Résolution du problème de tournées des véhicules

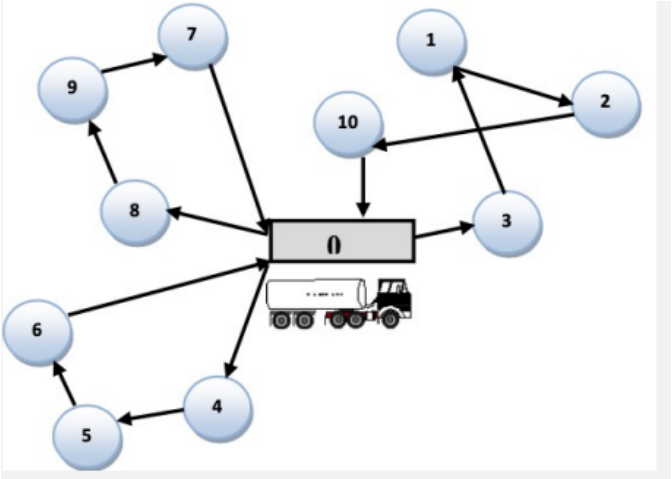
Rapport technique

Floris ROBART - Tran thai Duc DINH - Bryan Lemagnen

Table des matières

Introduction

Le projet en question se penche sur une problématique cruciale dans le domaine de la logistique et de la distribution : l'optimisation des tournées d'une flotte de véhicules chargés de livrer divers clients à partir d'un dépôt central. Cette tâche complexe est caractérisée par la présence de plusieurs contraintes, telles que la capacité limitée de chaque véhicule, l'obligation de satisfaire la demande spécifique de chaque client, et la nécessité pour chaque véhicule de retourner au dépôt après avoir effectué sa tournée. L'objectif principal de ce projet est de développer des solutions efficaces pour minimiser la distance parcourue par l'ensemble des véhicules, optimisant ainsi le processus de livraison dans son ensemble. Il convient de noter que ce défi s'inscrit dans la lignée du problème classique des tournées de véhicules, mais avec des paramètres en moins tels que les horaires et la gestion du carburant.



Formulation Mathématique

Données

- C est le nombre de clients
- N est l'ensemble de clients que $N = \{1, 2, \ldots C\}$
- $Vertices$ est l'ensemble de sommets que $Vertices = \{0\} \cup N$

- A est l'ensemble des arêtes, que $A = \{(i,j) \in V^2 : i \neq j\}$
- $dist_{ij}$ *est la distance de l'arête (i,j)* $\in A$
- Q est la capacité maximale de chaque véhicules
- $demande_i$ est la demande de client $i \in N$
- V est le nombre de véhicules
- $Vehicles$ est l'ensemble de véhicules $Vehicles = \{1, 2, \ldots V\}$

On considère que le "Vertice" 0 est le dépôt, et les autres sont les clients

Variables de décision

- x_{ijv}, $i \in Vertices$, $j \in Vertices$, $v \in Vehicles$, si le véhicule v passe de noeud i à noeud j, x_{ijv} =1. Sinon, x_{ijv} = 0
- nbr le nombre de véhicules utilisés
- $Q_{aprestour_v}$ Capacité restante du véhicule v au retour au dépôt
- $u_i, i \in N$, variable d'éliminer les sous tours (Méthode MTZ)

Objectif

Trouver la minimisation de la distance totale parcourue par les véhicules en respectant les contraintes

$$min \sum_{i,j \in A} dist_{ij} x_{ijv}$$

Contraintes

- Il y a pas de chemin de node i vers lui même

$$x_{iiv} = 0 \; \forall i \in Vertices, \forall v \in Vehicles$$

- Au moins 1 véhicules utilisés

$$\sum_{j \in N, v \in Vehicles} x_{0jv} = 1$$

On se rappelle que le "vertice" 0 est le dépôt

- Les véhicules s' il quitte le dépôt, il rentrera au dépôt à la fin

1) Le nombre de fois que chaque véhicules sort du dépôt est soit 0, soit 1

$$\sum_{j \in N} x_{0jv} <= 1 \; \forall v \in Vehicles$$

2) Le nombre de fois que chaque véhicules sort du dépôt est égale au nombre de fois qu'elle rentre au dépôt

$$\sum_{j \in N} x_{0jv} = \sum_{j \in N} x_{j0v} \forall v \in Vehicles$$

- Tous les véhicules quittent la node qu'il a visité

$$\sum_{j \in Vertices} x_{jiv} = \sum_{j \in Vertices} x_{ijv} \; \forall v \in Vehicles, \forall i \in Vertices$$

- Chaque client est passé par exactement 1 fois

$$\sum_{i \in Vertices, v \in Vehicles} x_{ijv} = 1 \forall j \in N$$

On se rappelle que N est l'ensemble de clients

- Pour chaque véhicule utilisé, elle passe les clients en respectant sa capacité.

$$\sum_{i \in Vertices, j \in N} demande_j x_{ijv} \leq Q \; \forall v \in Vehicles$$

- Les contraintes pour éliminer les sous-tours (méthode MTZ)

$$u_j - u_i \geq demande_j - Q(1 - x_{ijv}) \ \forall v \in Vehicles, \forall i \in N, \forall j \in N$$

$$demande_i \leq u_i \leq Q \ \forall i \in N$$

Formulation CPlex

Code CPlex

```
//Données
int nombreNode=...;
int Q=...; //capacité max des véhicules
int V=...; //Nombre de véhicules
float demande[2..nombreNode]=...; //Demandes des clients
float dist[1..nombreNode][1..nombreNode]=...;
dvar boolean x[1..nombreNode][1..nombreNode][1..V];

minimize sum(i in 1..nombreNode,j in 1..nombreNode,v in 1..V ) dist[i][j]*x[i][j][v];
dvar int+ u[2..nombreNode]; // u est un variable pour éliminer les sous tours
//Contrainte
```

```
      subject to {
         // Il y a pas de chemin de node i vers lui même
         forall (i in 1..nombreNode, v in 1..V){
           x[i][i][v] == 0;
         }
         // Au moins 1 véhicule utlisé
        sum(j in 1..nombreNode,v in 1..V) x[1][j][v] >= 1;
         // Quitter dépôt, rentrer le dépôt à la fin
         forall (v in 1..V){
           sum(j in 2..nombreNode) x[j][1][v] == sum (j in 2..nombreNode) x[1][j][v];
           sum(j in 2..nombreNode) x[j][1][v] <= 1;
       }
         // Tous les véhicules Quitte la node qu'il a visiter - nombre de fois un node est quitté = nombre
de fois un node est visité
         forall (j in 1..nombreNode,v in 1..V) {
```

```
    sum(i in 1..nombreNode)x[j][i][v] == sum (i in 1..nombreNode) x[i][j][v];
  }
  //Chaque client est "entré" par 1 fois
  forall (j in 2..nombreNode) {
    sum(v in 1..V, i in 1.. nombreNode) x[i][j][v] == 1;
   }
  //Capacité ne doit pas être dépassé
  forall (v in 1..V)
  {
      sum(i in 1..nombreNode, j in 2..nombreNode) demande[j]*x[i][j][v] <= Q;
  }
  //Eliminer le sous-tours
  forall (i in 2..nombreNode, j in 2..nombreNode, v in 1..V : i!=j)
  {
```

```
        u[j]-u[i] >= demande[j] - Q*(1-x[i][j][v]);
      }
      forall (i in 2..nombreNode){
        demande[i] <= u[i];
        u[i] <= Q;
      }
    }

    execute {
     // Pour chaque véhicule
     for (var v = 1; v <= V; v++) {
       writeln("Chemin du véhicule ", v, ": ");

       // Trouver le chemin parcouru par le véhicule v
```

```
        var i = 1; // Départ du dépôt
        var j = 0;
        var sommeCout = 0;
        var sommeDemande = 0;
        write("Dépôt ", i);
        // Tant que le véhicule n'est pas retourné au dépôt
        while (j != 1) {
          // Trouver le prochain nœud j dans le chemin du véhicule v
          for (var k = 1; k <= nombreNode; k++) {

            if(i == 0){
              j = 1;
              break;
            }
```

```
if (x[i][k][v] == 1) {
  j = k;
  //affichage des noeuds
  if(j == 1) {
    write("-> Dépôt ", j);
    sommeCout += dist[i][j];
  }
  else {
    write(" -> Client ", j, " Distance ", i, " vers ", j, " : ", dist[i][j]);
    sommeCout += dist[i][j];
    sommeDemande += demande[j];
  }
```

```
                break;
            }
        }

        // Mettre à jour le nœud de départ pour la prochaine itération
        i = j;
    }

    //Mettre la capacité restante
    writeln();
    writeln("Capacité restantes : ", Q - sommeDemande);
    writeln("Coût de tournées   : ",sommeCout);
    writeln("-------------------------------------");
}
```

```
        }
```

Jeu de données pour CPlex

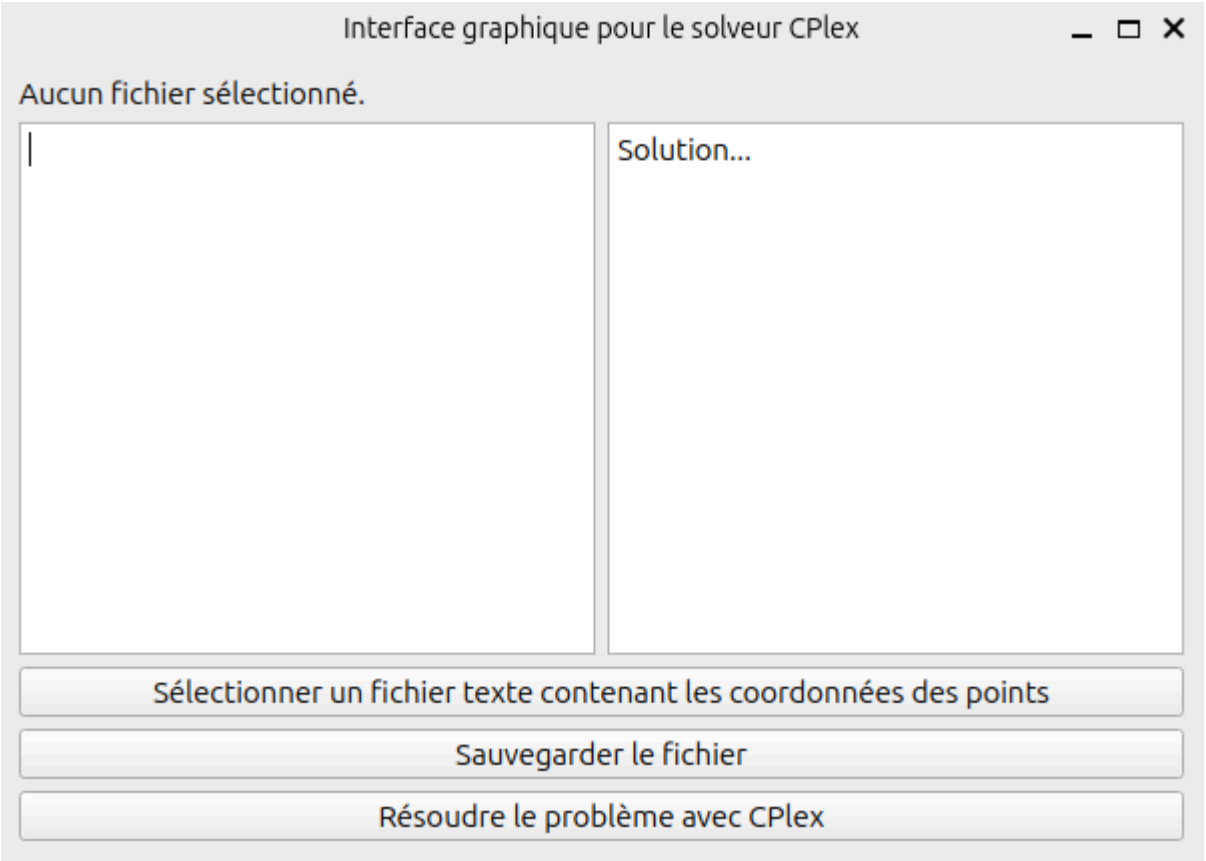
```
nombreNode = 11;

demande = [60,18,26,15,44,32,20,10,27,11];

Q = 100;

V = 4;

dist = [[0 2.7 4.6 2.8 3 3.3 3.1 2.7 5.1 3.9 4.7],

[2.7 0 3.1 0.8 1.8 2.5 4.2 1.4 3.6 2.5 3],

[4.6 3.1 0 3.3 4.4 1.7 6.8 4.1 1.3 1.7 1.4],

[2.8 0.8 3.3 0 1.9 2 4 1.5 3.8 2.8 3.2],

[3 1.8 4.4 1.9 0 3.4 2.6 0.5 4.7 4.7 4.1],

[3.3 2.5 1.7 2 3.4 0 5.8 3 1.8 0.5 2.6],

[3.1 4.2 6.8 4 2.6 5.8 0 3 7.4 6.1 7.6],

[2.7 1.4 4.1 1.5 0.5 3 3 0 4.6 3.7 4.3],

[5.1 3.6 1.3 3.8 4.7 1.8 7.4 4.6 0 1.4 2.8],
```

```
    [3.9 2.5 1.7 2.8 4.7 0.5 6.1 3.7 1.4 0 2.8],

    [4.7 3 1.4 3.2 4.1 2.6 7.6 4.3 2.8 2.8 0]];
```

Application IHM en Python

Pour réaliser l'interface graphique de notre application, nous avons utilisé plusieurs librairies Python. Parmi ces librairies, nous avons utilisé la librairie 'sys' pour manipuler les paramètres liés à l'exécution du script, la librairie 'PyQt5.QtWidgets' pour créer des composants d'interface utilisateur (UI) et des éléments graphiques, la librairie 'PyQt5.QtGui' pour enrichir les capacités graphiques de l'application, et la librairie 'PyQt5.QtCore' pour assurer le bon fonctionnement de l'architecture de l'application PyQt5.

Voici le résultat que nous avons obtenus



Librairie utilisée

- sys

Le module sys est une bibliothèque standard de Python qui fournit un accès à certaines fonctionnalités spécifiques du système. En particulier, l'importation de sys est couramment utilisée pour manipuler des paramètres liés à l'exécution du script, tels que les arguments de la ligne de commande. Il offre également des fonctionnalités pour interagir avec l'environnement d'exécution, comme la gestion des chemins d'accès et la sortie standard.

- PyQt5.QtWidgets

Le module PyQt5.QtWidgets fait partie de la bibliothèque PyQt5, qui est une liaison Python pour la bibliothèque graphique Qt. Ce module fournit des composants d'interface utilisateur (UI) et des éléments graphiques essentiels pour le développement d'applications GUI. En important PyQt5.QtWidgets, vous avez accès à des classes telles que les fenêtres, les boutons, les boîtes de dialogue, etc., facilitant ainsi la création d'interfaces utilisateur interactives et conviviales.

- PyQt5.QtGui

Le module PyQt5.QtGui offre des fonctionnalités graphiques supplémentaires pour le développement d'interfaces utilisateur. Il inclut des éléments tels que les polices, les couleurs, les images, et d'autres outils graphiques. En important ce module, vous enrichissez les capacités graphiques de votre application PyQt5, permettant la manipulation avancée des composants visuels et des éléments esthétiques.

- PyQt5.QtCore

Le module PyQt5.QtCore est un composant fondamental de PyQt5, fournissant des fonctionnalités de base pour le développement d'applications. Il offre des classes liées à la gestion des événements, aux signaux et aux slots, ainsi qu'à la manipulation du temps et des threads. Importer PyQt5.QtCore est crucial pour assurer le bon fonctionnement de l'architecture de l'application PyQt5, en facilitant la gestion des interactions utilisateur et des tâches concurrentes.

Docplex - API Python avec CPLEX

Pour résoudre le problème d'optimisation à l'aide de CPLEX, nous avons intégré l'API CPLEX dans notre programme Python. L'utilisation de l'API a été réalisée en respectant les étapes suivantes :

1. Définition des Contraintes et des Variables

Les contraintes et les variables du modèle d'optimisation ont été spécifiées en utilisant les fonctionnalités fournies par l'API DOCplex. Les contraintes et variables sont déclarées directement dans le code Python, en utilisant les méthodes et les classes fournies par l'API pour construire le modèle de manière intuitive et efficace.

```
from docplex.mp.model import Model


# Création du modèle
model = Model('NomDuModele')


# Déclaration des variables
x = model.integer_var_dict( ... )
# Déclaration des contraintes
model.add_constraint( ... )
```

2. Importation des Données

Les données nécessaires pour résoudre le problème sont importées à partir d'un fichier généré par une application externe. Ces données sont ensuite traitées à l'aide d'un programme spécifique qui convertit une matrice de coordonnées en une matrice de distances. Les données résultantes sont ensuite intégrées dans le modèle CPLEX pour être utilisées lors de la résolution du problème.

```
# Importation des données à partir d'un fichier externe
donnees = importer_donnees('chemin/vers/fichier.csv')
```

```
    # Traitement des données pour obtenir la matrice de distances
    matrice_distances = convertir_en_matrice_distances(donnees)

    # Utilisation des données dans le modèle
    model.add_constraints( ... )
```

3. Résolution du Problème avec CPLEX

Une fois que le modèle a été construit et que les données ont été importées, nous utilisons l'API CPLEX pour résoudre le problème d'optimisation. La résolution se fait en appelant la méthode solve() du modèle.

```
    # Résolution du modèle
    solution = model.solve()

    # Obtention des résultats
    if solution:
        # Traitement et affichage des résultats
        afficher_resultats(solution)
    else:
        print("Aucune solution trouvée.")
```

4. Génération des Solutions et du Graphe

Les solutions obtenues après la résolution du modèle sont exploitées pour générer les résultats finaux. L'API DOCplex facilite la récupération des valeurs des variables, ce qui permet de créer le graphe de manière dynamique.

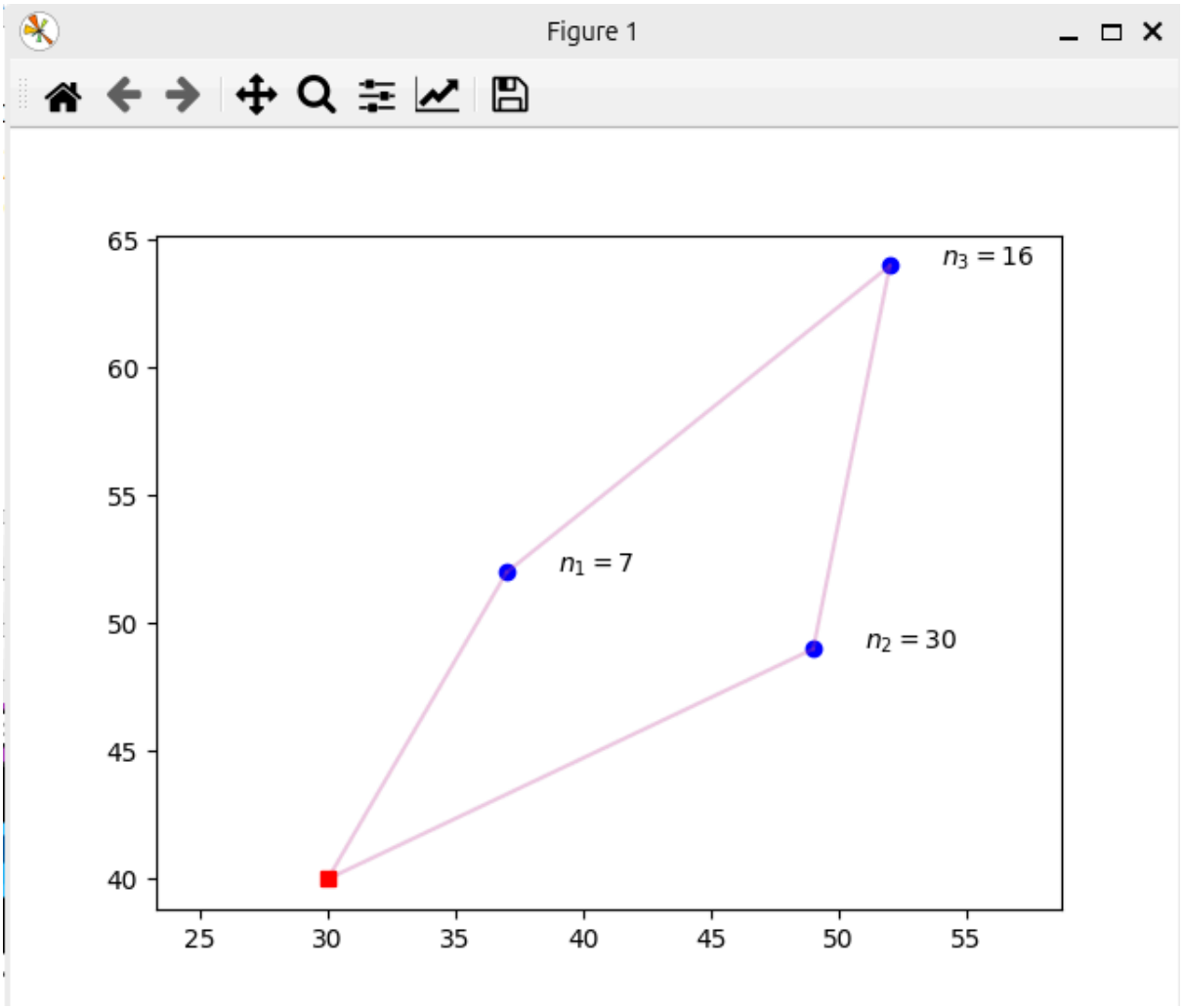
```
# Récupération des valeurs des variables
variables_solution = solution.get_all_values()

# Génération du graphe
generer_graphe(variables_solution)
```

En suivant cette approche, nous avons pu intégrer l'API CPLEX de manière transparente dans notre programme Python, simplifiant ainsi le processus de modélisation, de résolution et de visualisation des résultats.

Matplotlib - Visualiser avec Python

Matplotlib est une librairie mathématique de python qui permet de visualiser très facilement des graphes. C'est donc en toute logique que nous avons fait le choix de l'utiliser afin de visualiser le dépôt, les différents clients ainsi que le trajet des différents véhicules.

Résultat d'un graphe à l'aide de Matplotlib



Il est bon de noter que le dépôt est représenté par un carré rouge, les clients par un rond bleu et le trajet des véhicules avec des arcs qui forme un circuit hamiltonien de différente couleur en fonction du camion.

Nous n'avons pas orienté le graphe parce qu'on ne jugeait pas cela utile mais rétrospectivement il aurait fallu l'orienté.