INSTITUT NATIONAL DE STATISTIQUE ET D'ÉCONOMIE APPLIQUÉE

Novembre 2022

Rapport du projet

Problème du voyageur de commerce TSP

Présenté à

Mr.Benmansour

Présenté par

Ezzagrani Habiba
El Ghazi Siham

SOMMAIRE

RÉSUMÉ

L'objectif principal de ce travail est de trouver la solution optimale pour le problème de voyageur de commerce . Pour mener à bien ce projet, on a d'abord répertorié et analysé les besoins afin d'obtenir une expression précise des besoins. Cette analyse a permis de développer plus efficacement les différentes fonctionnalités. Ensuite, on a effectué des tests afin de comparer les exigences attendues avec les résultats obtenus et d'améliorer ces derniers. Et enfin, on a fait le point sur notre projet

ABSTRACT

The main objective of this work is to find the optimal solution for the traveling salesman problem.
To carry out this project, we first listed and analyzed the needs in order to obtain a precise expression of the needs. This analysis made it possible to develop the various functionalities more efficiently. Then, tests were carried out in order to compare the expected requirements with the results obtained and to improve the latter. And finally, we took stock of our project

TABLE DES FIGURES

INTRODUCTION GÉNÉRALE

Dans le cadre de ce module, on a été amener à effectuer un projet . Le but de ce dernier est de mettre en pratique les connaissances théoriques assimilées à l'ensemble des solutions vu en cours. Ce rapport sera organisé comme suit : dans la première sera consacrée à la présentation du problème . Ensuite, on va présenter une étude préalable du projet en analysant le contexte ainsi que les objectifs. on présentera aussi une description de l'existant. Dans la troisième partie, une étude technique sera présentée en décrivant l'environnement de développement matériel et logiciel et on présente les différentes fonctionnalités du programme à travers des captures d'écran.

CHAPITRE I

1. Contexte générale :

- Problématique :

Le problème du voyageur de commerce peut être modélisé à l'aide d'un graphe valué et complète constitué d'un ensemble de sommets et d'un ensemble d'arêtes lié entre eux . Chaque sommet représente une ville, une arête symbolise le passage d'une ville à une autre, et on lui associe une valeur pouvant représenter une distance, un temps de parcours ou encore un coût.

- Objectif du projet :

Comme déjà mentionner , l'objectif principal est de trouver une solution optimale à ce problème . Cela est faite en utilisant deux méthodes et comparer le temps de réponse des deux programmes .

- Bref descriptif sur le fonctionnement du projet :

Le projet contient deux programmes , chacun de ces derniers est développé par des méthodes de réalisation différentes . Les deux sont développés en Python . Pour tester les programmes , on va se servir d'un fichier Excel qui contient des matrices de différentes tailles , ces matrices représentent la distance entre les différentes villes que le boyageur souhaite visiter .

CHAPITRE II

1. Méthodes de résolution :

- ### Présentation des méthodes utilisées :

On a été chargé de développer cette solution en utilisant tout d'abord une méthode exacte **la programmation dynamique** et une autre méthode approchée de notre choix . On a choisi l'**heuristique** .

- ### La programmation dynamique :

La programmation dynamique est une technique générale de résolution exacte de problèmes d'optimisation qui consiste à énumérer les solutions du problème.

- ### L'heuristique :

Un raisonnement heuristique est une méthode de résolution de problème qui ne s'appuie pas sur une analyse détaillée ou exhaustive du problème.

3. La solution trouvée :

- ### En utilisant la méthode exacte :

L'intérêt de la résolution du TSP par la programmation dynamique réside d'une part dans la rapidité des calculs et d'autres part dans la possibilité d'intégrer des contraintes supplémentaires .

On a essayé d'implémenter la fonction mais sous forme d'un programme et non pas sous forme d'une solution mathématique .
=>implémentation mathématique :

- $f(S, x) = \min_{y \in S} (f (S - \{y\}, y) + d(y, x))$

=>implémentation du code Python :

1.On a remarqué que si l'utilisateur veut débuter avec une ville qui se situe à l'intérieur ou à la fin de la matrice il faut la permuter et rendre la ville choisi par l'utilisateur la première ville , donc on a créer une fonction de permutation qui prend en paramètre la ligne et la colonne de notre matrice .

```
Entrée [2]:
#Permuter matrice
def PermuterMatrice(mat, l, c):
    for i in range(len(mat)):
        tmp2 = mat[i][0]
        mat[i][0] = mat[i][l]
        mat[i][l] = tmp2
    for j in range(len(mat[0])):
        tmp = mat[0][j]
        mat[0][j] = mat[c][j]
        mat[c][j] = tmp
    return mat
```

```
Entrée [74]:
PermuterMatrice(matrix, 0, 2)

Out[74]:
[[0, 1, 6, 4], [7, 15, 0, 8], [2, 0, 9, 10], [3, 6, 12, 0]]
```

Figure 1 : fonction de permutation

2. On a crée ensuite une fonction qui retourne une liste qui contient la ville de départ et la ville qui a la distance minimale ,calculée par cette fonction, entre la ville de départ et les autres villes qui appartient à l'ensemble des villes de notre matrice.

```
def get_minimumm(k, a, mat):
    if (k, a) in g:

        return g[k, a]

    values = []
    all_min = []
    for j in a:
        set_a = copy.deepcopy(list(a))
        set_a.remove(j)
        all_min.append([j, tuple(set_a)])
        result = get_minimumm(j, tuple(set_a), mat)
        values.append(mat[k-1][j-1] + result)


    g[k, a] = min(values)
    p.append(((k, a), all_min[values.index(g[k, a])]))
   # p.reverse()
    return g[k, a]
```

Figure 2 : fonction GetMinimum()

3.Puis la fonction principale qui prend en paramètre la matrice souhaité , qui demande à l'utilisateur d'entre la ville de départ , si la ville de départ se positionne déjà au début de la matrice , le programme exécute l'ensemble des instructions sans aucun problème , sinon , il fait appel à la fonction "permuterMatrice" , après la saisie de la ville de départ cette dernière sera exclus de l'ensemble des villes puis on fait un appel à la fonction qui calcule li minimum entre la ville choisi et le reste de l'ensemble .

```
import sys
import copy
n = len(data)
all_sets = []
g = {}
p = []

def main(mat):
    ville_départ = input("ville départ : ")
    vdd = ville_départ
    for cle,value in data.items():
        if ville_départ == value:
            vd = cle
            for i in range(len(mat)):
                tmp2 = mat[i][0]
                mat[i][0] = mat[i][vd-1]
                mat[i][vd-1] = tmp2
            for j in range(len(mat[i])):
                tmp = mat[0][j]
                mat[0][j] = mat[vd-1][j]
                mat[vd-1][j] = tmp
    data[vd] = data[1]
    tmm = data[1]
    data[1] = ville_départ
    ville_départ = tmm
   # print(data[1])
    for x in range(1, n):
        g[x + 1, ()] = mat[x][0]
    print(data)
    print(mat)
    data.pop(1)
    #print(data)
    get_minimumm(vd, tuple(data), mat)
```

```
    print('\n\nSolution du TSP: {',vdd,',', end='')
    solution = p.pop()
    for cle,value in data.items():
        if cle == solution[1][0]:
            vi = value
            print(vi, end=', ')
            break
    for x in range(n - 2):
        for new_solution in p:
            if tuple(solution[1]) == new_solution[0]:
                solution = new_solution
                for cle,value in data.items():
                    if cle == solution[1][0]:
                        vi = value
                print(vi, end=', ')
                break
    print(vdd,'}')
    return
```

Figure 3 : fonction principale

Voici le résultat obtenu en exécutant la deuxième instance .

```
Entrée [4]: main(instance2)

ville départ : A
{1: 'A', 2: 'B', 3: 'C', 4: 'D', 5: 'E', 6: 'F', 7: 'G', 8: 'H', 9: 'I', 10: 'J', 11: 'K', 12
: 'L', 13: 'M', 14: 'N', 15: 'O', 16: 'P', 17: 'Q', 18: 'R', 19: 'S', 20: 'T'}
[[0, 85, 48, 74, 74, 84, 10, 78, 20, 69, 15, 50, 70, 17, 23, 83, 84, 61, 47, 37], [38, 0, 78,
65, 85, 77, 78, 30, 76, 12, 29, 33, 34, 13, 37, 41, 73, 42, 89, 50], [65, 13, 0, 78, 60, 95,
74, 36, 94, 36, 73, 11, 15, 38, 29, 25, 59, 24, 25, 76], [24, 34, 18, 0, 93, 46, 58, 97, 89,
49, 64, 72, 81, 40, 46, 19, 95, 99, 96, 72], [55, 22, 21, 16, 0, 99, 96, 45, 12, 37, 21, 88,
38, 48, 26, 15, 94, 76, 42, 12], [60, 99, 82, 14, 52, 0, 93, 60, 75, 33, 21, 31, 49, 66, 21,
49, 16, 62, 94, 47], [92, 83, 15, 40, 24, 13, 0, 94, 50, 99, 88, 93, 56, 47, 20, 52, 44, 60,
94, 41], [11, 27, 87, 97, 68, 19, 85, 0, 65, 91, 90, 99, 60, 50, 12, 85, 37, 59, 28, 96], [11
, 87, 83, 80, 13, 69, 78, 89, 0, 93, 37, 65, 70, 82, 27, 75, 98, 36, 89, 41], [48, 88, 66, 21
, 38, 67, 29, 22, 15, 0, 62, 100, 81, 56, 13, 17, 62, 75, 36, 68], [44, 57, 45, 44, 59, 72, 4
8, 27, 89, 63, 0, 87, 43, 13, 54, 72, 23, 13, 16, 38], [76, 40, 99, 52, 82, 79, 96, 39, 53, 5
2, 25, 0, 31, 39, 24, 100, 73, 82, 19, 86], [64, 55, 89, 57, 21, 68, 100, 77, 48, 41, 58, 25,
0, 13, 62, 99, 12, 70, 46, 14], [83, 34, 44, 71, 81, 68, 56, 46, 75, 38, 50, 98, 61, 0, 18, 6
3, 51, 74, 70, 56], [13, 80, 36, 100, 32, 69, 95, 74, 58, 78, 17, 27, 13, 81, 0, 76, 66, 44,
91, 45], [70, 55, 19, 87, 51, 35, 14, 54, 88, 35, 81, 98, 61, 43, 69, 0, 95, 50, 69, 63], [13
, 28, 87, 36, 39, 46, 89, 31, 15, 62, 82, 78, 62, 60, 19, 79, 0, 61, 32, 70], [69, 39, 21, 20
, 79, 74, 67, 98, 15, 88, 60, 35, 51, 27, 62, 44, 38, 0, 65, 19], [79, 67, 12, 55, 87, 33, 36
, 92, 50, 25, 25, 92, 54, 16, 75, 65, 38, 26, 0, 47], [26, 24, 99, 47, 95, 94, 98, 100, 60, 8
9, 83, 33, 18, 17, 35, 31, 40, 44, 10, 0]]


Solution to TSP: { A ,G, F, D, P, C, L, K, R, I, E, T, S, N, O, M, Q, B, J, H, A }
```

Figure 4 : Exécution

Le temps de réponse de la premiere instance était 1s , pour la 2ème instance qui contient 20 villes presque 5min , les autres ont pris beaucoup de temps .

On a vérifier la solution avec une bibliothèque prédéfinie dans python , il faut d'abord installer la bibliothèque avec la commande pip

```
Entrée [1]: pip install python-tsp

Requirement already satisfied: python-tsp in /opt/anaconda3/lib/python3.9/site-packages (0.3.
1)
Requirement already satisfied: numpy in /opt/anaconda3/lib/python3.9/site-packages (from pyth
on-tsp) (1.21.5)
Requirement already satisfied: requests<3.0.0,>=2.28.0 in /opt/anaconda3/lib/python3.9/site-p
ackages (from python-tsp) (2.28.1)
Requirement already satisfied: tsplib95<0.8.0,>=0.7.1 in /opt/anaconda3/lib/python3.9/site-pa
ckages (from python-tsp) (0.7.1)
Requirement already satisfied: certifi>=2017.4.17 in /opt/anaconda3/lib/python3.9/site-packag
es (from requests<3.0.0,>=2.28.0->python-tsp) (2021.10.8)
Requirement already satisfied: charset-normalizer<3,>=2 in /opt/anaconda3/lib/python3.9/site-
packages (from requests<3.0.0,>=2.28.0->python-tsp) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /opt/anaconda3/lib/python3.9/site-pac
kages (from requests<3.0.0,>=2.28.0->python-tsp) (1.26.9)
Requirement already satisfied: idna<4,>=2.5 in /opt/anaconda3/lib/python3.9/site-packages (fr
om requests<3.0.0,>=2.28.0->python-tsp) (3.3)
Requirement already satisfied: Deprecated~=1.2.9 in /opt/anaconda3/lib/python3.9/site-package
s (from tsplib95<0.8.0,>=0.7.1->python-tsp) (1.2.13)
Requirement already satisfied: networkx~=2.1 in /opt/anaconda3/lib/python3.9/site-packages (f
rom tsplib95<0.8.0,>=0.7.1->python-tsp) (2.7.1)
Requirement already satisfied: tabulate~=0.8.7 in /opt/anaconda3/lib/python3.9/site-packages
(from tsplib95<0.8.0,>=0.7.1->python-tsp) (0.8.9)
Requirement already satisfied: Click>=6.0 in /opt/anaconda3/lib/python3.9/site-packages (from
tsplib95<0.8.0,>=0.7.1->python-tsp) (8.0.4)
Requirement already satisfied: wrapt<2,>=1.10 in /opt/anaconda3/lib/python3.9/site-packages (
from Deprecated~=1.2.9->tsplib95<0.8.0,>=0.7.1->python-tsp) (1.12.1)
Note: you may need to restart the kernel to use updated packages.
```

Figure 5: Install Package

Voici le résultat obtenu en exécutant la deuxième instance avec cette bibliothèque , cette dernière affiche un résultat en utilisant la programmation dynamique :

```
Entrée [3]: import numpy as np
from python_tsp.exact import solve_tsp_dynamic_programming

distance_matrix = np.array([
    [0, 85, 48, 74, 74, 84, 10, 78, 20, 69, 15, 50, 70, 17, 23, 83, 84, 61, 47, 37],
    [38, 0, 78, 65, 85, 77, 78, 30, 76, 12, 29, 33, 34, 13, 37, 41, 73, 42, 89, 50],
    [65, 13, 0, 78, 60, 95, 74, 36, 94, 36, 73, 11, 15, 38, 29, 25, 59, 24, 25, 76],
    [24, 34, 18, 0, 93, 46, 58, 97, 89, 49, 64, 72, 81, 40, 46, 19, 95, 99, 96, 72],
    [55, 22, 21, 16, 0, 99, 96, 45, 12, 37, 21, 88, 38, 48, 26, 15, 94, 76, 42, 12],
    [60, 99, 82, 14, 52, 0, 93, 60, 75, 33, 21, 31, 49, 66, 21, 49, 16, 62, 94, 47],
    [92, 83, 15, 40, 24, 13, 0, 94, 50, 99, 88, 93, 56, 47, 20, 52, 44, 60, 94, 41],
    [11, 27, 87, 97, 68, 19, 85, 0, 65, 91, 90, 99, 60, 50, 12, 85, 37, 59, 28, 96],
    [11, 87, 83, 80, 13, 69, 78, 89, 0, 93, 37, 65, 70, 82, 27, 75, 98, 36, 89, 41],
    [48, 88, 66, 21, 38, 67, 29, 22, 15, 0, 62, 100, 81, 56, 13, 17, 62, 75, 36, 68],
    [44, 57, 45, 44, 59, 72, 48, 27, 89, 63, 0, 87, 43, 13, 54, 72, 23, 13, 16, 38],
    [76, 40, 99, 52, 82, 79, 96, 39, 53, 52, 25, 0, 31, 39, 24, 100, 73, 82, 19, 86],
    [64, 55, 89, 57, 21, 68, 100, 77, 48, 41, 58, 25, 0, 13, 62, 99, 12, 70, 46, 14],
    [83, 34, 44, 71, 81, 68, 56, 46, 75, 38, 50, 98, 61, 0, 18, 63, 51, 74, 70, 56],
    [13, 80, 36, 100, 32, 69, 95, 74, 58, 78, 17, 27, 13, 81, 0, 76, 66, 44, 91, 45],
    [70, 55, 19, 87, 51, 35, 14, 54, 88, 35, 81, 98, 61, 43, 69, 0, 95, 50, 69, 63],
    [13, 28, 87, 36, 39, 46, 89, 31, 15, 62, 82, 78, 62, 60, 19, 79, 0, 61, 32, 70],
    [69, 39, 21, 20, 79, 74, 67, 98, 15, 88, 60, 35, 51, 27, 62, 44, 38, 0, 65, 19],
    [79, 67, 12, 55, 87, 33, 36, 92, 50, 25, 25, 92, 54, 16, 75, 65, 38, 26, 0, 47],
    [26, 24, 99, 47, 95, 94, 98, 100, 60, 89, 83, 33, 18, 17, 35, 31, 40, 44, 10, 0]
])
permutation, distance = solve_tsp_dynamic_programming(distance_matrix)
print(permutation, distance)

[0, 6, 5, 3, 15, 2, 11, 10, 17, 8, 4, 19, 18, 13, 14, 12, 16, 1, 9, 7] 306
```


Figure 6 : Verification

- En utilisant la méthode approchée :

d'après nos recherches, il existe de nombreuses procédures de construction heuristiques d'une solution au TSP. On a choisi l'algotithme de **plus proche voisin** :

1. **L'heuristique du plus proche voisin :**

Cet algorithme construit le cycle en faisant croître une chaîne. On part d'un sommet arbitraire à partir duquel on va au sommet voisin le plus proche, puis de celui-là à son plus proche voisin non visité, etc ...., jusqu'à ce que tous les sommets aient été parcourus, où l'on revient au départ.

=>implémentation du code Python :

1.On a crée une fonction qui calcule le transposée d'une matrice pour savoir si la matrice est symétrique ou non , si la matrice est symétrique le programme s'exécutera sans aucun problème , sinon il faut renverser le résultat final .

Entrée [10]:
```
'''Fonction qui calcule le transposée'''
def transpose(matrixx):
    if matrixx == None or len(matrixx) == 0:
        return []

    result = [[None for i in range(len(matrixx))] for j in range(len(matrixx[0]))]

    for i in range(len(matrixx[0])):
        for j in range(len(matrixx)):
            result[i][j] = matrixx[j][i]

    return result
```

Figure 7 : Transposer

2. Puis la fonction principal qui calcul le plus court chemin entre lla ville de départ et le reste des villes dans l'ensemble .

```python
#Le plus court chemin
def glouton(matriceDistance,names):
    ville_départ = input("ville départ : ")
    vdd = ville_départ
    vd = 1
    l = []
    for cle,value in names.items():
        if ville_départ == value:
            vd = cle
            for i in range(len(matriceDistance)):
                tmp2 = matriceDistance[i][0]
                matriceDistance[i][0] = matriceDistance[i][vd-1]
                matriceDistance[i][vd-1] = tmp2
            for j in range(len(matriceDistance[i])):
                tmp = matriceDistance[0][j]
                matriceDistance[0][j] = matriceDistance[vd-1][j]
                matriceDistance[vd-1][j] = tmp
    names[vd] = names[1]
    tmm = names[1]
    names[1] = ville_départ
    ville_départ = tmm
    print(names[1])
    print(matriceDistance)
    print(names)
    liste = [name for name in names]
    solutionConstruite = [1]
    liste.pop(0)
    while len(liste)>0:
        u = solutionConstruite[-1]
        v = liste[-1]
        distance = float("inf")
        for w in liste:
            if matriceDistance[u-1][w-1] < distance:
                distance = matriceDistance[u-1][w-1]
                v = w
        solutionConstruite.append(v)
        liste.remove(v)
    #solutionConstruite.append(1)
    print('\n\nSolution to TSP heuristique : {',names[1],',', end='')
```

```python
if transpose(matriceDistance) != matriceDistance:
    solutionConstruite.reverse()
    for i in solutionConstruite :
        for cle,value in data.items():
            if i == cle :
                vi = value
                print(vi, end=', ')
    print('}')
else :
    for i in solutionConstruite :
        for cle,value in data.items():
            if i == cle :
                vi = value
                #print(vi, end=', ')
                l.append(vi)
    l.reverse()
    print(l)
    print('}')

#return solutionConstruite
```

Figure 8 :Algorithme de Glouton

Voici le chemin obtenu avec la méthode heuristique , l'exécution a pris presque 1s avec l'instance de 10 villes et presque 6min en utilisant 20 villes .

```
Entrée [12]: glouton(matrixx,data)

ville départ : A
A
[[0, 31, 27, 20, 24, 24, 22, 17, 38, 33], [31, 0, 12, 45, 35, 39, 43, 49, 44, 35], [27, 12, 0, 29, 40, 50, 23, 47, 47
, 20], [20, 45, 29, 0, 14, 49, 18, 13, 20, 26], [24, 35, 40, 14, 0, 48, 29, 19, 36, 28], [24, 39, 50, 49, 48, 0, 19,
21, 20, 11], [22, 43, 23, 18, 29, 19, 0, 44, 46, 46], [17, 49, 47, 13, 19, 21, 44, 0, 12, 30], [38, 44, 47, 20, 36, 2
0, 46, 12, 0, 30], [33, 35, 20, 26, 28, 11, 46, 30, 30, 0]]
{1: 'A', 2: 'B', 3: 'C', 4: 'D', 5: 'E', 6: 'F', 7: 'G', 8: 'H', 9: 'I', 10: 'J'}


Solution to TSP heuristique : { A ,['B', 'C', 'G', 'F', 'J', 'E', 'D', 'I', 'H', 'A']
}
```

Figure 9 :Le plus court chemin

On a vérifié le chemin en utilisant une bibliothèque prédéfinie de la mèthode heuristique :

```
Entrée [17]: import numpy as np
from python_tsp.heuristics import solve_tsp_simulated_annealing
distance_matrix = np.array([
    [0, 85, 48, 74, 74, 84, 10, 78, 20, 69, 15, 50, 70, 17, 23, 83, 84, 61, 47, 37],
    [38, 0, 78, 65, 85, 77, 78, 30, 76, 12, 29, 33, 34, 13, 37, 41, 73, 42, 89, 50],
    [65, 13, 0, 78, 60, 95, 74, 36, 94, 36, 73, 11, 15, 38, 29, 25, 59, 24, 25, 76],
    [24, 34, 18, 0, 93, 46, 58, 97, 89, 49, 64, 72, 81, 40, 46, 19, 95, 99, 96, 72],
    [55, 22, 21, 16, 0, 99, 96, 45, 12, 37, 21, 88, 38, 48, 26, 15, 94, 76, 42, 12],
    [60, 99, 82, 14, 52, 0, 93, 60, 75, 33, 21, 31, 49, 66, 21, 49, 16, 62, 94, 47],
    [92, 83, 15, 40, 24, 13, 0, 94, 50, 99, 88, 93, 56, 47, 20, 52, 44, 60, 94, 41],
    [11, 27, 87, 97, 68, 19, 85, 0, 65, 91, 90, 99, 60, 50, 12, 85, 37, 59, 28, 96],
    [11, 87, 83, 80, 13, 69, 78, 89, 0, 93, 37, 65, 70, 82, 27, 75, 98, 36, 89, 41],
    [48, 88, 66, 21, 38, 67, 29, 22, 15, 0, 62, 100, 81, 56, 13, 17, 62, 75, 36, 68],
    [44, 57, 45, 44, 59, 72, 48, 27, 89, 63, 0, 87, 43, 13, 54, 72, 23, 13, 16, 38],
    [76, 40, 99, 52, 82, 79, 96, 39, 53, 52, 25, 0, 31, 39, 24, 100, 73, 82, 19, 86],
    [64, 55, 89, 57, 21, 68, 100, 77, 48, 41, 58, 25, 0, 13, 62, 99, 12, 70, 46, 14],
    [83, 34, 44, 71, 81, 68, 56, 46, 75, 38, 50, 98, 61, 0, 18, 63, 51, 74, 70, 56],
    [13, 80, 36, 100, 32, 69, 95, 74, 58, 78, 17, 27, 13, 81, 0, 76, 66, 44, 91, 45],
    [70, 55, 19, 87, 51, 35, 14, 54, 88, 35, 81, 98, 61, 43, 69, 0, 95, 50, 69, 63],
    [13, 28, 87, 36, 39, 46, 89, 31, 15, 62, 82, 78, 62, 60, 19, 79, 0, 61, 32, 70],
    [69, 39, 21, 20, 79, 74, 67, 98, 15, 88, 60, 35, 51, 27, 62, 44, 38, 0, 65, 19],
    [79, 67, 12, 55, 87, 33, 36, 92, 50, 25, 25, 92, 54, 16, 75, 65, 38, 26, 0, 47],
    [26, 24, 99, 47, 95, 94, 98, 100, 60, 89, 83, 33, 18, 17, 35, 31, 40, 44, 10, 0]
])
permutation, distance = solve_tsp_simulated_annealing(distance_matrix)
print(permutation, distance)

[0, 18, 3, 2, 11, 10, 13, 7, 17, 19, 15, 6, 5, 16, 1, 9, 14, 12, 4, 8] 477
```

Figure 10 : Heuristique-Python

CHAPITRE III

1. Étude préliminaire et fonctionnelle :

• Description des besoins fonctionnels :

- Lecture du fichier excel .
- Exécution de l'instance dans les deux programmes .
- Comparer les résultats et le temps de réponse .
- Définir la méthode la plus optimale .

• Exigences non fonctionnelles :

Il s'agit des besoins qui caractérisent le système. Ce sont des besoins en matière de performance, de type de matériel ou de conception. Ces besoins peuvent concerner les contraintes d'implémentation (langage de programmation, de système d'Exploitation...).
Dans le cadre de notre interface, les besoins non fonctionnels sont :
1.le code devra être extensible, c'est-à-dire qu'il pourra y avoir une possibilité d'ajouter ou de modifier de nouvelles fonctionnalités
2.Le code doit être clair pour permettre des futures évolutions ou améliorations.
3.L'interface doit fournir un accès rapide aux informations .
4.Etre compatible avec n'importe quel système d'exploitation..

2. Environnement du travail :

L'étude technique d'un projet est une expertise en conception et une étude d'avant-projet afin de mener à bien la réalisation des travaux de construction répondant aux attentes.

- **Les outils utilisés :**

Figure 11 : Anaconda

Anaconda est une distribution libre et open source des langages de programmation Python et R appliqué au développement d'applications dédiées à la science des données et à l'apprentissage automatique, qui vise à simplifier la gestion des paquets et de déploiement.

Figure 12 :Jupyter

Le projet Jupyter est un projet dont les objectifs sont de développer des logiciels open source, des normes ouvertes et des services pour l'informatique interactive dans plusieurs langages de programmation.

Figure 12 : Python

Python est un langage de programmation interprété, multiparadigme et multiplateformes. Il favorise la programmation impérative structurée, fonctionnelle et orientée objet

3. Mise en oeuvre du projet:

La mise en œuvre du projet c'est la concrétisation, c'est se lancer dans un plan d'actions opérationnelles. Modéliser le projet et définir un modèle économique est primordial pour dimensionner son action et vérifier que l'on dispose bien des moyens de ses ambitions.

- **L'interface :**

Pour avoir un temp de réponse rapide , il faut executer le fichier "FULLTSP.py" dans Vs code qui se trouve dans Anaconda .

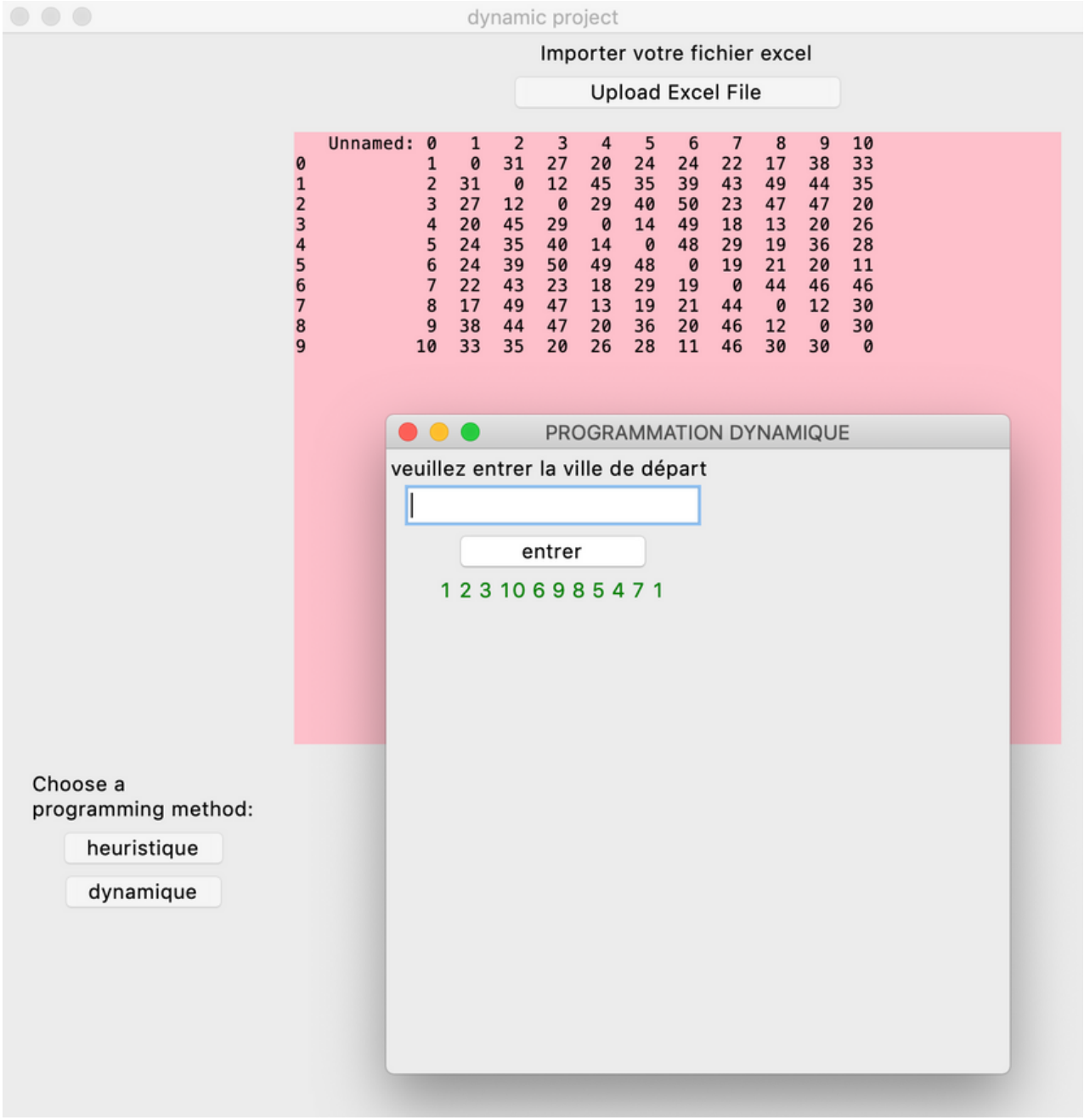

Figure 13 :L'interface

CONCLUSION

Le problème du voyageur de commerce est toujours d'actualité dans la recherche en informatique, étant donné le nombre important de problèmes réels auxquels il correspond. Les problèmes dérivés et les extensions en sont très nombreux. Ce projet m'a beaucoup intéressé, car c'était une expérience très enrichissante et bénéfique et m'a permis d'approfondir mes connaissances, de les augmenter et de découvrir beaucoup de choses, ce qui est profitable pour notre culture informatique.Malgré toutes les difficultés rencontrées dans le processus de développement et les contraintes de temps, j'ai été en mesure de réaliser l'ensemble de mon interface.

WEBOGRAPHIE

- Jupyter : https://fr.wikipedia.org/wiki/Jupyter
- Anaconda https://fr.wikipedia.org/wiki/Anaconda_(distribution_Python)
- Python: : https://fr.wikipedia.org/wiki/Python_(langage)