TPs: Métaheuristiques

Les Imports :

In [1]:
```python
import numpy as np
import random
import math
import time
import matplotlib.pyplot as plt
from typing import Union, List, Tuple
```

Exercice 2 (Voyageur de commerce).

Le célèbre problème du Voyageur de commerce (Travelling Salesman Problem TSP) consiste à trouver une tournée passant par toutes les villes ayant la plus courte distance totale. On dispose pour cela d'un fichier dans lequel on dispose d'une liste de n villes (le nombre n est donné au début du fichier) avec des identifiants et des coordonnées sous la forme : \ Id, x, y où Id est un numéro de ville, et (x,y) sont ses coordonnées, les villes sont toujours données dans l'ordre des Identifiants de 1 à n (Id est donc inutile et peut-être retrouvé avec le numéro de la ligne lue).

Une solution de ce problème est une séquence de n villes, elle représente une tournée partant du point (0,0) puis passant par chacune des villes de la séquence puis revenant au point (0,0). Pour spimplifier, on assimilera la distance entre deux villes à leur distance euclidienne :

$$d(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

Par exemple, en utilisant le fichier tsp5.txt décrivant les coordonnées de 5 villes, la solution $[5,3,4,1,2]$ correspond à une tournée de longueur 263.88 km.

Le problème consiste à trouver une tournée de longueur minimale.

In [2]:
```python
def read_city_file(nom_fichier: str) -> Tuple[int, List[Tuple[int, float, float]]]:
    with open(nom_fichier, "r") as f:
        n = int(f.readline())
        villes = []
        for ligne in f:
            id_ville, x, y = ligne.split()
            villes.append((int(id_ville), float(x), float(y)))
    return n, villes
```

La fonction read_city_file prend en paramètre un nom de fichier de type str et renvoie n le nombre de villes dans le fichier et une liste de Tuple[int, float, float] contenant l'id de la ville est c'est coordonée x et y.

Cette fonction ouvre le fichier dont le nom est donné en paramètre et lit son contenu ligne par ligne. La première ligne du fichier est lue et convertie en entier, qui est stocké dans la variable n.

Ensuite, la fonction crée une liste vide villes qui sera remplie par la suite avec les informations des villes lues dans le fichier.

Enfin, la fonction renvoie un tuple contenant n et la liste villes.

In [3]:
```python
def afficher_solution(solution: List[int], villes: List[Tuple[int, float, float]]):
    villes = [next(v for v in villes if v[0] == id_ville)
              for id_ville in solution]
    x, y = [v[1] for v in villes], [v[2] for v in villes]
    x.insert(0, 0)
    x.append(0)
    y.insert(0, 0)
    y.append(0)
    plt.plot(x, y, "ro")
    plt.plot(x, y, "k--", alpha=0.3)
    plt.title("Affichage de la solution")
    plt.xlabel("Coordonnée x")
    plt.ylabel("Coordonnée y")
    for i, ville in enumerate(villes):
        plt.text(ville[1], ville[2], f"{ville[0]}")
    plt.show()


tsp5 = read_city_file("files/tsp5.txt")
tsp101 = read_city_file("files/tsp101.txt")
```

Cette fonction ci-dessus permet d'afficher graphiquement les villes et le chemin parcouru par la solution données.

Question 2.1 :

Écrire une fonction qui renvoie une solution initiale au hasard pour ce problème, c'est-à-dire qui renvoie un vecteur de n villes tirées aléatoirement (la taille du vecteur sera paramètrable selon la valeur lue dans le fichier utilisé en entrée). \ Pour cette fonction vous pouvez soit utiliser une méthode constructive (créer une liste chaînée des villes et tirer aléatoirement l'indice de la première ville de 1 à n, supprimer la ville correspondante de la liste puis tirer aléatoirement entre 1 et n − 1 la deuxième ville etc) ou une méthode itérative (partir de la solution X où les villes sont dans l'ordre croissant de 1 à n et pour i = 1 à n échanger X[i] et X[random(n)]).

In [4]:
```python
def solution_initiale_au_hasard(n: int) -> List[int]:
    villes = list(range(1, n+1))
    solution = []
    for i in range(n):
        indice_ville = random.randint(0, len(villes)-1)
        solution.append(villes.pop(indice_ville))
    return solution
```

Cette fonction renvoie une solution initiale au hasard pour obtenir un point de depart pour la recherche de la solution optimale.

Question 2.2 :

Écrire une fonction qui calcule la valeur de cette solution c'est-à-dire la distance que le voyageur de commerce doit parcourir en partant de (0,0) puis en passant par toutes les villes $X[1] \dots X[n]$ et en revenant en (0,0).

In [5]:
```python
def calculer_valeur_solution(solution: List[int], villes: List[Tuple[int, float, float]]) -> float:
    distance_totale = 0
    distance_totale += math.sqrt((villes[solution[0]-1][1])
                                 ** 2 + (villes[solution[0]-1][2])**2)
    for i in range(len(solution)-1):
        ville1 = villes[solution[i]-1]
        ville2 = villes[solution[i+1]-1]
        distance_totale += math.sqrt((ville1[1] - ville2[1])
                                     ** 2 + (ville1[2] - ville2[2])**2)
    distance_totale += math.sqrt((villes[solution[-1]-1][1])
                                 ** 2 + (villes[solution[-1]-1][2])**2)
    return distance_totale



print(' Valeur pour la solution [5, 3, 4, 1, 2] :')
print(calculer_valeur_solution( [5, 3, 4, 1, 2], tsp5[1]), 'km')
```

```
 Valeur pour la solution [5, 3, 4, 1, 2] :
263.88370589046286 km
```

Cette fonction permet de calculer la distance totale parcourue par un voyageur de commerce qui visite les villes dans l'ordre donné par la liste solution et retourne à son point de départ.

Question 2.3 :

Programmer une fonction meilleur_voisin qui renvoie la meilleure solution voisine de X où un voisin X' de X est la séquence obtenue en permutant deux villes dans la séquence X.

In [6]:
```python
def meilleur_voisin(solution: List[int], villes: List[Tuple[int, float, float]]) -> List[int]:
    meilleure_solution = solution[:]
    distance_minimale = calculer_valeur_solution(solution, villes)
    for i in range(len(solution)):
        for j in range(i+1, len(solution)):
            solution[i], solution[j] = solution[j], solution[i]
            distance_solution = calculer_valeur_solution(solution, villes)
            if distance_solution < distance_minimale:
                meilleure_solution = solution[:]
                distance_minimale = distance_solution
            solution[i], solution[j] = solution[j], solution[i]
    return meilleure_solution
```

La fonction meilleur_voisin permet de trouver la solution voisine de la solution donnée en entrée qui a la valeur minimale.

Question 2.4 :

Utilisez la méthode du Steepest Hill-Climbing sur les fichiers tsp5.txt et tsp101.txt et les méthodes avec redémarrages en redémarrant MAX_essais fois. Pour chaque essai, vous devrez être capable d'afficher la solution initiale tirée au hasard, la solution atteinte et le nombre de déplacements effectués depuis la solution initiale jusqu'à la solution atteinte.

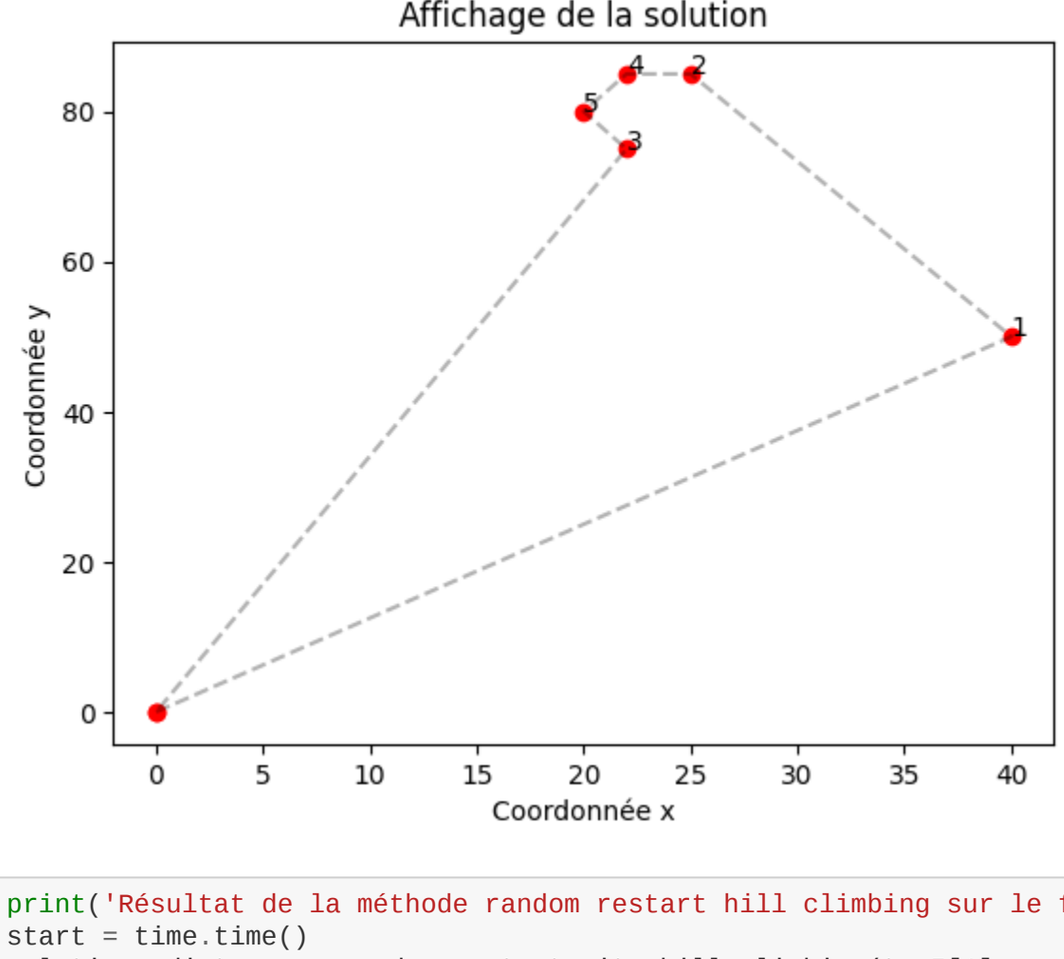
In [7]:
```python
def city_steepest_hill_climbing(villes: List[Tuple[int, float, float]], x: List[int], max_depl: int) -> Union[List[i
nt], int]:
    s_courant = x[:]
    distance_minimale = calculer_valeur_solution(s_courant, villes)
    nb_depl = 0
    stop = False
    while nb_depl < max_depl and not stop:
        s_voisin = meilleur_voisin(s_courant, villes)
        distance_voisin = calculer_valeur_solution(s_voisin, villes)
        if distance_voisin < distance_minimale:
            s_courant = s_voisin[:]
            distance_minimale = distance_voisin
        else:
            stop = True
        nb_depl += 1
    return s_courant, distance_minimale


def random_restart_city_hill_climbing(villes: List[Tuple[int, float, float]], x: List[int], max_depl: int, max_resta
rt: int) -> Union[List[int], int]:
    s_best = x[:]
    distance_minimale = calculer_valeur_solution(s_best, villes)
    nb_restart = 0
    while nb_restart < max_restart:
        s_prime, distance_prime = city_steepest_hill_climbing(
            villes, solution_initiale_au_hasard(len(villes)), max_depl)
        if distance_prime < distance_minimale:
            s_best = s_prime[:]
            distance_minimale = distance_prime
        nb_restart += 1
    return s_best, distance_minimale
```

La fonction city_steepest_hill_climbing essaye trouver la solution optimale d'un voyage de commerce entre des villes en utilisant la fonction meilleur_voisin pour déterminer le meilleur voisin de la solution courante.

La fonction random_restart_city_hill_climbing quand a elle essaye trouver la solution optimale en utilisant city_steepest_hill_climbing avec une solution initiale aléatoire à chaque redémarrage jusqu'à ce que le nombre maximal de redémarrages soit atteint.

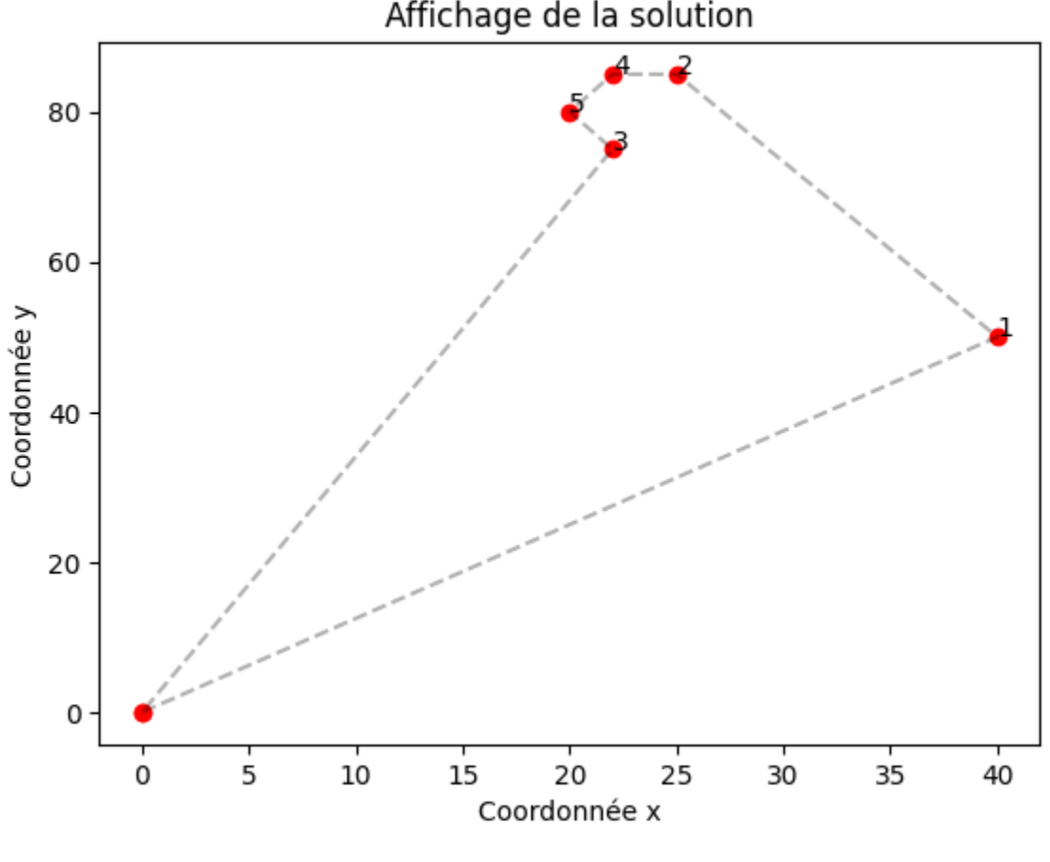
In [39]:
```python
print('Résultat de la méthode steepest hill climbing sur le fichier tsp5 :')
start = time.time()
solution, distance = city_steepest_hill_climbing(tsp5[1],
                                                 solution_initiale_au_hasard(5), 100)
print('Temps de calcul :', time.time() - start, 'secondes')
print(solution, distance, 'km')
afficher_solution(solution, tsp5[1])
```

```
Résultat de la méthode steepest hill climbing sur le fichier tsp5 :
Temps de calcul : 0.0010039806365966797 secondes
[1, 2, 4, 5, 3] 194.04052963659356 km
```



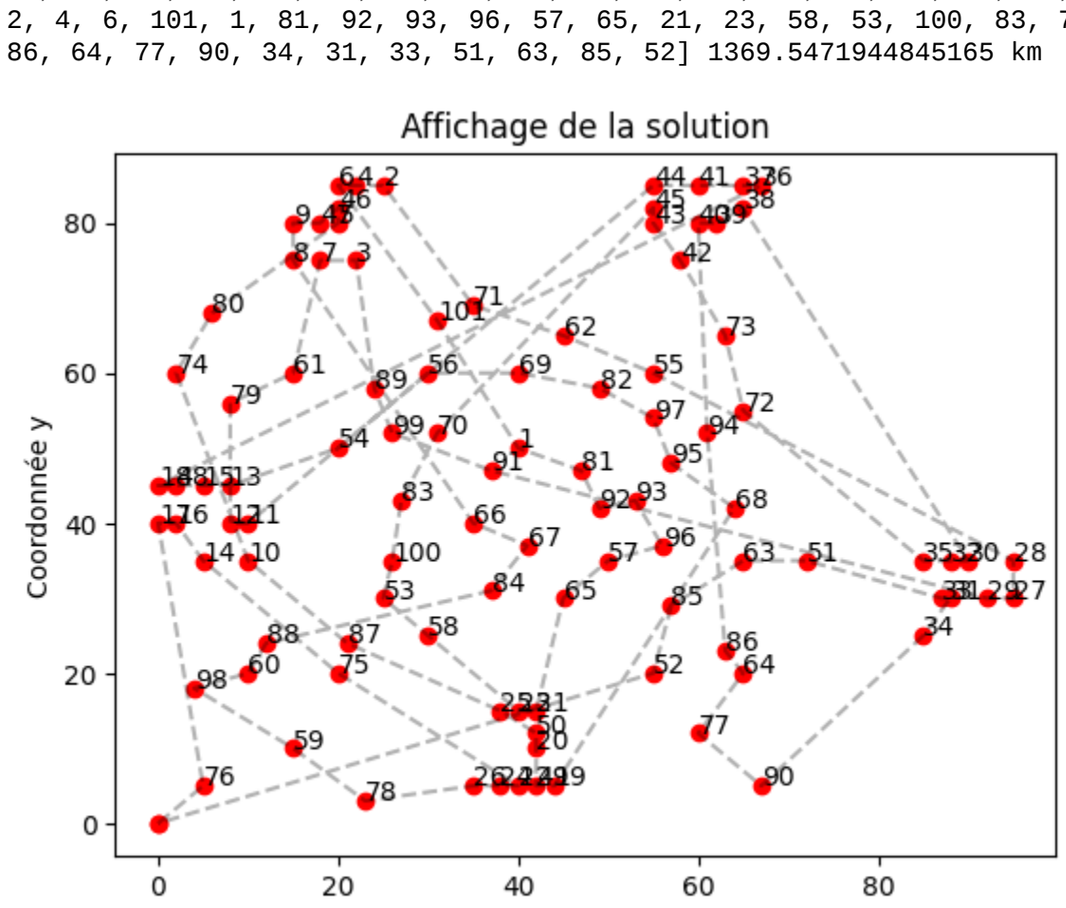
In [41]:
```python
print('Résultat de la méthode random restart hill climbing sur le fichier tsp5 :')
start = time.time()
solution, distance = random_restart_city_hill_climbing(tsp5[1],
                                                 solution_initiale_au_hasard(5), 100, 100)
print('Temps de calcul :', time.time() - start, 'secondes')
print(solution, distance, 'km')
afficher_solution(solution, tsp5[1])
```

```
Résultat de la méthode random restart hill climbing sur le fichier tsp5 :
Temps de calcul : 0.010558366775512695 secondes
[1, 2, 4, 5, 3] 194.04052963659356 km
```



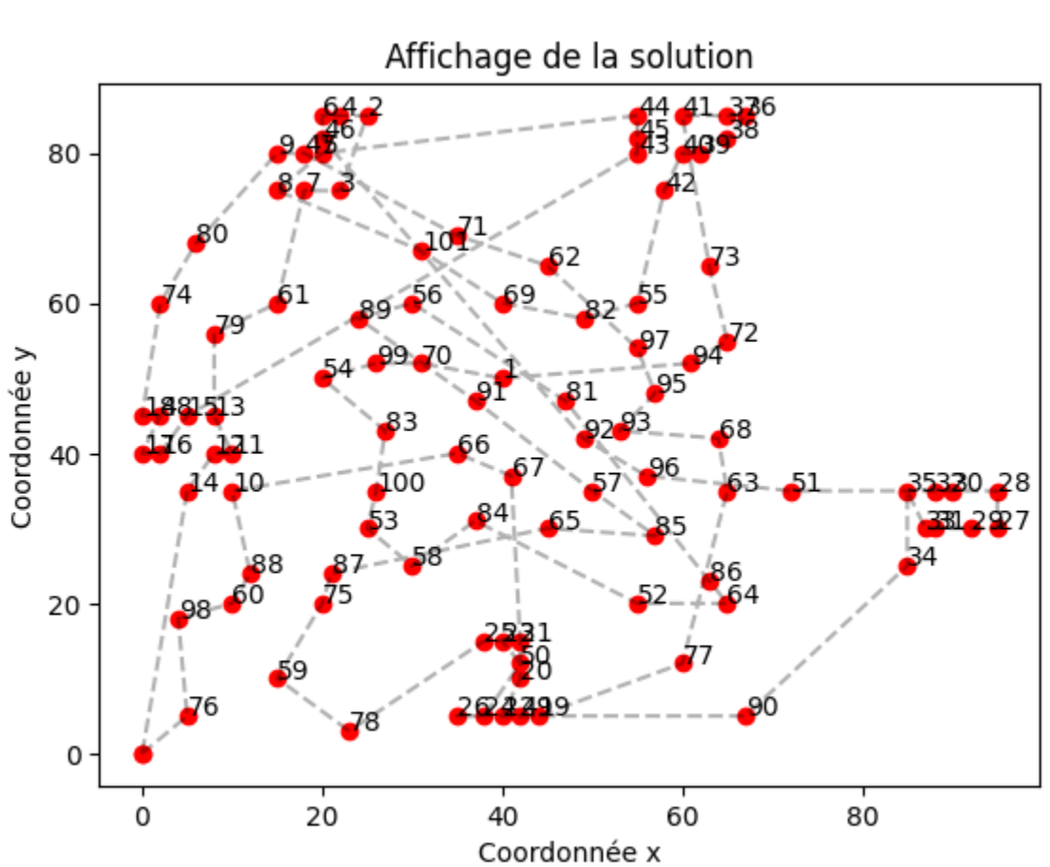
In [42]:
```python
print('Résultat de la méthode steepest hill climbing sur le fichier tsp101 :')
start = time.time()
solution, distance = city_steepest_hill_climbing(tsp101[1],
                                                     solution_initiale_au_hasard(101), 100)

print('Temps de calcul :', time.time() - start, 'secondes')
print(solution, distance, 'km')
afficher_solution(solution, tsp101[1])
```

```
Résultat de la méthode steepest hill climbing sur le fichier tsp101 :
Temps de calcul : 19.293062210083008 secondes
[76, 17, 16, 14, 75, 22, 24, 26, 78, 59, 98, 60, 88, 84, 67, 66, 8, 9, 47, 46, 5, 80, 74, 10, 87, 25, 50, 20, 49, 19,
68, 95, 97, 82, 69, 56, 54, 13, 15, 48, 18, 36, 37, 41, 44, 11, 12, 79, 61, 7, 3, 89, 99, 91, 29, 27, 28, 55, 62, 71,
2, 4, 6, 101, 1, 81, 92, 93, 96, 57, 65, 21, 23, 58, 53, 100, 83, 70, 45, 43, 42, 73, 72, 35, 32, 30, 38, 39, 40, 94,
86, 64, 77, 90, 34, 31, 33, 51, 63, 85, 52] 1369.5471944845165 km
```



In [43]:
```python
print('Résultat de la méthode random restart hill climbing sur le fichier tsp101 :')
start = time.time()
solution, distance = random_restart_city_hill_climbing(tsp101[1],
                                                 solution_initiale_au_hasard(101), 100, 10)

print('Temps de calcul :', time.time() - start, 'secondes')
print(solution, distance, 'km')
afficher_solution(solution, tsp101[1])
```

```
Résultat de la méthode random restart hill climbing sur le fichier tsp101 :
Temps de calcul : 203.67054343223572 secondes
[14, 12, 11, 13, 79, 61, 7, 3, 2, 4, 6, 46, 92, 96, 51, 32, 30, 28, 27, 29, 31, 33, 35, 34, 90, 26, 24, 50, 23, 25, 7
8, 59, 75, 87, 65, 85, 57, 91, 89, 56, 81, 86, 64, 52, 84, 58, 53, 100, 83, 54, 99, 70, 1, 94, 72, 73, 41, 37, 36, 3
8, 39, 40, 42, 55, 82, 69, 101, 8, 5, 44, 45, 43, 15, 16, 17, 48, 18, 74, 80, 9, 47, 71, 62, 97, 95, 93, 68, 63, 77,
19, 49, 22, 20, 21, 67, 66, 10, 88, 60, 98, 76] 1115.91946319573 km
```



Question 2.5 :

Utilisez la méthode tabou en essayant différentes tailles pour la liste tabou et pour MAX_depl. De la même manière, vous devrez pouvoir afficher la solution initiale, la solution atteinte, le nombre de déplacements effectués jusqu'à la solution atteinte, la meilleure solution rencontrée, le contenu de la liste tabou à la fin de la recherche.

In [8]:
```python
def tabou_search(villes: List[Tuple[int, float, float]], x: List[int], max_depl: int, tabou_size: int) -> Union[List
[int], int]:
    s_courant = x[:]
    distance_minimale = calculer_valeur_solution(s_courant, villes)
    tabou = []
    nb_depl = 0
    stop = False
    while nb_depl < max_depl and not stop:
        meilleurs_voisins = []
        distance_minimale_voisins = float("inf")
        for i in range(len(s_courant)):
            for j in range(i+1, len(s_courant)):
                s_voisin = s_courant[:]
                s_voisin[i], s_voisin[j] = s_voisin[j], s_voisin[i]
                distance_voisin = calculer_valeur_solution(s_voisin, villes)
                if distance_voisin < distance_minimale_voisins and s_voisin not in tabou:
                    meilleurs_voisins = [s_voisin]
                    distance_minimale_voisins = distance_voisin
                elif distance_voisin == distance_minimale_voisins and s_voisin not in tabou:
                    meilleurs_voisins.append(s_voisin)
        if meilleurs_voisins:
            s_courant = meilleurs_voisins[np.random.randint(
                0, len(meilleurs_voisins))]
            distance_minimale = distance_minimale_voisins
            tabou.append(s_courant)
            if len(tabou) > tabou_size:
                tabou.pop(0)
        else:
            stop = True
        nb_depl += 1
    return s_courant, distance_minimale
```

La fonction tabou_search utilise une méthode de recherche itérative dans l'espace des solutions du problème du voyageur de commerce pour trouver une solution approximative au problème. La fonction utilise une liste tabou qui contient des solutions interdites temporairement afin d'éviter de retomber sur des solutions déjà explorées et de s'assurer de ne pas rester bloqué dans un minimum local. À chaque itération, la fonction calcule les meilleurs voisins de la solution courante parmi ceux qui ne sont pas dans la liste tabou et met à jour la solution courante et la liste tabou en conséquence. Si aucun meilleur voisin n'est trouvé, la recherche est arrêtée.

In [13]:
```python
print("Solution tsp5 :")
for tabou_size in [5, 10, 20, 50, 100]:
    print(f"    Avec une liste tabou de taille {tabou_size} :")
    for max_depl in [1, 10, 100, 1000, 10000, 100000]:
        start = time.time()
        solution, distance = tabou_search(
            tsp5[1], solution_initiale_au_hasard(5), max_depl, tabou_size)
        print(
            f"      Avec un nombre maximal de déplacements {max_depl}: ")
        print(f"        {solution}(distance={distance: .2f})")
        print(f"        temps de calcule={(time.time()-start): .2f}")
```

```
Solution tsp5 :
    Avec une liste tabou de taille 5 :
      Avec un nombre maximal de déplacements 1:
        [3, 5, 4, 2, 1](distance= 194.04)
        temps de calcule= 0.00
      Avec un nombre maximal de déplacements 10:
        [1, 4, 2, 5, 3](distance= 197.00)
        temps de calcule= 0.00
      Avec un nombre maximal de déplacements 100:
        [1, 3, 5, 2, 4](distance= 198.09)
        temps de calcule= 0.01
      Avec un nombre maximal de déplacements 1000:
        [3, 4, 2, 5, 1](distance= 198.32)
        temps de calcule= 0.04
      Avec un nombre maximal de déplacements 10000:
        [3, 2, 4, 5, 1](distance= 197.07)
        temps de calcule= 0.42
      Avec un nombre maximal de déplacements 100000:
        [5, 2, 4, 3, 1](distance= 197.37)
        temps de calcule= 4.17
    Avec une liste tabou de taille 10 :
      Avec un nombre maximal de déplacements 1:
        [1, 4, 2, 5, 3](distance= 197.00)
        temps de calcule= 0.00
      Avec un nombre maximal de déplacements 10:
        [3, 5, 2, 4, 1](distance= 197.00)
        temps de calcule= 0.00
      Avec un nombre maximal de déplacements 100:
        [1, 3, 4, 2, 5](distance= 197.37)
        temps de calcule= 0.01
      Avec un nombre maximal de déplacements 1000:
        [2, 4, 5, 3, 1](distance= 197.21)
        temps de calcule= 0.04
      Avec un nombre maximal de déplacements 10000:
        [1, 5, 2, 4, 3](distance= 198.32)
        temps de calcule= 0.42
      Avec un nombre maximal de déplacements 100000:
        [1, 3, 4, 2, 5](distance= 197.37)
        temps de calcule= 4.28
    Avec une liste tabou de taille 20 :
      Avec un nombre maximal de déplacements 1:
        [2, 4, 3, 5, 1](distance= 207.07)
        temps de calcule= 0.00
      Avec un nombre maximal de déplacements 10:
        [1, 4, 2, 5, 3](distance= 197.00)
        temps de calcule= 0.00
      Avec un nombre maximal de déplacements 100:
        [3, 4, 2, 5, 1](distance= 198.32)
        temps de calcule= 0.00
      Avec un nombre maximal de déplacements 1000:
        [1, 2, 4, 5, 3](distance= 194.04)
        temps de calcule= 0.04
      Avec un nombre maximal de déplacements 10000:
        [1, 4, 2, 5, 3](distance= 197.00)
        temps de calcule= 0.41
      Avec un nombre maximal de déplacements 100000:
        [1, 3, 4, 2, 5](distance= 197.37)
        temps de calcule= 4.30
    Avec une liste tabou de taille 50 :
      Avec un nombre maximal de déplacements 1:
        [5, 3, 4, 2, 1](distance= 202.96)
        temps de calcule= 0.00
      Avec un nombre maximal de déplacements 10:
        [3, 5, 4, 2, 1](distance= 194.04)
        temps de calcule= 0.00
      Avec un nombre maximal de déplacements 100:
        [3, 5, 4, 1, 2](distance= 254.97)
        temps de calcule= 0.01
      Avec un nombre maximal de déplacements 1000:
        [1, 4, 3, 2, 5](distance= 213.36)
        temps de calcule= 0.05
      Avec un nombre maximal de déplacements 10000:
        [5, 3, 4, 2, 1](distance= 202.96)
        temps de calcule= 0.46
      Avec un nombre maximal de déplacements 100000:
        [5, 2, 3, 4, 1](distance= 213.36)
        temps de calcule= 4.62
    Avec une liste tabou de taille 100 :
      Avec un nombre maximal de déplacements 1:
        [1, 5, 3, 2, 4](distance= 206.71)
        temps de calcule= 0.00
      Avec un nombre maximal de déplacements 10:
        [2, 4, 5, 3, 1](distance= 197.21)
        temps de calcule= 0.00
      Avec un nombre maximal de déplacements 100:
        [4, 1, 3, 5, 2](distance= 259.02)
        temps de calcule= 0.01
      Avec un nombre maximal de déplacements 1000:
        [2, 5, 1, 4, 3](distance= 259.24)
        temps de calcule= 0.00
      Avec un nombre maximal de déplacements 10000:
        [3, 4, 1, 5, 2](distance= 259.24)
        temps de calcule= 0.01
      Avec un nombre maximal de déplacements 100000:
        [5, 2, 3, 1, 4](distance= 257.94)
        temps de calcule= 0.01
```

In [12]:
```python
print("Solution tsp101 :")
for tabou_size in [5, 10, 20, 50, 100]:
    print(f"    Avec une liste tabou de taille {tabou_size} :")
    for max_depl in [1, 10, 100]:
        start = time.time()
        solution, distance = tabou_search(
            tsp101[1], solution_initiale_au_hasard(101), max_depl, tabou_size)
        print(
            f"      Avec un nombre maximal de déplacements {max_depl}: ")
        print(f"        {solution}(distance={distance: .2f})")
        print(f"        temps de calcule={(time.time()-start): .2f}")
```

```
Solution tsp101 :
    Avec une liste tabou de taille 5 :
      Avec un nombre maximal de déplacements 1:
        [86, 95, 41, 20, 22, 26, 63, 83, 70, 74, 82, 19, 52, 11, 69, 8, 85, 48, 84, 36, 76, 33, 64, 21, 45, 43, 25, 5
7, 98, 81, 37, 101, 75, 59, 14, 78, 10, 28, 73, 24, 71, 39, 96, 77, 67, 53, 46, 40, 15, 32, 50, 60, 49, 61, 27, 92, 9
0, 87, 100, 4, 6, 7, 5, 99, 1, 9, 23, 65, 58, 93, 18, 34, 68, 38, 89, 79, 88, 12, 94, 56, 55, 17, 54, 62, 29, 30, 35,
44, 97, 13, 31, 91, 72, 2, 3, 16, 51, 42, 80, 66, 47](distance= 4211.39)
        temps de calcule= 0.41
      Avec un nombre maximal de déplacements 10:
        [22, 35, 94, 88, 83, 99, 74, 86, 31, 101, 18, 79, 92, 95, 13, 12, 98, 58, 32, 30, 29, 7, 40, 66, 11, 61, 15,
14, 89, 3, 42, 39, 4, 17, 71, 37, 34, 19, 52, 75, 48, 51, 68, 57, 25, 26, 50, 91, 69, 73, 82, 8, 96, 55, 43, 44, 38,
36, 62, 45, 41, 56, 87, 1, 84, 97, 72, 46, 9, 70, 64, 100, 24, 23, 49, 81, 53, 21, 59, 47, 2, 5, 6, 27, 28, 77, 90, 6
7, 93, 63, 33, 85, 65, 60, 20, 78, 80, 16, 10, 54, 76](distance= 2980.83)
        temps de calcule= 3.96
      Avec un nombre maximal de déplacements 100:
        [76, 78, 19, 90, 34, 51, 73, 42, 45, 44, 40, 39, 33, 31, 29, 27, 28, 30, 32, 35, 38, 36, 37, 41, 43, 62, 71,
101, 56, 69, 82, 68, 63, 85, 86, 64, 77, 52, 20, 49, 22, 24, 26, 75, 87, 53, 66, 67, 65, 21, 50, 23, 25, 58, 10, 12,
13, 15, 16, 17, 18, 48, 79, 74, 80, 9, 6, 4, 2, 3, 83, 100, 88, 60, 98, 14, 11, 54, 99, 89, 61, 8, 7, 70, 91, 81, 93,
95, 94, 72, 97, 55, 46, 47, 5, 1, 92, 96, 57, 84, 59](distance= 1098.25)
        temps de calcule= 28.25
    Avec une liste tabou de taille 10 :
      Avec un nombre maximal de déplacements 1:
        [98, 31, 62, 83, 2, 43, 56, 30, 92, 21, 35, 55, 71, 90, 9, 58, 26, 78, 73, 63, 49, 7, 29, 61, 20, 81, 25, 64,
91, 23, 74, 12, 5, 6, 94, 39, 80, 99, 37, 93, 47, 52, 8, 24, 18, 84, 11, 22, 79, 67, 68, 95, 14, 70, 89, 54, 34, 3, 5
7, 101, 87, 85, 66, 42, 76, 28, 13, 40, 59, 16, 27, 45, 36, 38, 4, 96, 82, 44, 46, 51, 32, 75, 10, 33, 53, 97, 48, 1
5, 19, 60, 69, 86, 77, 1, 100, 50, 72, 88, 17, 65, 41](distance= 4677.82)
        temps de calcule= 0.30
      Avec un nombre maximal de déplacements 10:
        [76, 98, 14, 80, 11, 100, 59, 84, 92, 70, 89, 16, 95, 62, 40, 6, 46, 4, 38, 37, 93, 82, 27, 28, 94, 99, 52, 3
1, 51, 43, 36, 67, 91, 10, 24, 88, 15, 61, 86, 55, 3, 101, 79, 60, 12, 54, 13, 64, 42, 9, 78, 90, 45, 69, 87, 22, 50,
49, 26, 25, 20, 1, 77, 35, 48, 17, 18, 83, 2, 7, 97, 23, 19, 21, 85, 65, 68, 41, 39, 71, 81, 56, 74, 8, 5, 96, 32, 3
3, 57, 44, 73, 47, 66, 53, 30, 34, 72, 63, 29, 58, 75](distance= 2943.36)
        temps de calcule= 2.87
      Avec un nombre maximal de déplacements 100:
        [78, 26, 24, 20, 75, 87, 58, 85, 33, 31, 32, 94, 82, 69, 62, 97, 95, 65, 25, 23, 21, 52, 34, 29, 35, 57, 67,
66, 91, 43, 45, 44, 40, 42, 80, 74, 79, 54, 89, 99, 70, 56, 3, 4, 2, 71, 93, 96, 92, 81, 1, 101, 5, 6, 46, 47, 9, 11,
88, 60, 10, 12, 15, 48, 18, 59, 22, 49, 19, 50, 53, 7, 8, 61, 83, 100, 84, 30, 28, 27, 90, 77, 64, 86, 63, 51, 68, 7
2, 41, 37, 36, 38, 39, 73, 55, 13, 17, 16, 14, 98, 76](distance= 1349.62)
        temps de calcule= 28.51
    Avec une liste tabou de taille 20 :
      Avec un nombre maximal de déplacements 1:
        [83, 100, 84, 15, 14, 87, 57, 7, 93, 52, 38, 59, 70, 49, 36, 20, 54, 5, 81, 99, 6, 68, 60, 56, 69, 96, 97, 2
5, 58, 37, 17, 39, 74, 50, 16, 1, 11, 62, 53, 76, 89, 73, 98, 10, 19, 44, 8, 26, 78, 33, 30, 55, 12, 91, 48, 4, 79, 6
6, 72, 9, 41, 95, 77, 27, 61, 21, 46, 92, 31, 82, 45, 42, 64, 101, 75, 32, 47, 65, 88, 34, 3, 40, 90, 67, 94, 51, 24,
22, 43, 71, 35, 28, 29, 23, 86, 18, 85, 13, 63, 2, 80](distance= 4589.33)
        temps de calcule= 0.29
      Avec un nombre maximal de déplacements 10:
        [19, 22, 24, 84, 100, 72, 21, 52, 81, 55, 38, 41, 47, 7, 5, 92, 69, 82, 36, 6, 46, 2, 101, 42, 33, 59, 76, 2
0, 60, 48, 79, 83, 71, 3, 9, 89, 17, 43, 40, 45, 37, 68, 53, 66, 1, 65, 28, 25, 10, 11, 14, 18, 78, 26, 49, 70, 85, 9
0, 73, 4, 12, 74, 61, 56, 91, 75, 88, 80, 98, 30, 35, 86, 64, 77, 8, 15, 16, 32, 51, 94, 57, 97, 99, 95, 27, 29, 34,
31, 96, 50, 67, 39, 44, 54, 58, 93, 13, 23, 87, 63, 62](distance= 2867.97)
        temps de calcule= 3.00
      Avec un nombre maximal de déplacements 100:
        [23, 21, 84, 67, 44, 41, 40, 39, 31, 33, 51, 63, 92, 81, 91, 70, 99, 89, 61, 48, 18, 8, 46, 6, 4, 2, 5, 47,
9, 3, 35, 32, 30, 45, 43, 42, 73, 55, 82, 62, 69, 71, 101, 56, 54, 10, 88, 65, 96, 68, 72, 38, 37, 36, 28, 27, 29, 3
4, 90, 77, 22, 24, 25, 53, 100, 83, 66, 50, 20, 49, 19, 57, 93, 95, 94, 97, 1, 11, 12, 14, 16, 17, 15, 13, 79, 74, 8
0, 7, 85, 86, 64, 52, 58, 87, 75, 59, 76, 98, 60, 26, 78](distance= 1341.18)
        temps de calcule= 27.29
    Avec une liste tabou de taille 50 :
      Avec un nombre maximal de déplacements 1:
        [77, 52, 89, 49, 60, 51, 39, 82, 90, 100, 14, 25, 9, 76, 85, 40, 37, 31, 71, 80, 10, 22, 74, 68, 41, 67, 99,
93, 16, 18, 83, 56, 70, 36, 35, 64, 20, 97, 33, 4, 7, 12, 42, 27, 73, 48, 72, 54, 38, 30, 29, 50, 59, 46, 79, 78, 91,
53, 55, 94, 57, 81, 3, 95, 17, 84, 88, 19, 43, 45, 86, 69, 6, 24, 96, 1, 23, 87, 15, 32, 21, 65, 98, 8, 92, 5, 11, 7
5, 47, 61, 34, 2, 66, 28, 26, 44, 13, 62, 101, 63, 58](distance= 4423.02)
        temps de calcule= 0.30
      Avec un nombre maximal de déplacements 10:
        [16, 18, 28, 43, 67, 48, 12, 53, 10, 56, 62, 25, 20, 24, 49, 17, 66, 58, 89, 1, 29, 91, 88, 86, 55, 42, 34, 3
3, 57, 44, 37, 36, 79, 8, 7, 32, 51, 22, 19, 65, 72, 63, 96, 94, 85, 64, 21, 52, 41, 97, 71, 82, 3, 92, 23, 87, 74, 1
4, 11, 80, 99, 27, 31, 77, 50, 35, 30, 93, 45, 100, 84, 9, 39, 38, 70, 61, 54, 46, 4, 5, 13, 75, 101, 15, 69, 83, 6,
95, 60, 98, 78, 76, 59, 73, 81, 40, 47, 2, 90, 68, 26](distance= 3170.47)
        temps de calcule= 2.85
      Avec un nombre maximal de déplacements 100:
        [78, 59, 75, 87, 53, 58, 65, 57, 92, 93, 82, 42, 39, 40, 41, 44, 45, 73, 55, 97, 95, 81, 1, 91, 66, 84, 23, 2
2, 49, 19, 90, 34, 33, 31, 29, 27, 28, 30, 32, 35, 36, 37, 38, 43, 62, 21, 20, 50, 25, 12, 18, 48, 15, 54, 99, 70, 9
6, 85, 67, 83, 74, 80, 8, 7, 3, 89, 61, 79, 13, 11, 52, 86, 63, 69, 9, 47, 5, 46, 6, 4, 2, 71, 101, 56, 10, 88, 60, 1
4, 16, 17, 26, 24, 77, 64, 51, 68, 72, 94, 100, 98, 76](distance= 1220.06)
        temps de calcule= 27.64
    Avec une liste tabou de taille 100 :
      Avec un nombre maximal de déplacements 1:
        [48, 75, 22, 86, 97, 13, 53, 63, 51, 98, 83, 70, 56, 84, 71, 91, 43, 94, 44, 37, 28, 35, 95, 81, 74, 80, 29,
42, 4, 3, 39, 66, 79, 77, 64, 40, 32, 60, 23, 50, 8, 2, 24, 90, 52, 33, 73, 6, 85, 36, 49, 14, 30, 59, 96, 31, 9, 10
0, 21, 26, 87, 25, 15, 92, 58, 72, 93, 78, 1, 18, 65, 38, 17, 20, 55, 88, 5, 101, 69, 47, 54, 16, 99, 61, 34, 89, 45,
12, 76, 57, 41, 19, 46, 68, 67, 82, 62, 7, 27, 11, 10](distance= 4214.81)
        temps de calcule= 0.29
      Avec un nombre maximal de déplacements 10:
        [78, 57, 29, 30, 39, 54, 63, 35, 32, 66, 13, 14, 87, 75, 77, 85, 26, 49, 64, 33, 34, 95, 60, 76, 88, 53, 93,
92, 82, 37, 43, 44, 56, 67, 72, 4, 11, 55, 1, 73, 3, 6, 47, 79, 15, 99, 51, 28, 52, 98, 70, 68, 94, 27, 22, 65, 19, 5
8, 83, 89, 45, 36, 96, 81, 24, 50, 20, 25, 84, 101, 2, 69, 16, 18, 7, 40, 31, 90, 9, 8, 41, 17, 100, 61, 46, 74, 80,
97, 48, 59, 91, 12, 62, 38, 5, 71, 21, 10, 42, 86, 23](distance= 3106.86)
        temps de calcule= 2.88
      Avec un nombre maximal de déplacements 100:
        [98, 13, 15, 89, 70, 81, 97, 95, 93, 92, 61, 7, 3, 101, 71, 62, 69, 1, 82, 43, 45, 44, 41, 39, 35, 32, 29, 2
7, 28, 30, 31, 33, 34, 90, 22, 24, 26, 65, 57, 67, 66, 99, 54, 83, 100, 53, 87, 10, 11, 12, 14, 88, 78, 59, 75, 60, 1
6, 17, 18, 48, 84, 23, 50, 20, 52, 85, 42, 40, 37, 36, 38, 63, 19, 49, 91, 56, 5, 47, 46, 2, 4, 6, 9, 8, 80, 74, 79,
58, 25, 21, 77, 64, 86, 51, 94, 55, 73, 72, 68, 96, 76](distance= 1260.54)
        temps de calcule= 27.63
```