeing time in the consideration.
- Now, we only create clusters to match the total travel days
  (# of clusters = total travel days).
  In the future work, we can make more detailed clusters (can move without using public transportations) and make more specific plan of the travel.

Reference

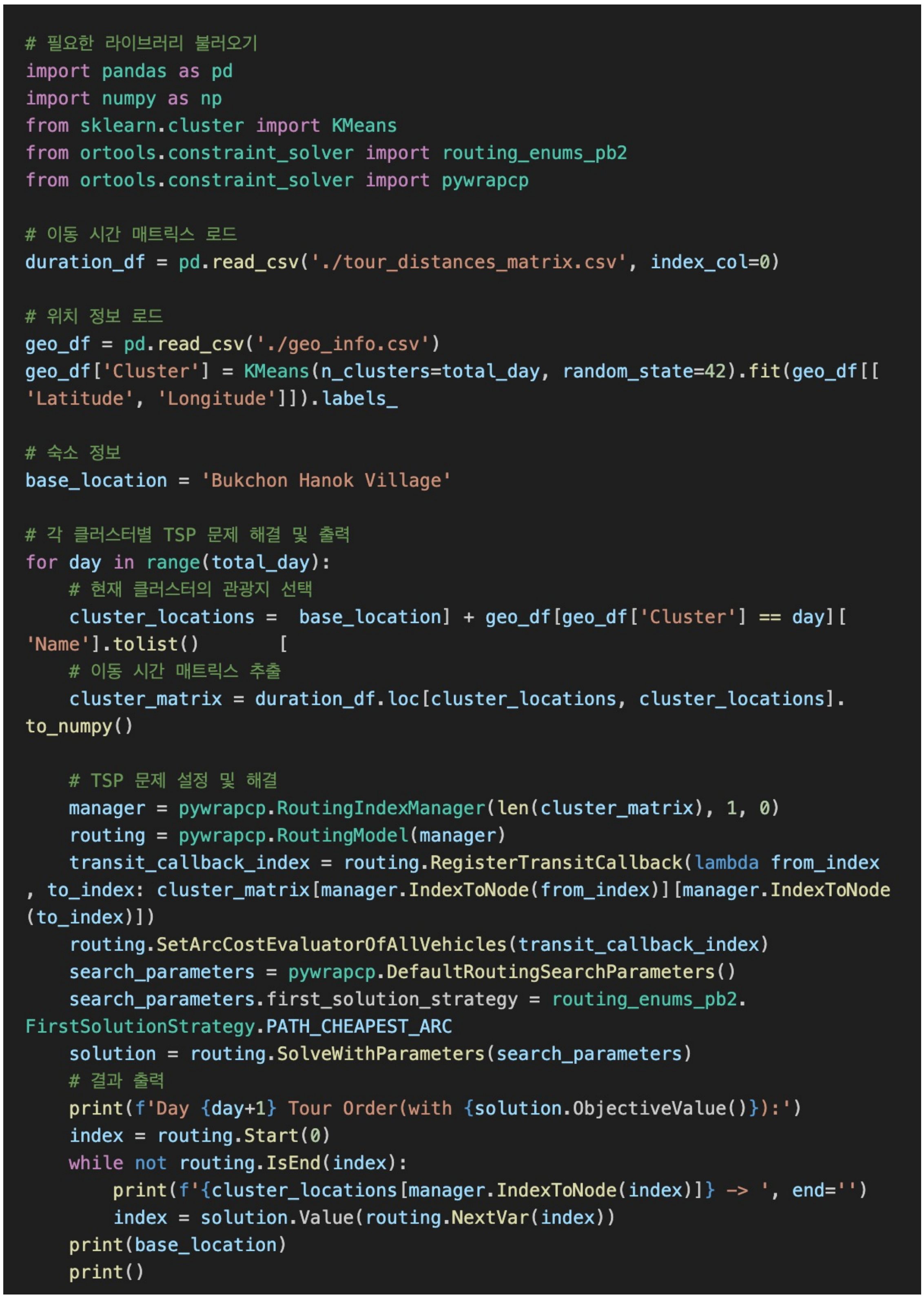
Rani, S., Kholidah, K. N., & Huda, S. N. (2018). A Development of Travel Itinerary Planning Application using Traveling Salesman Problem and K-Means Clustering Approach. In Proceedings of the 2018 7th International Conference on Software and Computer Applications (ICSCA '18) (pp. 327–331). https://doi.org/10.1145/3185089.3185142

https://data.seoul.go.kr/dataList/OA-21050/S/1/datasetView.do

https://www.google.co.kr/maps/?hl=ko

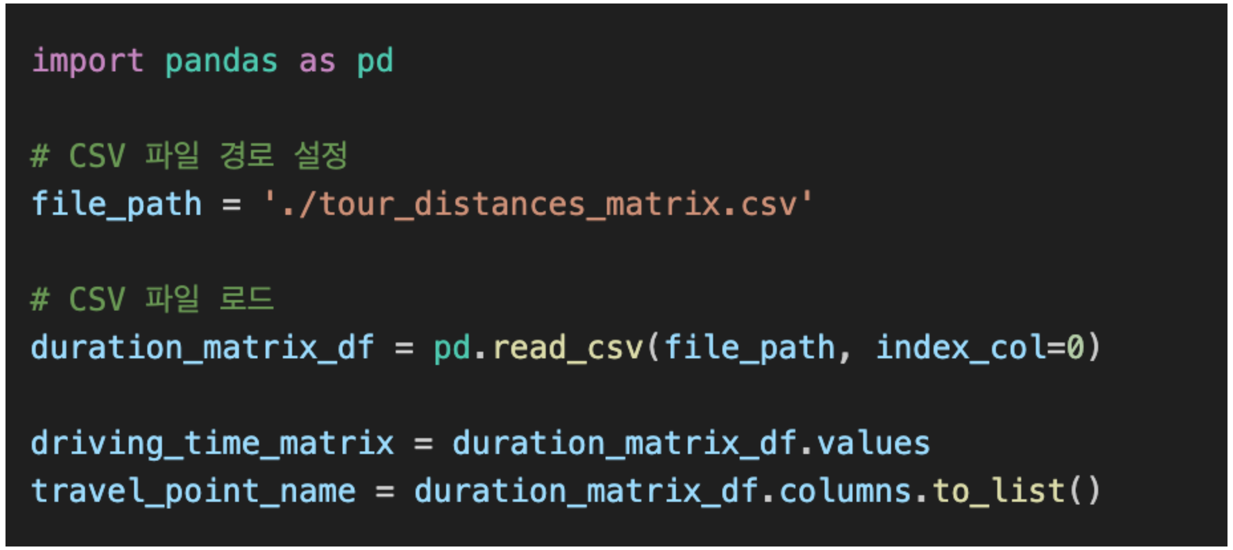
https://developers.google.com/optimization?hl=ko

Appendix 1 – Original paper code



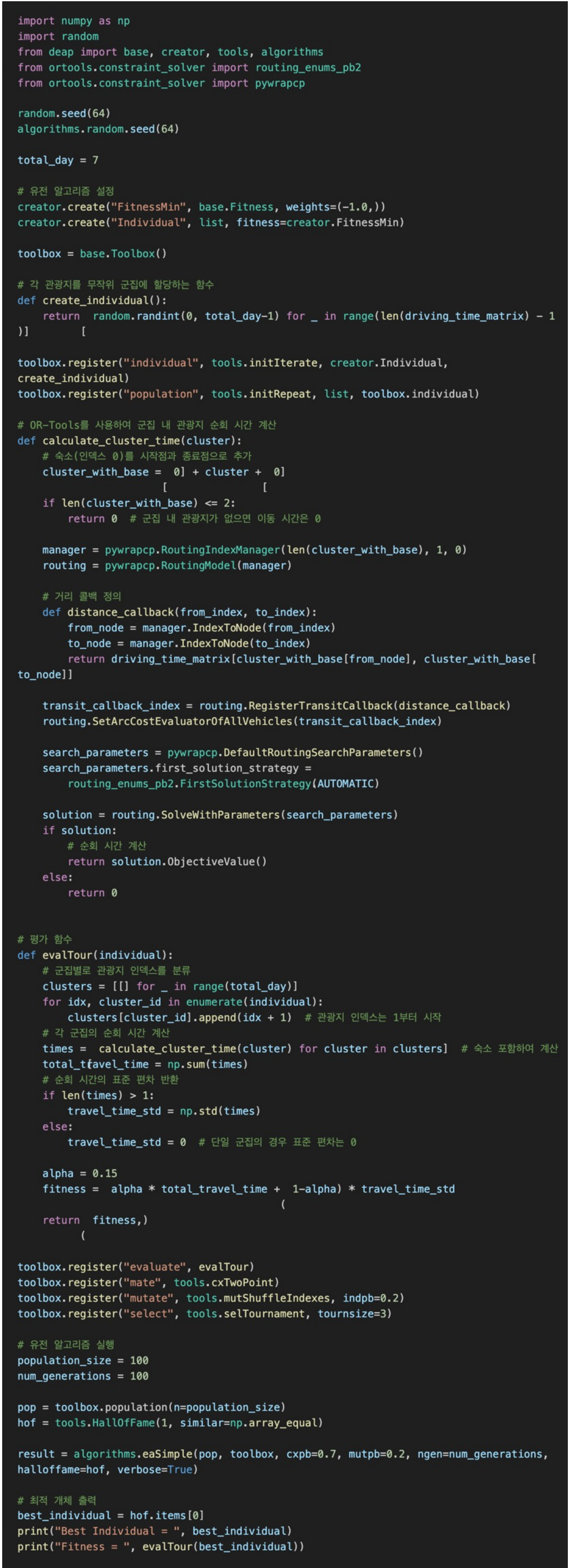
```
# 필요한 라이브러리 불러오기
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp

# 이동 시간 매트릭스 로드
duration_df = pd.read_csv('./tour_distances_matrix.csv', index_col=0)

# 위치 정보 로드
geo_df = pd.read_csv('./geo_info.csv')
geo_df['Cluster'] = KMeans(n_clusters=total_day, random_state=42).fit(geo_df[[
'Latitude', 'Longitude']]).labels_

# 숙소 정보
base_location = 'Bukchon Hanok Village'

# 각 클러스터별 TSP 문제 해결 및 출력
for day in range(total_day):
    # 현재 클러스터의 관광지 선택
    cluster_locations =  base_location] + geo_df[geo_df['Cluster'] == day][
'Name'].tolist()        [
    # 이동 시간 매트릭스 추출
    cluster_matrix = duration_df.loc[cluster_locations, cluster_locations].
to_numpy()

    # TSP 문제 설정 및 해결
    manager = pywrapcp.RoutingIndexManager(len(cluster_matrix), 1, 0)
    routing = pywrapcp.RoutingModel(manager)
    transit_callback_index = routing.RegisterTransitCallback(lambda from_index
, to_index: cluster_matrix[manager.IndexToNode(from_index)][manager.IndexToNode
(to_index)])
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy = routing_enums_pb2.
FirstSolutionStrategy.PATH_CHEAPEST_ARC
    solution = routing.SolveWithParameters(search_parameters)
    # 결과 출력
    print(f'Day {day+1} Tour Order(with {solution.ObjectiveValue()}):')
    index = routing.Start(0)
    while not routing.IsEnd(index):
        print(f'{cluster_locations[manager.IndexToNode(index)]} -> ', end='')
        index = solution.Value(routing.NextVar(index))
    print(base_location)
    print()
```

Appendix 2 – Proposed code



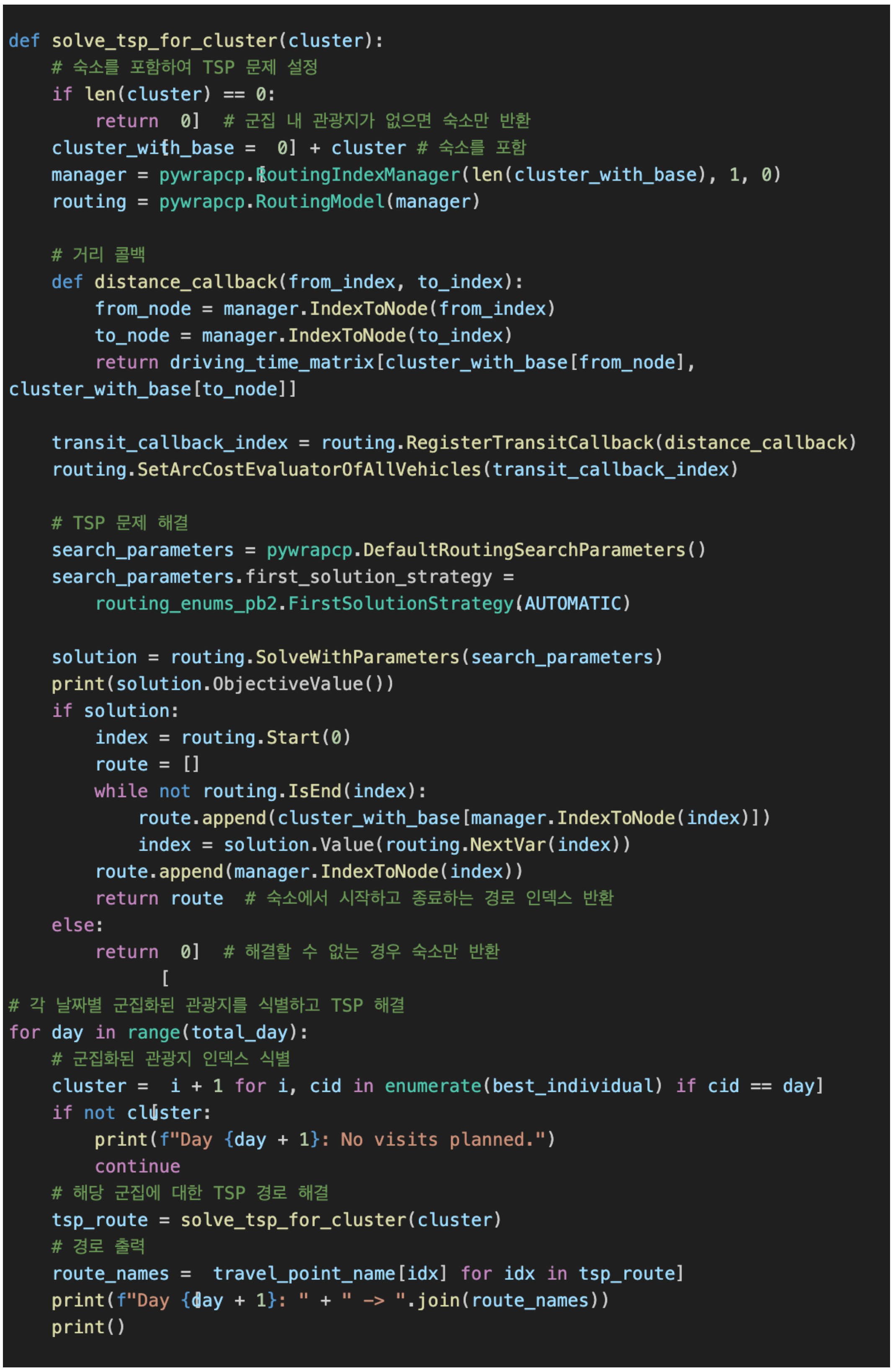
```
import pandas as pd

# CSV 파일 경로 설정
file_path = './tour_distances_matrix.csv'

# CSV 파일 로드
duration_matrix_df = pd.read_csv(file_path, index_col=0)

driving_time_matrix = duration_matrix_df.values
travel_point_name = duration_matrix_df.columns.to_list()
```

Appendix 2 – Proposed code



```
import numpy as np
import random
from deap import base, creator, tools, algorithms
from ortools.constraint_solver import routing_enums_pb2
from ortools.constraint_solver import pywrapcp

random.seed(64)
algorithms.random.seed(64)

total_day = 7

# 유전 알고리즘 설정
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()

# 각 관광지를 무작위 군집에 할당하는 함수
def create_individual():
    return  random.randint(0, total_day-1) for _ in range(len(driving_time_matrix) - 1
)]        [

toolbox.register("individual", tools.initIterate, creator.Individual,
create_individual)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

# OR-Tools를 사용하여 군집 내 관광지 순회 시간 계산
def calculate_cluster_time(cluster):
    # 숙소(인덱스 0)를 시작점과 종료점으로 추가
    cluster_with_base =  0] + cluster +  0]
                        [               [
    if len(cluster_with_base) <= 2:
        return 0  # 군집 내 관광지가 없으면 이동 시간은 0

    manager = pywrapcp.RoutingIndexManager(len(cluster_with_base), 1, 0)
    routing = pywrapcp.RoutingModel(manager)

    # 거리 콜백 정의
    def distance_callback(from_index, to_index):
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return driving_time_matrix[cluster_with_base[from_node], cluster_with_base[
to_node]]

    transit_callback_index = routing.RegisterTransitCallback(distance_callback)
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy =
        routing_enums_pb2.FirstSolutionStrategy(AUTOMATIC)

    solution = routing.SolveWithParameters(search_parameters)
    if solution:
        # 순회 시간 계산
        return solution.ObjectiveValue()
    else:
        return 0


# 평가 함수
def evalTour(individual):
    # 군집별로 관광지 인덱스를 분류
    clusters = [[] for _ in range(total_day)]
    for idx, cluster_id in enumerate(individual):
        clusters[cluster_id].append(idx + 1)  # 관광지 인덱스는 1부터 시작
    # 각 군집의 순회 시간 계산
    times =  calculate_cluster_time(cluster) for cluster in clusters]  # 숙소 포함하여 계산
    total_travel_time = np.sum(times)
    # 순회 시간의 표준 편차 반환
    if len(times) > 1:
        travel_time_std = np.std(times)
    else:
        travel_time_std = 0  # 단일 군집의 경우 표준 편차는 0

    alpha = 0.15
    fitness =  alpha * total_travel_time +  1-alpha) * travel_time_std
                                          (
    return  fitness,)
          (

toolbox.register("evaluate", evalTour)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)

# 유전 알고리즘 실행
population_size = 100
num_generations = 100

pop = toolbox.population(n=population_size)
hof = tools.HallOfFame(1, similar=np.array_equal)

result = algorithms.eaSimple(pop, toolbox, cxpb=0.7, mutpb=0.2, ngen=num_generations,
halloffame=hof, verbose=True)

# 최적 개체 출력
best_individual = hof.items[0]
print("Best Individual = ", best_individual)
print("Fitness = ", evalTour(best_individual))
```

Appendix 2 – Proposed code



```python
def solve_tsp_for_cluster(cluster):
    # 숙소를 포함하여 TSP 문제 설정
    if len(cluster) == 0:
        return  0]  # 군집 내 관광지가 없으면 숙소만 반환
    cluster_with_base =  0] + cluster # 숙소를 포함
    manager = pywrapcp.RoutingIndexManager(len(cluster_with_base), 1, 0)
    routing = pywrapcp.RoutingModel(manager)

    # 거리 콜백
    def distance_callback(from_index, to_index):
        from_node = manager.IndexToNode(from_index)
        to_node = manager.IndexToNode(to_index)
        return driving_time_matrix[cluster_with_base[from_node],
cluster_with_base[to_node]]

    transit_callback_index = routing.RegisterTransitCallback(distance_callback)
    routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

    # TSP 문제 해결
    search_parameters = pywrapcp.DefaultRoutingSearchParameters()
    search_parameters.first_solution_strategy =
        routing_enums_pb2.FirstSolutionStrategy(AUTOMATIC)

    solution = routing.SolveWithParameters(search_parameters)
    print(solution.ObjectiveValue())
    if solution:
        index = routing.Start(0)
        route = []
        while not routing.IsEnd(index):
            route.append(cluster_with_base[manager.IndexToNode(index)])
            index = solution.Value(routing.NextVar(index))
        route.append(manager.IndexToNode(index))
        return route  # 숙소에서 시작하고 종료하는 경로 인덱스 반환
    else:
        return  0]  # 해결할 수 없는 경우 숙소만 반환
              [
# 각 날짜별 군집화된 관광지를 식별하고 TSP 해결
for day in range(total_day):
    # 군집화된 관광지 인덱스 식별
    cluster =  i + 1 for i, cid in enumerate(best_individual) if cid == day]
    if not cluster:
        print(f"Day {day + 1}: No visits planned.")
        continue
    # 해당 군집에 대한 TSP 경로 해결
    tsp_route = solve_tsp_for_cluster(cluster)
    # 경로 출력
    route_names =  travel_point_name[idx] for idx in tsp_route]
    print(f"Day {day + 1}: " + " -> ".join(route_names))
    print()
```