Daegu Gyeongbuk
Institute of Science & Technology

Traveling Salesman Problem

2021.12.13

202123003 **Namki Kim**
M.S. Student, Robotics Engineering
ttorangs3@dgist.ac.kr / 010-9960-2926

202123008 **Jinmin Kim**
M.S. Student, Robotics Engineering
rlawlsals@dgist.ac.kr / 010-6266-6099

Contents

Introduction

< World Trip >

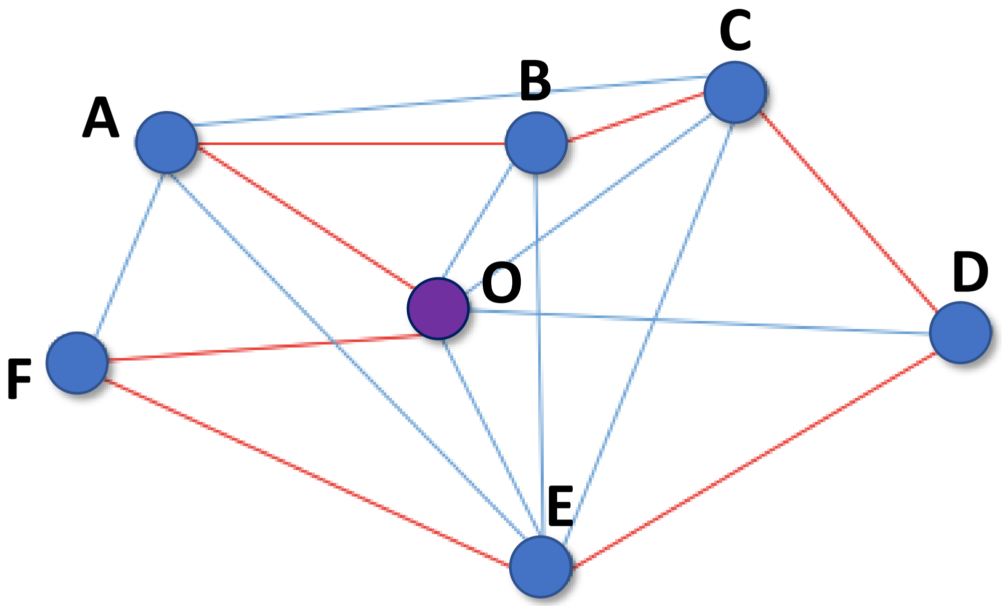
- We want to travel around the world.
- There are many paths we travel.
- Our objective is to travel to all countries once, and find the least expensive travel paths.

ROBOTICS ENGINEERING

Introduction

World Trip = Traveling Salesman Problem(TSP)

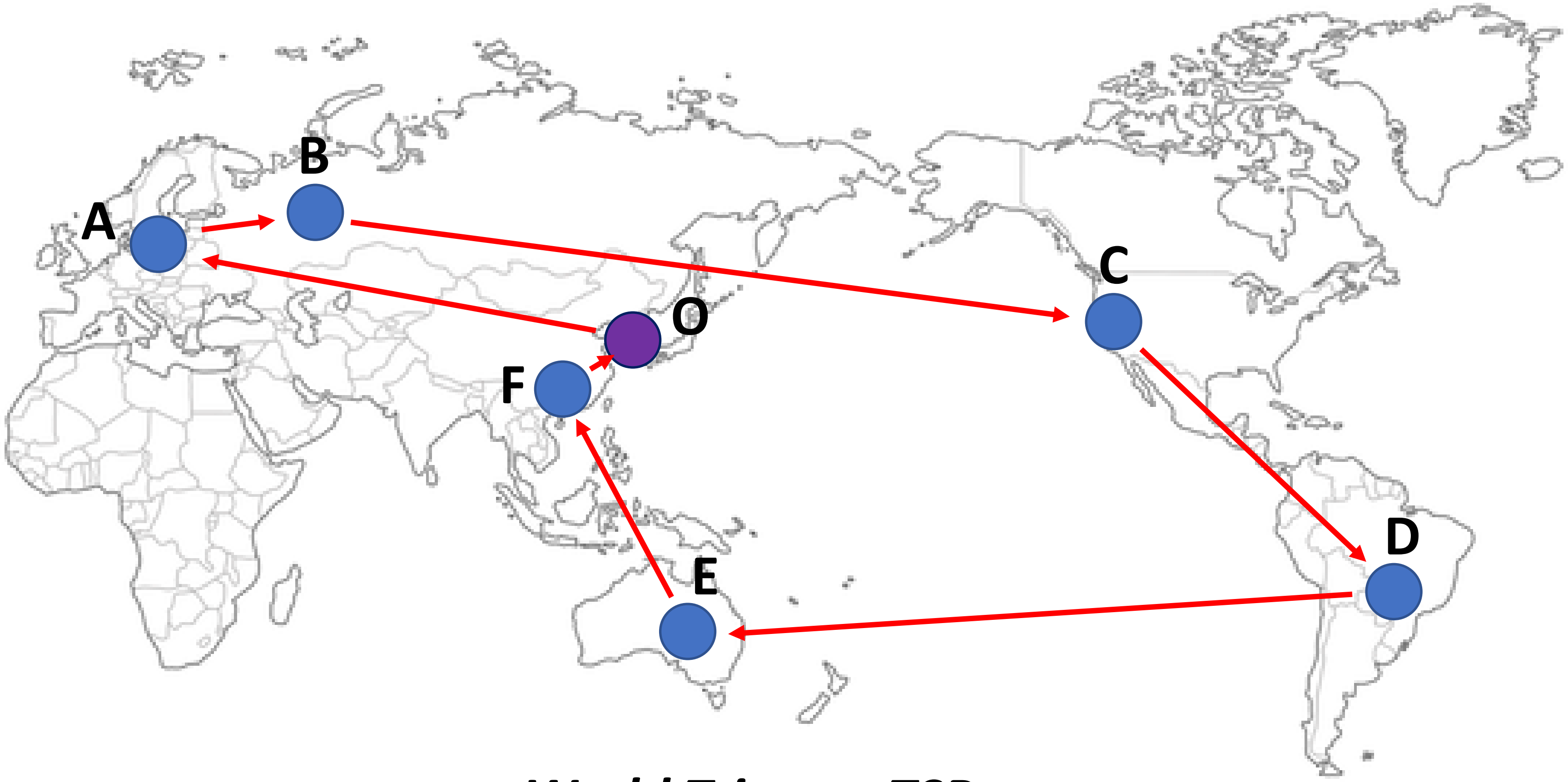
World Trip



TSP

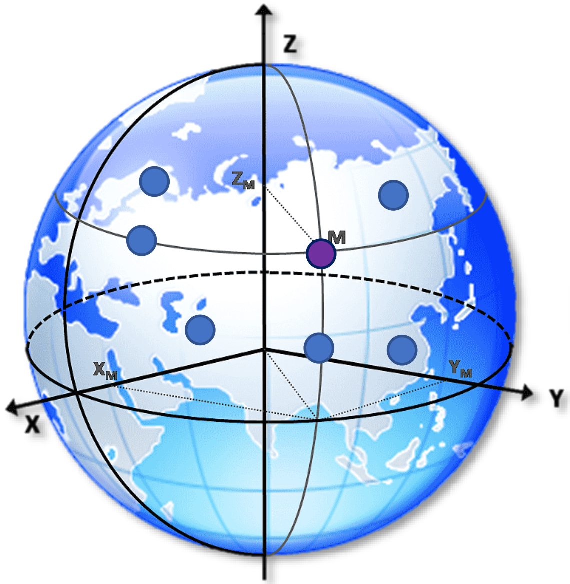
Introduction

World Trip = Traveling Salesman Problem(TSP)

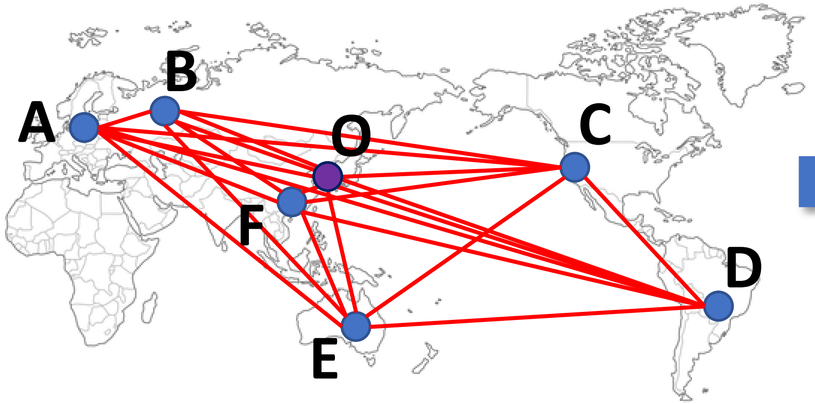


World Trip => TSP

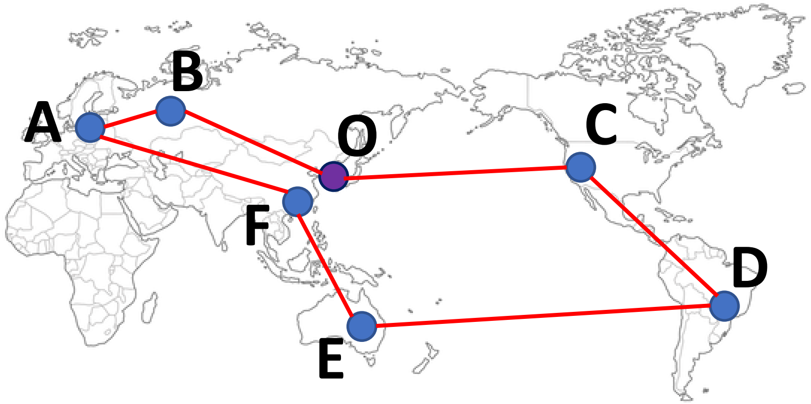
System model



Obtain the N cities
with geographic coordinate
(latitude, longitude)



Calculate the distances between each city
to make <u>distance matrix</u>



Find path to minimize
sum of distances

Mathematical Problem

Objective Function : $\min \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{i,j} x_{i,j}$

n : # of cities
$c_{i,j}$: distance from i to j
$x_{i,j}$: weight term from i to j

Ex) n = 7

$$c_{i,j} = \begin{bmatrix} c_{0,0} & c_{0,1} & c_{0,2} & c_{0,3} & c_{0,4} & c_{0,5} & c_{0,6} \\ c_{1,0} & c_{1,1} & c_{1,2} & c_{1,3} & c_{1,4} & c_{1,5} & c_{1,6} \\ c_{2,0} & c_{2,1} & c_{2,2} & c_{2,3} & c_{2,4} & c_{2,5} & c_{2,6} \\ c_{3,0} & c_{3,1} & c_{3,2} & c_{3,3} & c_{3,4} & c_{3,5} & c_{3,6} \\ c_{4,0} & c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4} & c_{4,5} & c_{4,6} \\ c_{5,0} & c_{5,1} & c_{5,2} & c_{5,3} & c_{5,4} & c_{5,5} & c_{5,6} \\ c_{6,0} & c_{6,1} & c_{6,2} & c_{6,3} & c_{6,4} & c_{6,5} & c_{6,6} \end{bmatrix} * x_{i,j} = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} & x_{0,5} & x_{0,6} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} & x_{1,6} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} & x_{2,6} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} & x_{3,6} \\ x_{4,0} & x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} & x_{4,5} & x_{4,6} \\ x_{5,0} & x_{5,1} & x_{5,2} & x_{5,3} & x_{5,4} & x_{5,5} & x_{5,6} \\ x_{6,0} & x_{6,1} & x_{6,2} & x_{6,3} & x_{6,4} & x_{6,5} & x_{6,6} \end{bmatrix} = \begin{cases} 1 \\ 0 \end{cases}$$

0 Known

Unknown 0

$\Rightarrow$ We have to find optimal "x_{ij}" to minimize $\sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{i,j} x_{i,j}$

Mathematical Problem

Constraints :

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n;$$

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n;$$

① : Each row has only one path, except for the path to itself.

+ *Miller-Tucker-Zemlin(MTZ) formula*

$$u_i - u_j + 1 \leq (n-1)(1 - x_{i,j}) \qquad 2 \leq i \neq j \leq n;$$

$$0 \leq u_i \leq n \qquad 2 \leq i \leq n;$$

② : Except for the point selected before.

$$x_{i,j} = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} & x_{0,5} & x_{0,6} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} & x_{1,6} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} & x_{2,6} \\ x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} & x_{3,6} \\ x_{4,0} & x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} & x_{4,5} & x_{4,6} \\ x_{5,0} & x_{5,1} & x_{5,2} & x_{5,3} & x_{5,4} & x_{5,5} & x_{5,6} \\ x_{6,0} & x_{6,1} & x_{6,2} & x_{6,3} & x_{6,4} & x_{6,5} & x_{6,6} \end{bmatrix} \quad 0$$

$$u_i = \begin{pmatrix} u_0 = 1 \\ 2 \leq u_1 \leq n \\ 2 \leq u_2 \leq n \\ 2 \leq u_3 \leq n \\ 2 \leq u_4 \leq n \\ 2 \leq u_5 \leq n \\ 2 \leq u_6 \leq n \end{pmatrix}$$

$$if\ x_{i,j} == 1$$

$$u_i + 1 \leq u_j$$

$\Rightarrow$ Make the ranking of j greater than the ranking of i.

Mathematical Problem

Objective Function :

$$\min \sum_{i=1}^{n} \sum_{j\neq i, j=1}^{n} c_{i,j} x_{i,j}$$

n : # of cities
$c_{i,j}$: distance from i to j
$x_{i,j}$: weight term from i to j

Constraints :

$$\sum_{i=1, i\neq j}^{n} x_{ij} = 1 \qquad j = 1, \ldots, n;$$

$$\sum_{j=1, j\neq i}^{n} x_{ij} = 1 \qquad i = 1, \ldots, n;$$

+ Miller-Tucker-Zemlin(MTZ) formula

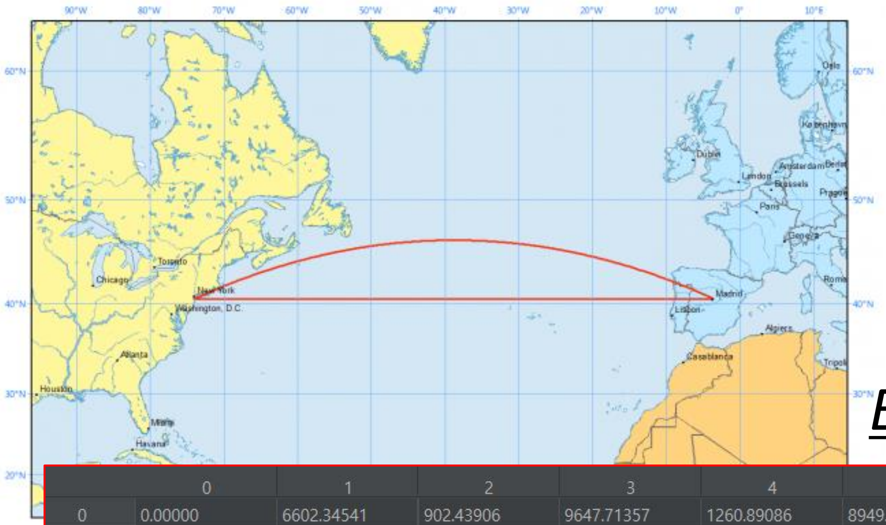
$$u_i - u_j + 1 \le (n-1)(1 - x_{i,j}) \qquad 2 \le i \ne j \le n;$$
$$0 \le u_i \le n \qquad 2 \le i \le n;$$

u_i : ranking (variable 2~n)

Implementation of the Problem using CVXPY

- **Data setting**

```
##################################################
# Original Data
##################################################
points = [(37.460352589222516, 126.44069569793629), #인천 국제 공항
          (55.98657825317026, 37.409135729903284),#러시아 세레메티예보 국제공항
          (40.07992295253526, 116.60325157289097), #베이징 서우두 국제 공항
          (33.94179359694278, -118.40861583285489),#LA 공항

          (35.77202149951756, 140.39280718254184), #도쿄 나리타 공항
          (49.00984277619373, 2.5479659983148215),#파리 샤를 드 골 공항
          (-33.97138290223474, 18.601945622962848),#남아공 케이프타운 국제공항
          # (-22.805106742790127, -43.256296708479944),#브라질 리우데자네이루 갈레앙 국제공항
          # (51.47015594442427, -0.4539307211496268),#영국 런던 히스로 공항
          # (45.31942321207133, -75.66880042136992),#캐나다 오타와 국제 공항
          # (38.95337482429059, -77.45622766575748),#워싱턴 덜레스 국제공항
          # (-33.93515126261149, 151.16684782853076),#호주 시드니 국제 공항
          # (52.36405019165199, 13.509147929132032), #독일 브란덴 브루크 국제공항
          ]
```



Ex) n = 7

	0	1	2	3	4	5	6
0	0.00000	6602.34541	902.43906	9647.71357	1260.89086	8949.88345	13646.62096
1	6602.34541	0.00000	5811.45888	9779.59776	7525.07709	2460.99949	10126.45509
2	902.43906	5811.45888	0.00000	10058.23236	2138.60247	8211.86870	12957.20306
3	9647.71357	9779.59776	10058.23236	0.00000	8771.79261	9124.31330	16082.61108
4	1260.89086	7525.07709	2138.60247	8771.79261	0.00000	9733.46474	14779.12180
5	8949.88345	2460.99949	8211.86870	9124.31330	9733.46474	0.00000	9328.18063
6	13646.62096	10126.45509	12957.20306	16082.61108	14779.12180	9328.18063	0.00000

1. Obtain the N cities **with geographic coordinate** **(latitude, longitude)**

2. Using the library for geographical calculations, the **distance matrix (** $c_{i,j}$ **)** can be obtained.

Implementation of the Problem using CVXPY

- **Define & solve the problem**

```python
# Defining variables
X = cp.Variable(C.shape, boolean=True)
# print(X)
u = cp.Variable(n, integer=True)
ones = np.ones((n,1))

# Defining the objective function
objective = cp.Minimize(cp.sum(cp.multiply(C, X)))
# var 객체를 이용하여 objective function을 정의(현재 상태에서 계산이 행해지지는 않았음.)

# Defining the constraints
constraints = []
constraints += [X @ ones == ones]
constraints += [X.T @ ones == ones]
constraints += [cp.diag(X) == 0]
constraints += [u[1:] >= 2]
constraints += [u[1:] <= n]
constraints += [u[0] == 1]

for i in range(1, n):
    for j in range(1, n):
        if i != j:
            constraints += [ u[i] - u[j] + 1  <= (n - 1) * (1 - X[i, j]) ]
```

3. Define the **objective function** and **constraints**

```python
# Solving the problem
prob = cp.Problem(objective, constraints)

prob.solve(verbose=True)

for variable in prob.variables():
    print("Variable %s: value %s" % (variable.name(), variable.value))

# Transforming the solution to a path
X_sol = np.argwhere(X.value==1)
print(X_sol)
orden_x = X_sol[0]
orden = X_sol[0].tolist()

for i in range(1, n):
    row = orden[-1]
    orden.append(X_sol[row,1])

# Showing the optimal path
print('The path is:\n')
print( ' => '.join(map(str, orden)))

# Showing the optimal distance
distance = np.sum(np.multiply(C, X.value))
print('The optimal distance is:', np.round(distance,2), 'km')
```

4. Solve the problem **using CVXPY**

Implementation of the Problem using CVXPY

- **Result**

<Console window>

<u>Ex) n = 7</u>

```
[[0 2]
 [1 5]
 [2 1]
 [3 4]
 [4 0]
 [5 6]
 [6 3]]
The path is:

0 => 2 => 1 => 5 => 6 => 3 => 4 => 0
The optimal distance is: 44618.37 km
```

<Debug window>

$\{\boldsymbol{x_{i,j}}\} =$

	0	1	2	3	4	5	6
0	0.00000	-0.00000	1.00000	-0.00000	-0.00000	-0.00000	-0.00000
1	-0.00000	0.00000	-0.00000	-0.00000	-0.00000	1.00000	-0.00000
2	-0.00000	1.00000	0.00000	-0.00000	-0.00000	-0.00000	-0.00000
3	-0.00000	-0.00000	-0.00000	0.00000	1.00000	-0.00000	-0.00000
4	1.00000	-0.00000	-0.00000	-0.00000	0.00000	-0.00000	-0.00000
5	-0.00000	-0.00000	-0.00000	-0.00000	-0.00000	0.00000	1.00000
6	-0.00000	-0.00000	-0.00000	1.00000	-0.00000	-0.00000	0.00000

$x_{i,j}$: *weight term from i to j*

$\{\boldsymbol{u_i}\} =$

	0	1	2	3	4	5	6
0	1.00000	3.00000	2.00000	6.00000	7.00000	4.00000	5.00000

u_i : *ranking (variable 2~n)*

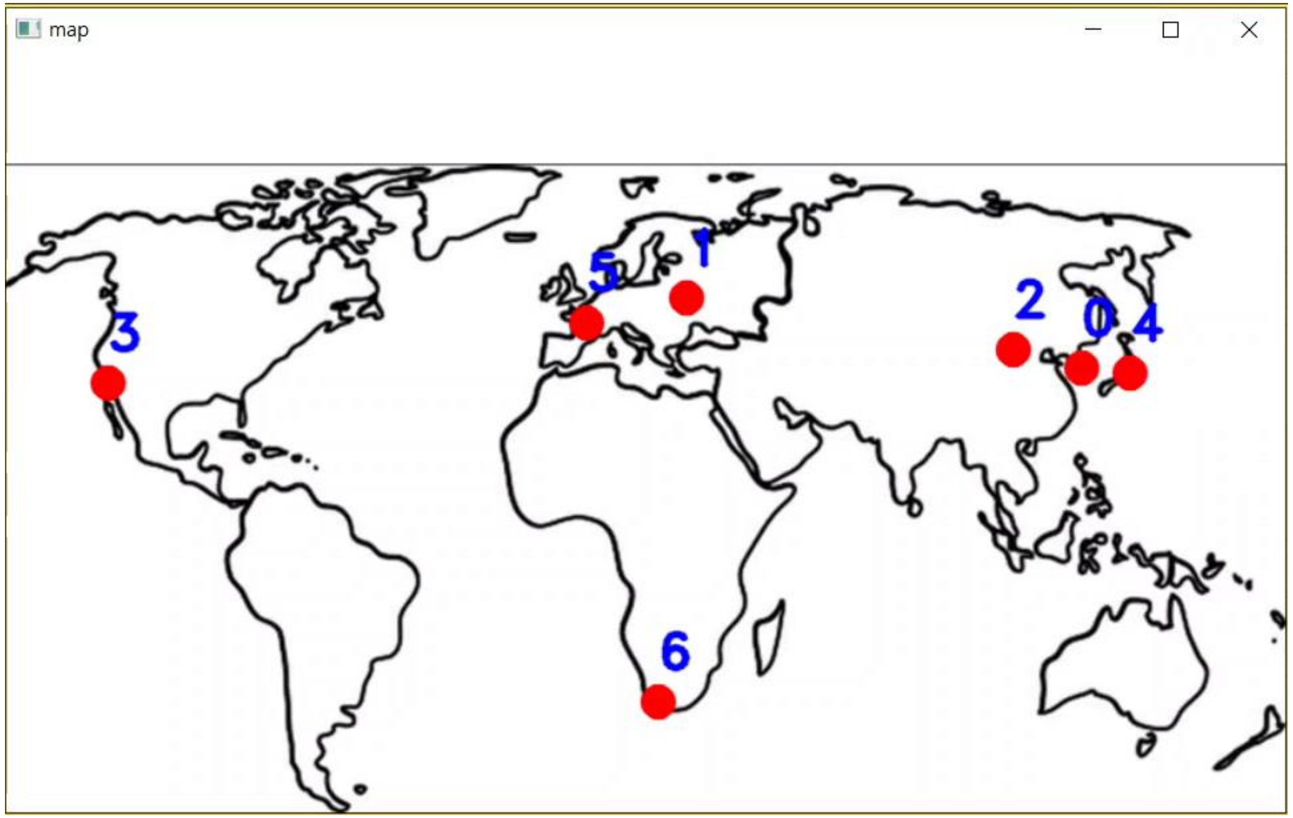
<u>The optimal path</u> and **its <u>distance</u>** are printed

$\{\boldsymbol{x_{i,j}}\}$ & $\{\boldsymbol{u_i}\}$ can be checked in **<u>debug mode</u>**

Validation to Ground-truth

Ex) n = 7

$\Rightarrow$ Number of paths = (7-1)! = 720



Detect **the minimum distance** among **all the paths**

Ground-truth Result

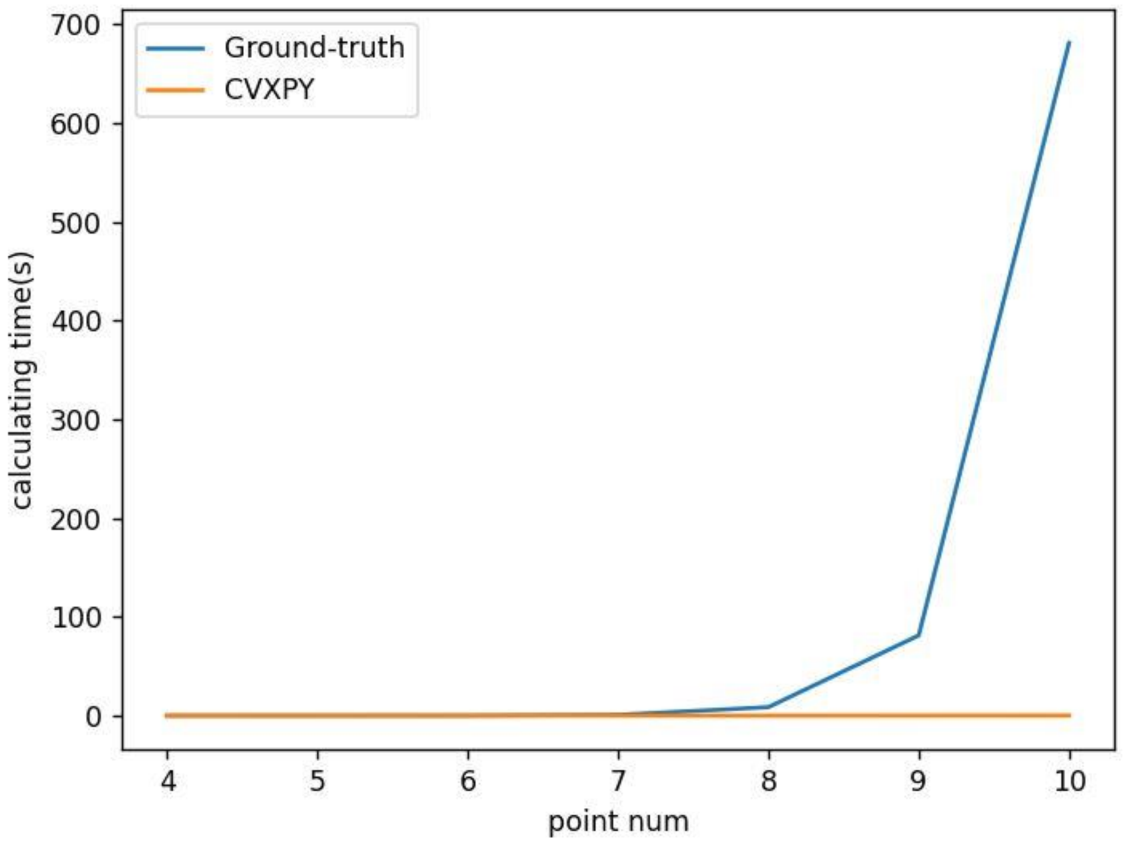
```
계산갯수 : 720

The optimal distance is: 44618.37 km

The path is:

0 => 2 => 1 => 5 => 6 => 3 => 4 => 0
```

$\Rightarrow$ *The result is same as CVXPY result*

Computing Time Comparison

- **Computing time (sec)**

N (# of cities)	Ground-truth	CVXPY
4	0.0010	0.0289
5	0.0113	0.0329
6	0.0452	0.0438
7	0.7634	0.0768
8	8.5877	0.0847
9	81.2387	0.1177
10	680.9935	0.1705



$\Rightarrow$ **If N becomes extremely large, CVXPY can be expected to solve the problem quickly.**

Conclusion

- Traveling Salesman Problem (TSP) was proposed.

- The path with the minimum cost was found by CVXPY module.

- For validation, the ground-truth was calculated by python, and was same as result of CVXPY.

- This problem is applicable to many fields of the path combination.

  ex)
  - Find the best route for a robot
  - Search navigation routes
  - Delivery service

Thank you