UNIVERSITÉ PARIS 8
VINCENNES-SAINT-DENIS

Mathématiques et applications
Master Cyber Sécurité et Sciences des Données

Optimisation et théorie de la décision

Voyageur du commerce par RT

Mouctar Diallo

Professeur référent
Prof. Boubaker DAACHI

Année académique 2022-2023

Table des matières

1 Introduction

Les métaheuristiques sont des algorithmes de recherche d'optimisation qui sont utilisés pour résoudre des problèmes dans lesquels il est difficile ou impossible de trouver une solution optimale en utilisant des algorithmes exacts en raison de la complexité du problème ou de la taille de l'espace de recherche. Elles sont souvent utilisées pour résoudre des problèmes combinatoires, tels que le voyageur de commerce ou le problème du sac à dos, ou pour optimiser des fonctions mathématiques.

Il existe de nombreuses métaheuristiques différentes, chacune ayant ses propres avantages et inconvénients en fonction du type de problème à résoudre.
Parmi les métaheuristiques les plus couramment utilisées, on peut citer :

1. La Recherche tabou
2. L'algorithme génétique
3. L'algorithme de recuit simulé
4. L'algorithme de colonie de fourmis
5. L'algorithme de recherche à base de particules

En général, les métaheuristiques suivent un processus commun pour trouver une solution optimale :

— Initialisation : définition de la population initiale de solutions candidates
— Évaluation : calcul de la qualité de chaque solution de la population
— Sélection : sélection des solutions les plus prometteuses pour la génération suivante
— Modification : modification des solutions sélectionnées pour générer de nouvelles solutions
— Répéter jusqu'à ce qu'une solution satisfaisante soit trouvée ou que le nombre maximal d'itérations soit atteint

Il est important de noter que les métaheuristiques nécessitent souvent l'utilisation de paramètres de configuration pour contrôler leur comportement et aider à trouver de bonnes solutions. Ces paramètres doivent être choisis de manière appropriée en fonction du problème à résoudre.

2 Recherche Tabou

La Recherche Tabou (en anglais "Taboo Search") est un algorithme de recherche d'optimisation utilisé pour résoudre des problèmes complexes en identifiant les meilleures solutions possibles.

2.1 Principe

Le principe de base de l'algorithme de Recherche tabou est de déplacer l'élément actuel dans la liste vers un autre élément voisin, qui est choisi en fonction d'une fonction de voisinage. Le choix de l'élément voisin est basé sur la valeur de la fonction de coût ou de bénéfice, et sur la présence ou non de l'élément dans une liste tabou, qui est une liste d'éléments interdits temporairement. Si l'élément voisin est meilleur que l'élément actuel selon la fonction de coût ou de bénéfice, il est sélectionné et devient l'élément actuel. Si l'élément voisin n'est pas meilleur, mais n'est pas non plus dans la liste tabou, il peut quand même être sélectionné avec une probabilité décroissante, afin de permettre à l'algorithme de s'échapper de situations locales optimales qui peuvent empêcher la trouvée de la solution optimale globale.

2.2 Liste tabou

La liste tabou est une liste de solutions interdites, qui sont mises à jour à chaque itération de l'algorithme. Les solutions taboues peuvent être interdites pour une période de temps déterminée, ou jusqu'à ce qu'une meilleure solution soit trouvée.

2.3 Voisinage

La liste tabou est une liste de solutions interdites, qui sont mises à jour à chaque itération de l'algorithme. Les solutions taboues peuvent être interdites pour une période de temps déterminée, ou jusqu'à ce qu'une meilleure solution soit trouvée.

2.4 Affectation quadratique

L'affectation quadratique est une technique utilisée pour choisir de manière aléatoire un élément voisin de la liste, en fonction de la probabilité de sélection de chaque élément. Elle consiste à tirer aléatoirement un nombre entier entre 0 et le carré de la longueur de la liste, et à sélectionner l'élément à l'index correspondant dans la liste. Cette technique est utilisée afin de permettre à l'algorithme de s'échapper de situations locales optimales et de trouver des solutions globales optimales.

2.5 Exemple

Supposons que nous voulons minimiser la fonction $f(x, y) = x^2 + y^2$, où x et y sont des variables réelles.
Nous définissons une liste tabou vide et une solution initiale

$$(x0, y0) = (0, 0)$$

.
À chaque itération, nous explorons le voisinage de la solution courante en modifiant chacune des variables x et y de $+1$ ou -1.
Nous évaluons chaque solution du voisinage en utilisant l'affectation quadratique et nous sélectionnons la solution qui a la valeur la plus positive.
Si aucune solution du voisinage n'est meilleure que la solution courante, nous ajoutons la solution courante à la liste tabou et continuons à explorer le voisinage en excluant les solutions taboues.
Nous répétons ce processus jusqu'à ce que nous trouvions la solution optimale ou jusqu'à ce que nous atteignions un nombre maximal d'itérations.

3 Voyageur du commerce

Le voyageur a donc pour mission de visiter chaque ville une seule fois et de ne pas y revenir (sauf pour la ville de départ). La matrice symétrique des distances entre les 12 villes considérées est la suivante :

0	12	13	15	23	25	56	27	38	29	60	32	12
12	0	23	22	34	56	24	78	79	34	23	24	13
13	23	0	25	23	19	18	42	25	23	26	33	15
15	22	25	0	12	24	38	27	38	49	69	29	23
34	23	24	12	0	54	36	71	81	19	100	22	25
23	56	19	24	54	0	16	27	26	37	105	72	56
25	24	18	38	36	16	0	18	23	24	25	26	27
56	78	42	27	71	27	18	0	18	19	30	53	38
27	79	25	38	81	26	23	18	0	29	23	22	29
38	34	23	49	19	37	24	19	29	0	22	42	60
29	23	26	69	100	105	25	30	23	22	0	36	32
60	24	33	29	22	72	26	53	22	42	36	0	24
12	13	15	23	25	56	27	38	29	60	32	24	0

3.1 Code python

[1]:
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Dec 20 14:19:25 2022

@author: Mouctar DIALLO
"""

import random
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

import copy

matrix_cout = [
    [0, 12, 13, 15, 23, 25, 56, 27, 38, 29, 60, 32],
    [12, 0, 23, 22, 34, 56, 24, 78, 79, 34, 23, 24],
    [13, 23, 0, 25, 23, 19, 18, 42, 25, 23, 26, 33],
    [15, 22, 25, 0, 12, 24, 38, 27, 38, 49, 69, 29],
    [23, 34, 23, 12, 0, 54, 36, 71, 81, 19, 100, 22],
    [25, 56, 19, 24, 54, 0, 16, 27, 26, 37, 105, 72],
    [56, 24, 18, 38, 36, 16, 0, 18, 23, 24, 25, 26],
    [27, 78, 42, 27, 71, 27, 18, 0, 18, 19, 30, 53],
    [38, 79, 25, 38, 81, 26, 23, 18, 0, 29, 23, 22],
    [29, 34, 23, 49, 19, 37, 24, 19, 29, 0, 22, 42],
    [60, 23, 26, 69, 100, 105, 25, 30, 23, 22, 0, 36],
    [32, 24, 33, 29, 22, 72, 26, 53, 22, 42, 36, 0]
]

# Nombre de points (Villes)

N = len(matrix_cout)

print('Nombre de ville: ', N)

# Nombre de déplacements à effectuer avant de mettre à jour la
 ↪liste tabou
max_deplacement = 100

# Nombre maximum d'itérations de l'algorithme
MAX_ITERATIONS = 1000

# Liste tabou qui stocke les déplacements interdits
liste_tabou = []

# Fonction qui calcule la distance totale d'un itinéraire donné
def distance_totale(route, matrix_cout):
  distance = 0
  if len(route) > 1:
```

```
    for i in range(len(route)):
      ville_a_idx = route[i]
      ville_b_idx = route[(i+1) % N]
      distance += matrix_cout[ville_a_idx][ville_b_idx]
  return distance


# Fonction qui génère un voisinage à partir d'un itinéraire
 ↪donné
def generateur_voisinage(route):
  voisinage = []
  for i in range(N):
    for j in range(N):
      if i == j:
        continue
      nouvelle_route = copy.copy(route)
      nouvelle_route[i], nouvelle_route[j] =
 ↪nouvelle_route[j], nouvelle_route[i]
      voisinage.append(nouvelle_route)
  return voisinage

# la fonction qui met à jour la liste tabou
def update_liste_tabou(deplacement):
  liste_tabou.append(deplacement)
  if len(liste_tabou) > max_deplacement:
    liste_tabou.pop(0)

# Fonction qui vérifie si un déplacement est interdit par la
 ↪liste tabou
def is_tabu(deplacement):
  return deplacement in liste_tabou

def search(matrix_cout, N, max_deplacement, MAX_ITERATIONS):
  # Génère un itinéraire de départ aléatoire
  #route_actuelle = [6, 10, 8, 11, 1, 2, 0, 3, 4, 9, 7, 5]
  route_actuelle = list(range(N))
  #Mélanger la liste (réorganiser l'ordre des éléments de la
 ↪liste)
  random.shuffle(route_actuelle)

  # Calcul de la distance de l'itinéraire de départ
  distance_actuelle = distance_totale(route_actuelle,
 ↪matrix_cout)

  # Boucle jusqu'à ce que l'on trouve un itinéraire optimal ou
 ↪qu'on atteigne le nombre maximum d'itérations
  for i in range(MAX_ITERATIONS):
    # Génère un voisinage à partir de l'itinéraire courant
    voisinage = generateur_voisinage(route_actuelle)
```

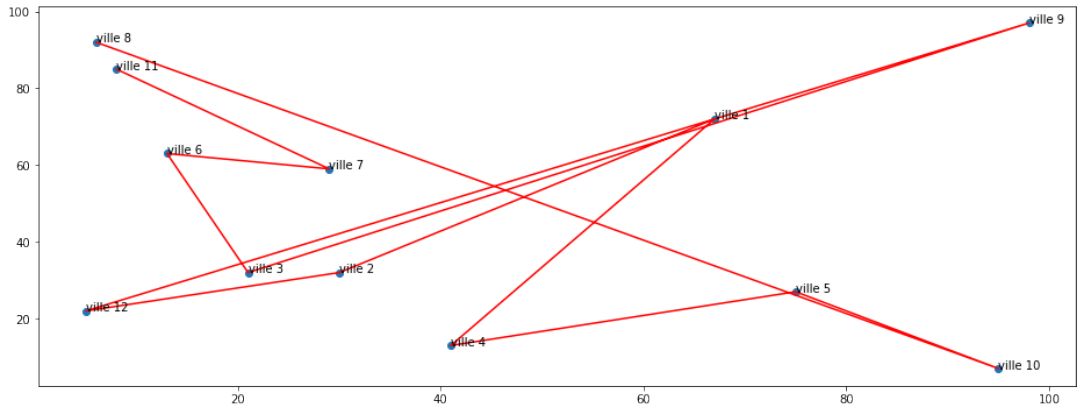
```
    # Initialise les variables meilleur_route et best_distance
 ↪avec des valeurs par défaut
    meilleur_route = None
    best_distance = float('inf')
    # Choisit le meilleur itinéraire du voisinage qui n'est
 ↪pas interdit par la liste tabou
    for j in range(len(voisinage)):
      nouvelle_route = voisinage[j]
      if is_tabu(nouvelle_route):
        continue
      new_distance = distance_totale(nouvelle_route,
 ↪matrix_cout)
      if new_distance < best_distance:
        meilleur_route = nouvelle_route
        best_distance = new_distance

    # Met à jour la liste tabou et l'itinéraire courant
    update_liste_tabou(meilleur_route)
    route_actuelle = meilleur_route
    distance_actuelle = best_distance

  return route_actuelle

# Appelle la fonction de recherche tabou
meilleur_route = search(matrix_cout, N, max_deplacement,
 ↪MAX_ITERATIONS)

# Affiche le résultat
print("Meilleur itinéraire trouvé :", meilleur_route)
print("Distance totale :", distance_totale(meilleur_route,
 ↪matrix_cout))
```

```
Nombre de ville:  12
Meilleur itinéraire trouvé : [4, 11, 6, 5, 7, 9, 10, 8, 2, 1,
 ↪0, 3]
Distance totale : 242
```

3.2 Illustrations graphiques

[2]:
```
# Génération aléatoire des noms de chaque ville
villes = [f'ville {i+1}' for i in range(N)]


# Génération aléatoire des coordonnées x et y de chaque ville
coords = [(random.randint(0, 100), random.randint(0, 100)) for
 ↪_ in range(N)]



# Récupérez les coordonnées des villes à partir de la matrice
 ↪de distance
```

```python
# (ici, nous supposons que chaque ville a des coordonnées x et
 ↪y)
x = [coord[0] for coord in coords]
y = [coord[1] for coord in coords]

# Créez un nouveau graphique et affichez les points
 ↪correspondant aux villes
plt.figure(figsize=(16, 6))
plt.scatter(x, y)

# Ajoutez des étiquettes sur chaque point (ici, nous utilisons
 ↪les noms des villes)
for i, ville in enumerate(villes):
    plt.annotate(ville, (x[i], y[i]))

# Affichez le graphique

# Créez une liste de tuples (x, y) qui contient les
 ↪coordonnées de chaque ville
# dans l'ordre de l'itinéraire courant
route_coords = [(x[ville], y[ville]) for ville in
 ↪meilleur_route]

# Créez un nouveau graphique et tracez une ligne entre chaque
 ↪ville


plt.plot(*zip(*route_coords), color="red")


# Affichez le graphique
plt.show()
```



4 Conclusion

En conclusion, la Recherche tabou est une métaheuristique de recherche d'optimisation utilisée pour résoudre des problèmes difficiles ou impossibles à résoudre avec des algorithmes exacts. Elle fonctionne en déplaçant l'élément actuel de la liste vers un élément voisin choisi en fonction d'une fonction de voisinage, en prenant en compte la valeur de la fonction de coût ou de bénéfice et en évitant les éléments de la liste tabou. La Recherche tabou est particulièrement utile pour échapper aux situations locales optimales et trouver des solutions globales optimales. Cependant, elle nécessite l'utilisation de paramètres de configuration pour contrôler son comportement et aider à trouver de bonnes solutions, et ne garantit pas de trouver la solution optimale.

Références

[1] Chaine youtube de Madame Orkia Derkaoui : https://www.youtube.com/watch?v=0T85WezYgeQ

[2] Laurent DUMAS : https://dumas.perso.math.cnrs.fr/