Solution for

Travelling Salesmen Problem

**Algorithm Used as Foundation:** Heuristic Algorithm (Nearest Neighbour)

**Heuristic Algorithm:**

A heuristic algorithm used to quickly solve this problem is the nearest neighbour (NN) algorithm (also known as the Greedy Algorithm). Starting from a randomly chosen city, the algorithm finds the closest city. The remaining cities are analysed again, and the closest city is found.

These are the **steps** of the NN algorithm:

1. Start at a random vertex
2. Determine the shortest distance connecting the current vertex and an unvisited vertex V
3. Make the current vertex the unvisited vertex V
4. Make V visited
5. Record the distance travelled
6. Terminate if no other unvisited vertices remain
7. Repeat step 2

**Code in Python with Explanation:** (File attached as TSP\_solution.py)

**# importing libraries**

import random

import numpy as np

import matplotlib.pyplot as plt

**# input for total cities**

total\_cities = int(input("Enter total number of cities: "))

Let’s say total\_cities = 5

**####### Creating Cost/Distance Matrix #######**

**# For simplification Generating coordinates of cities randomly**

cities\_cord = [random.sample(range(100), 2) for x in range(total\_cities)]

print("City Coordinates:",cities\_cord)

Cities\_cord = [[50, 8], [3, 98], [29, 15], [18, 66], [36, 82]]

cost\_matrix = list()

**# Calculating distances btw each city**

for city1 in cities\_cord:

row = map(lambda city2: round(((city1[0] - city2[0])\*\*2 + (city1[1] - city2[1])\*\*2) \*\* 0.5, 2) , cities\_cord)

cost\_matrix.append(list(row))

**# converting distance list into array**

cost\_matrix = np.array(cost\_matrix)

**# Now filling diagonal values as infinity**

np.fill\_diagonal(cost\_matrix, float('inf'))

print("Cost Matrix:\n",cost\_matrix)

Cost\_Matrix =

[ inf 101.53 22.14 66.24 75.31 ]

[101.53 inf 86.98 35.34 36.67 ]

[ 22.14 86.98 inf 52.17 67.36 ]

[ 66.24 35.34 52.17 inf 24.08 ]

[ 75.31 36.67 67.36 24.08 inf ]

**####### Solution for TSP #######**

def **TSP** (cost\_matrix, start=None):

""" Solution by more simplifying and optimizing

to going to nearest unvisited city (Heuristic Algorithm) """

**#Making start point as First City**

if start is None:

start = 0 #first city index is 0 in array

dist = 0.0

must\_visit = list(range(total\_cities)) **#to maintain list of unvisited cities**

path = [start+1] **# Adding city 1 in path as starting point**

while len(must\_visit)>1:

**# Gives nearest unvisited city distance**

dist += np.amin(cost\_matrix[start, 1:])

start = np.argmin(cost\_matrix[start, 1:])+1 **# Gives City Number**

path.append(start+1) **# Adding visited city in path**

**#Making visited city column's values as infinity** **so that no longer taking part in minnimum**

cost\_matrix[:, [start]] = float('inf')

**# Removing visited city from unvisited list**

must\_visit.remove(start)

dist += cost\_matrix[start,0] **# Adding distance of last city from start city**

path.append(1) **# Adding start city in path to complete cycle**

return dist, path

shortest\_dist, best\_path = TSP(cost\_matrix)

print("Optimized Distance/Cost:",shortest\_dist)

print("Best Route:",best\_path)

**Explanation:**

**Iteration First:**

First City -> start = 0

Hence -> Path = [1]

Unvisited = [1,2,3,4,5]

Then minimum distanced from city 1 = 22.14 and that city is 3

Hence -> distance = 0+22.14

Path = [1,3]

Unvisited = [1,2,4,5]

Then Making whole column of city 3 as “inf”

Then minimum distanced from city 3 (Excepting start city) = 52.17 and that city is 4

Hence -> distance = 22.14 + 52.17

Path = [1,3,4]

Unvisited = [1,2,5]

Repeating Above step until visited = [1]

-> distance = 22.14 + 52.17 + 24.08 + 36.37 = 135.02

-> path = [1, 3, 4, 5, 2]

Now we are at last city, Then we have to go starting city

Hence -> distance = 135.02 + 101.53 = 207.91 (Shortest Distance)

Hence -> Path = [1, 3, 4, 5, 2, 1]