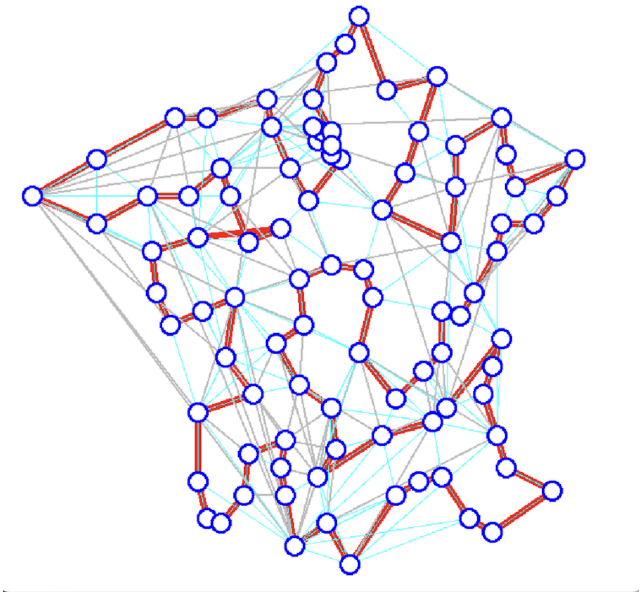
AI09 - Projet

Problème de tournées sélectives



Étudiants Clément MARTINS
Quentin VALAKOU

Enseignants responsables Aziz MOUKRIM

Date Automne 2024

I. Introduction

L'objectif de ce rapport est de développer plusieurs méthodes de résolutions pour le problème du TOP (Team Orienteering Problem). Le TOP est modélisé par un graphe complet : $G = (V, E)$ où $V = \{1, ..., n\} \cup \{d, a\}$ est l'ensemble des sommets et $E = \{i, j\} \quad i, j \in V$ est l'ensemble des arcs.

- Une flotte F est composée de m véhicules devant aller livrer des clients. Chaque véhicule possède un parcours de temps limite L considéré qu'il ne doit pas dépasser.
- Le sommet a (resp. d) représente le point de départ (resp. arrivé) des véhicules.
- Chaque client i est associé à un profit P_i et ne peut être collecté qu'une seule fois.
- Un temps de trajet C_{ij} est associé à chaque arc (i, j).
- V Représente l'enssemble des sommets visité par un véhicule
- U est l'ensemble des sommets

Nous allons donc procéder en deux méthodes distinctes : par la programmation en variables entières, et une adaptation des algorithmes de TSP/VRP programmés en Python.

II. Modélisation pour la programmation linéaire

La programmation linéaire est une modélisation mathématique par des combinaisons linéaires de variables respectant une liste de contraintes données. Une fois la modélisation mathématique completée, il est possible de calculer les solution grâce à un solveur. La complexité est dépendante du problème.

Variables :

- $x_{i,j}^k \in \{0, 1\}$: 1 si le véhicule a parcouru l'arc (i, j), 0 sinon.
- u_i^k : 1 si le véhicule k pass par le sommet i, 0 sinon

Paramètres :

- P_i : profit associé au client i.
- C_{ij} : coût de trajet de l'arc (i, j).
- L : temps de trajet limite des véhicules.
- m : Nombre de véhicules.
- n : Nombre de sommet

Objectif :

- Maximiser : $Max \sum_{k=1}^{m} \sum_{i=2}^{n} P_i * u_i^k$: soit le profit pour l'ensemble des tournées réalisés par les m vehicules de la flotte F.

Contraintes :

1. **Limite de distance (temps) par véhicule**

$$\forall k \in \{1, \ldots, m\}, \quad \sum_{i \in \mathrm{V}} C_{i,j} * u_i^k \leq L$$

2. **Chaque sommet (sauf a et d) doit avoir exactement deux arcs entrants et sortants, si $u_i^k = 1$.**

$$\forall k \in \{1, \ldots, m\}, \forall i \in \mathrm{V} \setminus \{a, d\}, \quad 2 \cdot u_i^k = \sum_{j \in \mathrm{V}, j \neq i} x_{j,i}^k + \sum_{j \in \mathrm{V}, j \neq i} x_{i,j}^k$$

3. **Passe par a**

$$\forall k \in \{1, \ldots, m\}, \quad u_{k,a} = 1$$

4. **Passe par d**

$$\forall k \in \{1, \ldots, m\}, \quad u_{k,d} = 1$$

5. **Chaque sommet (sauf a et d) est visité au plus une fois**

$$\forall i \in \mathrm{V} \setminus \{a, d\}, \quad \sum_{k=1}^{m} u_i^k \leq 1$$

6. **Élimination des sous-tours** Si un véhicule traverse de i à j, il ne peut pas revenir immédiatement de j à i dans le même trajet.

$$\forall k \in \{1, \ldots, m\}, \forall i, j \in \mathrm{V}, i \neq j, \quad x_{k,i,j} + x_{k,j,i} \leq 1$$

III. Algorithmes pour le TOP

Dans cette partie, nous allons développer deux algorithmes pour résoudre le problème de tournées sélectives.

— **Algorithme 1 : algorithme de programmation linéaire**. À l'aide de la modélisation présentée en partie 1, nous allons écrire un algorithme en langage Python avec les libaires numpy (représentation matricielle, matplotlib (création de graphique), xpress (API de programmation linéaire avec contraintes).

— **Algorithme 2 : algorithme glouton (ou heuristique gloutonne)** qui va chercher à chaque itération une solution locale optimale. Les algorithmes gloutons ne garantissent pas l'obtention d'une solution optimale et peuvent avoir une complexité supérieure mais sont plus aisés à implémenter. Cette algorithme sera écrit grâce à la librairie networkX pour la création de graphe.

L'objectif est de comparer les deux algorithmes sur deux critères : leur capacité à trouver une (ou plusieurs) solutions (optimale ou non) et leur complexité temporelle.

Pour exécuter le code sur votre machine, assurez vous d'installer Python et les dépendances suivantes grâce à Pip sur votre environnement :

```
pip install matplotlib
pip install numpy
pip install networkx
pip install xpress
pip install itertools
```

Le code est disponible sur Gitlab : Problème de tournées

III.1. Algorithme 1 : par la programmation linéaire

III.1.1. Visualisation des Points et des Gains

Nous avons modélisé les points du problème, incluant les clients, ainsi que les profits associés à chaque client. Les clients sont représentés par des points dans un plan 2D, et chaque point a un profit associé qui est affiché sur le graphique. Le point de départ a est situé en $(0, 0)$ et le point d'arrivée d en $(10, 10)$.

Voici la visualisation des points (clients, départ, arrivée) et des gains associés :

III.1.2. Génération du modèle

```python
import matplotlib.pyplot as plt
import numpy as np

# Nombre de clients
n_clients = 5

# Positions des clients et points de départ (a) et d'arrivée (d)
points = {
    'a': (0, 0, 0),  # Point de départ
    'd': (10, 10, 0),  # Point d'arrivée
}

# Génération de positions aléatoires pour les clients
#ainsi que le gain associé (3 ème chiffre)
np.random.seed(50)  # Pour la reproductibilité
for i in range(1, n_clients + 1):
    points[i] = (np.random.randint(1, 10),
                 np.random.randint(1, 10),
                 np.random.randint(1, 100))

# Création du graphique
plt.figure(figsize=(8, 8))
for key, value in points.items():
    plt.scatter(value[0], value[1],
        label=f'Client {key}'
        if isinstance(key, int) else key)
    plt.text(value[0] + 0.2, value[1] + 0.2,
            f'Gain: {value[2]}',
            fontsize=12)

# Titres et légende
plt.title('Représentation des points',
```

```
        fontsize=14)
plt.xlabel('Coordonnée X')
plt.ylabel('Coordonnée Y')
plt.legend(loc='upper left')
plt.grid(True)
plt.show()
```

III.1.3. Affichage des Coordonnées et Profits des Clients

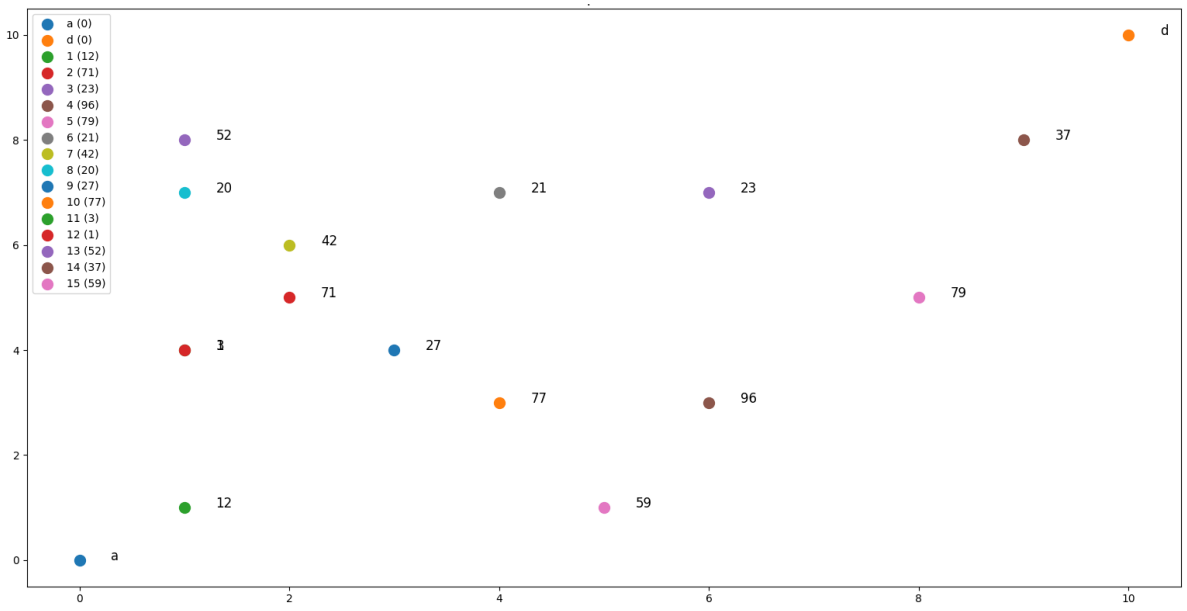


Figure 1 - Modélisation des clients avec leur gain associé.

III.1.4. Programmation du problème avec XpressNP

On doit d'abord définir les gains, coût, ainsi que nos derniers paramètres :

```python
#Calculer les distances entre les points
def distance(p1, p2):
    return np.sqrt((p1[0] - p2[0])**2 + (p1[1] - p2[1])**2)

# Calculer la matrice des distances
C = {}
for i in points:
    for j in points:
        if i != j:
            C[(i, j)] = distance(points[i], points[j])

# Définir les gains associés aux clients
P = {i: points[i][2] for i in range(1, n_clients + 1)}

# Paramètres du problème
L = 15  # Distance limite par véhicule
#(La diagonale est en 14, on leur laisse un peu de marge)
# Nombre de véhicules
m = 4
# Nombre de sommets (clients + points de départ et d'arrivée)
n = len(points)

```

Matrice des distances :

points	*a*	*d*	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a	0	14.14	1.41	5.38	9.22	6.71	9.43	8.06	6.32	7.07	5.00	5.00	4.12	4.12	8.06	12.04	5.10
d	14.14	0	12.73	9.43	5.00	8.06	5.39	6.71	8.94	9.49	9.22	9.22	10.82	10.82	9.22	2.24	10.30
1	1.41	12.73	0	4.12	7.81	5.39	8.06	6.71	5.10	6.00	3.61	3.61	3.00	3.00	7.00	10.63	4.00
2	5.38	9.43	4.12	0	4.47	4.47	6.00	2.83	1.00	2.24	1.41	2.83	1.41	1.41	3.16	7.62	5.00
3	9.22	5.00	7.81	4.47	0	4.00	2.83	2.00	4.12	5.00	4.24	4.47	5.83	5.83	5.10	3.16	6.08
4	6.71	8.06	5.39	4.47	4.00	0	2.83	4.47	5.00	6.40	3.16	2.00	5.10	5.10	7.07	5.83	2.24
5	9.43	5.39	8.06	6.00	2.83	2.83	0	4.47	6.08	7.28	5.10	4.47	7.07	7.07	7.62	3.16	5.00
6	8.06	6.71	6.71	2.83	2.00	4.47	4.47	0	2.24	3.00	3.16	4.00	4.24	4.24	3.16	5.10	6.08
7	6.32	8.94	5.10	1.00	4.12	5.00	6.08	2.24	0	1.41	2.24	3.61	2.24	2.24	2.24	7.28	5.83
8	7.07	9.49	6.00	2.24	5.00	6.40	7.28	3.00	1.41	0	3.61	5.00	3.00	3.00	1.00	8.06	7.21
9	5.00	9.22	3.61	1.41	4.24	3.16	5.10	3.16	2.24	3.61	0	1.41	2.00	2.00	4.47	7.21	3.61
10	5.00	9.22	3.61	2.83	4.47	2.00	4.47	4.00	3.61	5.00	1.41	0	3.16	3.16	5.83	7.07	2.24
11	4.12	10.82	3.00	1.41	5.83	5.10	7.07	4.24	2.24	3.00	2.00	3.16	0	0.00	4.00	8.94	5.00
12	4.12	10.82	3.00	1.41	5.83	5.10	7.07	4.24	2.24	3.00	2.00	3.16	0.00	0	4.00	8.94	5.00
13	8.06	9.22	7.00	3.16	5.10	7.07	7.62	3.16	2.24	1.00	4.47	5.83	4.00	4.00	0	8.00	8.06
14	12.04	2.24	10.63	7.62	3.16	5.83	3.16	5.10	7.28	8.06	7.21	7.07	8.94	8.94	8.00	0	8.06
15	5.10	10.30	4.00	5.00	6.08	2.24	5.00	6.08	5.83	7.21	3.61	2.24	5.00	5.00	8.06	8.06	0

III.1.5. Programmation linéaire avec Xpress

```
Définition des Variables et de notre objectif:
model = xp.problem()

# Variables de décision
x = {(k, i, j): xp.var(vartype=xp.binary)
    for k in range(m)
    for i in points
    for j in points
    if i != j
}
u = {(k, i): xp.var(vartype=xp.binary)
    for k in range(m)
    for i in points
}

# Ajout des variables au modèle
model.addVariable(x)
model.addVariable(u)

# Objectif : Maximiser les profits
model.setObjective(
    xp.Sum(P[i] * u[k, i]
        for k in range(m)
        for i in points
        if i != 'a' and i != 'd'
    ),
    sense=xp.maximize
)
```

On ajoute ensuite nos contraintes

```
# Contraintes

# Limite de distance (temps) par véhicule
for k in range(m):
    model.addConstraint(
        xp.Sum(C[i, j] * x[k, i, j]
            for i in points
            for j in points
            if i != j
        ) <= L  # Pas plus de L par véhicule
    )
```

```
12
13  # Lier les variables u[k, i] et x[k, i, j] et ainsi
14  obligé qu'il y ai 2 ARC sur chaque sommet
15  for k in range(m):
16      for i in points:
17          if i not in ['a', 'd']:
18              model.addConstraint(
19                  (2*u[k, i]
20                  ==
21                  (xp.Sum(x[k, j, i] for j in points if j != i) +
22                  xp.Sum(x[k,i,j] for j in points if j != i)))
23              )
24
25  #Démare par a
26  for k in range(m):
27      model.addConstraint(
28          xp.Sum(x[k, 'a', j] for j in points if j!='a') == 1
29      )
30      model.addConstraint(
31          xp.Sum(u[k, 'a'] ) == 1
32      )
33
34
35  #Fini par d
36  for k in range(m):
37      model.addConstraint(
38          xp.Sum(x[k, j, 'd'] for j in points if j!='d') == 1
39      )
40      model.addConstraint(
41          xp.Sum(u[k, 'd'] ) == 1
42      )
43
44  # Chaque sommet est visité une seul fois
45  for i in points:
46      if i not in ['a', 'd']:
47          model.addConstraint(
48              xp.Sum(u[k, i] for k in range(m)) <= 1
49          )
50
51  # Ajout des contraintes d'élimination des sous-tours
52  for k in range(m):
53      for i, j in itertools.combinations([i for i in points], 2):
54          # Contrainte pour empêcher le sous-tour : si on va de i à j,
55          # alors on ne peut pas revenir de j à i dans le même trajet
```

```
        model.addConstraint(
            x[k, i, j] + x[k, j, i] <= 1
            # Si x[k, i, j] = 1, alors x[k, j, i] doit être 0
        )

```

Il nous reste à génerer la meilleure solution et d'afficher les traits correspondants au meilleur chemin

```

# Résolution
model.solve()

# Résultats
# Vérification de la solution
routes = {k: [] for k in range(m)}
# Stocker les routes pour chaque véhicule
total_costs = {k: 0 for k in range(m)}
# Stocker les coûts totaux pour chaque véhicule
total_profits = {k: 0 for k in range(m)}
# Stocker les gains totaux pour chaque véhicule

print("\nSolution optimale trouvée :")
for k in range(m):
    print(f"\nVéhicule {k+1}:")
    for i in points:
        for j in points:
            if i != j and model.getSolution(x[k, i, j]) > 0.5:
                routes[k].append((i, j))
                cost = C[i, j]  # Coût pour le chemin de i à j
                total_costs[k] += cost
                print(f" - Traverse de {i} à {j} (Coût : {cost})")

    # Calcul des gains des clients pour ce véhicule
    vehicle_profit = sum(
        P[i] * model.getSolution(u[k, i])
            for i in points
            if i != 'a' and i != 'd'
    )
    total_profits[k] = vehicle_profit

    # Affichage des totaux
    print(f"  Coût total pour le véhicule {k+1} : {total_costs[k]}")
```

```
    print(f"  Gain total pour le véhicule {k+1} : {total_profits[k]}")

# Calcul des totaux globaux
total_cost = sum(total_costs.values())
total_profit = sum(total_profits.values())

# Résumé global
print("\nRésumé global :")
print(f"Gain total : {total_profit}")
# Visualisation du graphe
plt.figure(figsize=(8, 8))
colors = ['blue', 'green', 'red', 'orange', 'purple']
# Couleurs pour les véhicules

# Tracer les points
for i in points:
    plt.scatter(points[i][0],
                points[i][1],
                label=f"{i} ({points[i][2]})",
                s=100)

# Tracer les lignes pour chaque véhicule
for k, route in routes.items():
    for (i, j) in route:
        plt.plot([points[i][0],
                  points[j][0]],
                  [points[i][1],
                  points[j][1]],
                  color=colors[k % len(colors)],
                  linestyle='-',
                  linewidth=2)

# Ajouter des labels pour chaque point (exclure 'a' et 'd')
#On affichera le gain associé à chaque point
for i in points:
    if i not in ['a', 'd']:
        plt.text(points[i][0] + 0.3,
                points[i][1], f"{P[i]}",
                fontsize=12)
    else :
        plt.text(points[i][0] + 0.3,
                points[i][1], f"{i}",
                fontsize=12)

```

```
79
80  # Légende
81  plt.legend()
82  plt.title("Solution du problème de VRP")
83  plt.show()
```

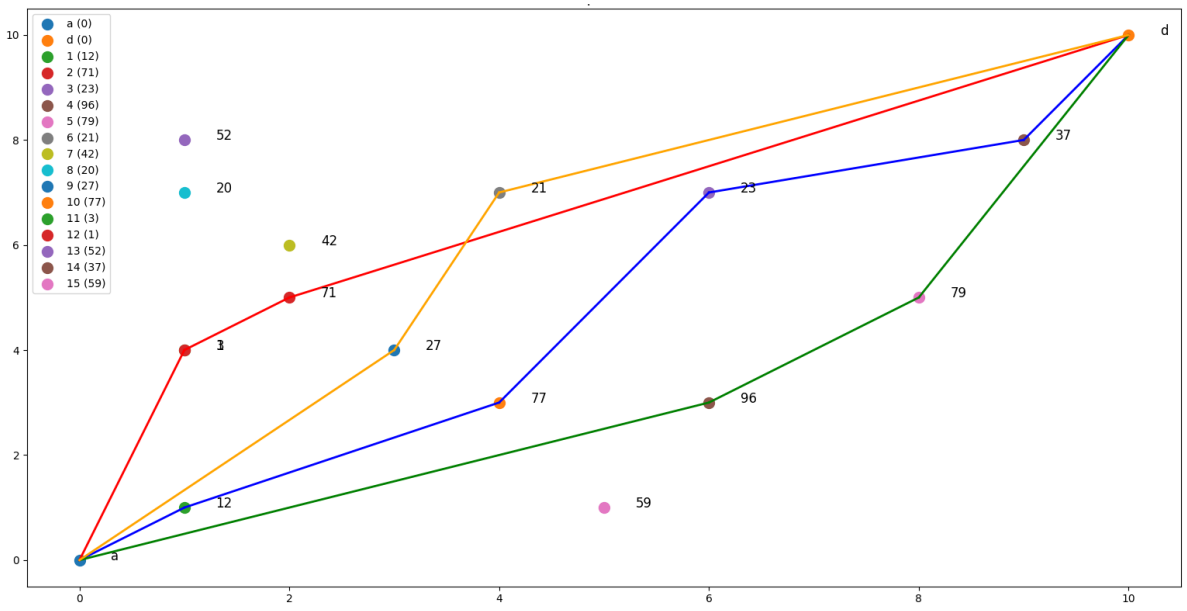


Figure 2 - Solution optimale via Xpress.

y
Elle correspond au chemin :

III.1.6. Tableau des résultats

Véhicule 1 :

- Traverse de a à 1 (Coût : 1.41)
- Traverse de 1 à 10 (Coût : 3.60)
- Traverse de 10 à 3 (Coût : 4.47)
- Traverse de 3 à 14 (Coût : 3.16)
- Traverse de 14 à d (Coût : 2.23)

Résumé Véhicule 1 :

- Coût total : 14.89
- Gain total : 149

Véhicule 2 :

- Traverse de a à 4 (Coût : 7.70)
- Traverse de 4 à 5 (Coût : 2.82)
- Traverse de 5 à d (Coût : 5.38)

Résumé Véhicule 2 :

— Coût total : 14.92
— Gain total : 175

Véhicule 3 :

— Traverse de a à 12 (Coût : 4.12)
— Traverse de 12 à 11 (Coût : 0)
— Traverse de 11 à 2 (Coût : 1.41)
— Traverse de 2 à d (Coût : 9.43)

Résumé Véhicule 3 :

— Coût total : 14.97
— Gain total : 75

Véhicule 4 :

— Traverse de a à 9 (Coût : 5.00)
— Traverse de 9 à 6 (Coût : 3.16)
— Traverse de 6 à d (Coût : 6.70)

Résumé Véhicule 4 :

— Coût total : 14.87
— Gain total : 48

Résumé global :

— Coût total de l'ensemble de la flotte : 59.6538
— Gain total de l'ensemble de la flotte : 447

La solution à été trouvé en 3.82sec

III.2. Algorithme 2 : Heuristique gloutonne

L'algorithme glouton pour ce problème de tournée sélective est simple :

Algorithm 1 Algorithme glouton pour le TOP

```
1: procedure GLOUTON(F, G)
2:     trajet ← []
3:     NonVisitedClient ← G.nodes()
4:     for i in range F do
5:         traget ← [A]
6:         totalCost ← 0
7:         totalProfit ← 0
8:         while NonVisitedClient is not null do
9:             bestC ← null
10:            bestP ← 0
11:            for c in range trajet.voisin do
12:                if trajet.voisin in NonVisitedClient then
13:                    p ← c.profit()
14:                    cost ← c.cost() + d.cost()
15:                    if totalCost + cost ≤ L then
16:                        if profit > bestP then
17:                            bestP ← profit
18:                            bestC ← c
19:                        end if
20:                    end if
21:                end if
22:            end for
23:            if bestC not null then
24:                trajet ← trajet.append(bestC)
25:                totalCost ← totalCost + c.cost()
26:                totalProfit ← totalProfit + c.profit()
27:                NonVisitedClient.remove(c)
28:            end if
29:        end while
30:    end for
31:    return trajet, profit, G
32: end procedure
```