UNIVERSITÉ LE HAVRE NORMANDIE

PROBLÈME DE TOURNÉES DE VÉHICULES

RAPPORT DE PROJET

2024

Présenté par
Antoine Dardanne
Sébastien Chauvière
Théo Cudeville
Noé Florence

Dans le cadre d'un projet universitaire, il nous est demandé de réaliser une application permettant de résoudre le problème VRP.

Sommaire

Introduction

Dans le cadre d'un projet académique de développement d'application dans le cadre d'une situation d'apprentissage à l'Université du Havre, nous sommes amenés à réaliser une application dont le but est de résoudre un problème de tournées de véhicules.

L'objectif principal de ce projet est de fournir un outil afin de minimiser le déplacement des camions lors de leur tournée de distribution de produits et d'afficher ces résultats sous forme d'un graphique pour représenter les différents chemins de la tournée. L'application sert à faciliter la planification et l'organisation des tournées.

Ce document est un compte-rendu de notre conception de notre application et des différents choix pour les technologies que nous avons décidé d'utiliser.

I. Description du projet

I. I. Contexte et explications

Le problème de tournées de véhicules (aussi appelé VRP pour Vehicule Routing Problem) est une classe de problèmes de recherche opérationnelle et d'optimisation combinatoire. Il s'agit de déterminer les tournées d'une flotte de véhicules afin de livrer une liste de clients, ou de visites. Le but est de minimiser le coût de livraison des biens. Ce problème est une extension classique du problème du voyageur de commerce.

Pour ce projet, nous avons pour objectif de réaliser le modèle mathématique du problème afin de pouvoir l'utiliser à bon escient dans notre code CPLEX afin de pouvoir résoudre le problème de tournées de véhicules peu importe le jeu de données.

Ensuite, nous avons pour objectif de créer une application utilisant l'API Python de CPLEX. Nous devrons réaliser une interface graphique permettant de sélectionner un jeu de données que nous voulons tester, d'afficher les solutions du jeu de données et enfin, de pouvoir tracer les tournées sur un graphique grâce à Python.

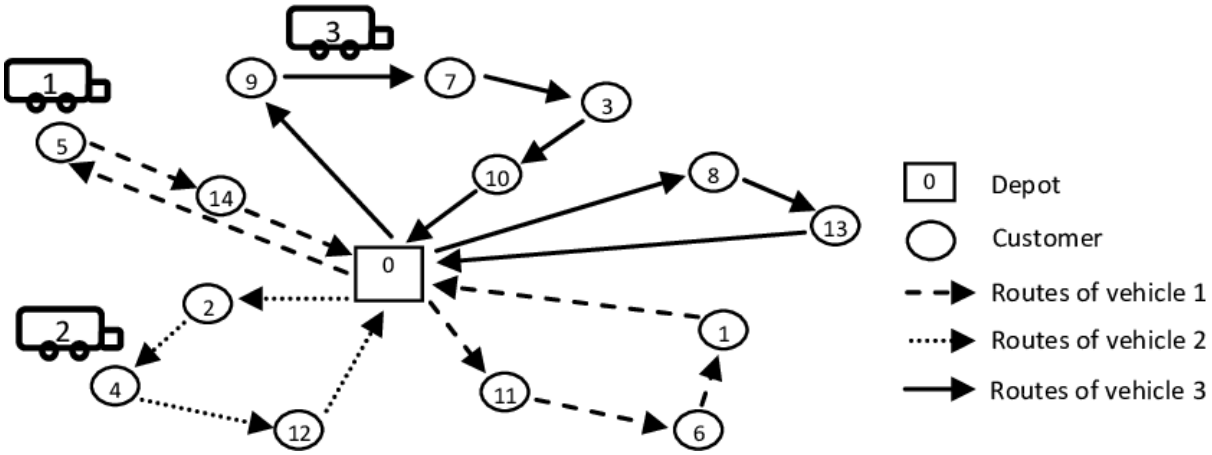


Figure n°1 : Schéma du problème de tournées

I. II. Exigences fonctionnelles

I. II. I. Modèle Mathématique

- Formulation du Problème : Définir clairement le problème de tournées de véhicules et élaborer un modèle mathématique précis, incluant la fonction objectif et toutes les contraintes nécessaires.
- Adaptabilité des Données : Assurer que le modèle est capable de prendre en charge différents jeux de données sans nécessiter des modifications majeures.

I. II. II. Intégration avec CPLEX

- Utilisation de l'API Python : Implémenter l'intégration entre le modèle mathématique et CPLEX en utilisant l'API Python de CPLEX.
- Résolution du problème : Utiliser CPLEX pour résoudre le problème de tournées de véhicules basé sur le modèle mathématique.

I. II. III. Interface Graphique

- Sélection des Jeux de Données : Permettre à l'utilisateur de choisir un jeu de données parmi plusieurs options disponibles.
- Affichage des solutions : Afficher les solutions du problème résolu, y compris les itinéraires des véhicules et les coûts associés.
- Tracer des Tournées sur un graphique : Utiliser des bibliothèques graphiques (telles que Matplotlib) pour tracer les tournées des véhicules sur un graphique interactif.

I. II. IV. Gestion des Jeux de Données

- Ajout/Suppression de Jeux de Données : Permettre à l'utilisateur d'ajouter de nouveaux jeux de données ou de supprimer des jeux existants pour tester différentes configurations.
- Validation des Données : Assurer que les données fournies sont valides pour la résolution du problème.

II. Analyse et conception

II. I. Conception fonctionnelle

II. I. I. Modélisation mathématique

Fonction objectif

$$Z = \sum_{i \in I} \sum_{j \in J} \sum_{k \in K} c_{ij} * x_{ijk}$$

➪ La fonction objectif Z est définie comme la somme pondérée des coûts associés à chaque arc du graphe du problème. Voici les termes associés :

- i représente le premier client chez lequel le véhicule est initialement situé.
- j représente le deuxième client chez lequel le véhicule doit se rendre.
- k représente le véhicule qui doit se déplacer.
- c_{ij} représente le coût associé au déplacement du client i au client j.
- x_{ijk} est une variable binaire qui indique si le véhicule k se déplace de i à j. Si $x_{ijk} = 1$, cela signifie que le véhicule k emprunte l'arc (i, j); sinon $x_{ijk} = 0$.

Contrainte 1

$$\sum_{i \in I} \sum_{k \in K} x_{ijk} = 1 \qquad \forall j \in I/\{1\}$$

- i représente le premier client chez lequel le véhicule est initialement situé.
- j représente le deuxième client chez lequel le véhicule doit se rendre.
- k représente le véhicule qui doit se déplacer.

Cette contrainte dit que, pour chaque client j (à l'exception du premier client, car il s'agit du dépôt), la somme des variables binaires x_{ijk} sur tous les clients de départ i et tous les véhicules k doit être égale à 1. Cela signifie qu'un seul véhicule doit être choisi pour aller du client initial i au client j. Ainsi, chaque client doit être visité exactement une seule fois par un seul véhicule.

Contrainte 2

$$\sum_{j \in J} x_{ijk} = \sum_{j \in J} x_{jik} \qquad \forall i \in I : i > 1 \qquad \forall k \in K$$

- i représente le premier client chez lequel le véhicule est initialement situé.
- j représente le deuxième client chez lequel le véhicule doit se rendre.
- k représente le véhicule qui doit se déplacer.

Cette contrainte dit que, pour chaque client j et chaque véhicule k, la sommes des variables binaires x_{ijk} sur tous les clients de départ i doit être égale à la somme des variables binaires x_{jik} sur tous les clients d'arrivée i. Ainsi, la somme des arcs entrants vers le client j pour le véhicule k doit être égale à la somme des arcs sortants du client j pour le même véhicule. Cette contrainte assure la continuité des tournées pour chaque véhicule, garantissant qu'un véhicule entrant dans un client doit en ressortir, évitant alors la création de sous-tours, afin de former des chemins complets sans cycle.

Contrainte 3

$$\sum_{j \in I/\{1\}} x_{1jk} = 1 \qquad \forall k \in K$$

- j représente le client d'arrivée.
- k représente le véhicule qui doit se déplacer.

Cette contrainte dit que, pour chaque véhicule k, la somme des variables binaires x_{1jk} sur tous les clients d'arrivée j (à l'exception du premier car il s'agit du dépôt) doit être égale à 1. Ainsi, chaque véhicule doit commencer sa tournée au dépôt.

Contrainte 4

$$\sum_{i \in I} \sum_{j \in I/\{1\}} c_i x_{jik} \leq q_k \qquad \forall k \in K$$

- i représente le premier client chez lequel le véhicule est initialement situé.
- j représente le deuxième client chez lequel le véhicule doit se rendre.
- k représente le véhicule qui doit se déplacer.
- c_i représente la demande du client i.
- q_k représente la capacité maximale du véhicule k.

Cette contrainte dit que la sommes des demandes pondérées des clients (représentée par d_i) visités par un véhicule donné ne dépasse pas la capacité maximale de ce véhicule q_k. Ainsi, elle s'assure que la capacité du véhicule n'est pas dépassée lors de la planification de la tournée afin que les véhicules respectent leurs limites de capacité.

Contrainte 5

$$u_{ik} - u_{jk} + (I - K) * x_{ijk} \leq I - K - 1 \qquad \forall i, j \in I, j > 1 \qquad \forall k \in K \quad u_{ik} \geq 0$$

- i et j représentent les clients de départ et d'arrivée.
- k représente le véhicule qui doit se déplacer.
- u_{ik} est la variable de coût associée au client i pour le véhicule k.
- x_{ij} est la variable binaire indiquant si le véhicule k va du client i au client j.
- I est la taille de l'ensemble des clients.
- K est le nombre de véhicules.

Cette contrainte d'inégalité est une version de la contrainte Miller-Tucker-Zemlin (MTZ) utilisée pour éliminer les sous-tours. Elle assure qu'il n'y a pas de cycles sous-tours dans la solution, garantissant alors la formation de tournées cohérentes. Les variables u_{ik} sont des variables auxiliaires qui représentent les coûts cumulatifs associés au passage par le client i pour le véhicule k.

La partie $u_{ik} - u_{jk} + (I - K) * x_{ijk} \leq I - K - 1$ exprime la relation de coût entre les clients i et j pour le véhicule k et pénalise la présence d'arcs entre les clients i et j en ajoutant le terme $(I - K) * x_{ijk}$. Les variables $u_{ik} \geq 0$ garantissent que les coûts cumulatifs soient non négatifs. Cette contrainte vise alors à éviter la formation de sous-tours dans la solution.

II. I. II. Exigences de l’interface

Interface utilisateur intuitive

- Une interface utilisateur intuitive qui permet aux utilisateurs de comprendre rapidement comment utiliser l'application.

Sélection des Jeux de Données

- Des éléments d'interface clairs, tels qu’une liste déroulante ou un bouton, pour permettre à l'utilisateur de sélectionner facilement le jeu de données à tester.

Affichage des Solutions

- Présentation claire des résultats de la résolution du problème, y compris les itinéraires des véhicules.
- Des visualisations graphiques, comme un graphique, pour rendre les informations facilement compréhensibles.

Tracé des Tournées sur un Graphique

- Utilisation d’une bibliothèque graphique, telle que Matplotlib ou Tkinter, pour tracer les tournées des véhicules sur un graphique.
- Des itinéraires clairement visibles et différenciés pour faciliter la compréhension.

Adaptabilité

- Une interface adaptative, de sorte qu'elle fonctionne correctement sur différentes tailles d'écrans.

III. Réalisation technique

III. I. Outils et technologies utilisés

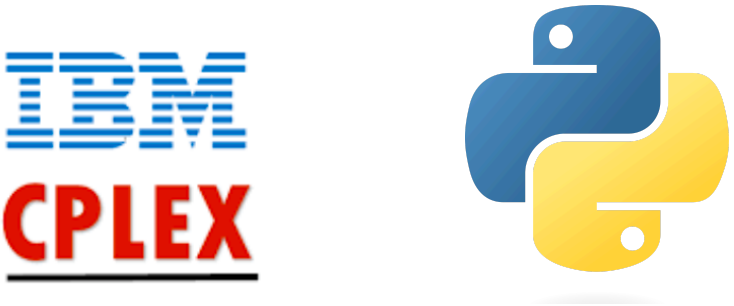


Figure n°2 : Logos de l'ensemble des technologies utilisées
(de gauche vers la droite : CPLEX par IBM & Python)

Pour réaliser ce projet, nous avons utilisé l'outil CPLEX qui est un puissant solveur d'optimisation développé par IBM. Il est largement utilisé pour résoudre des problèmes complexes d'optimisation linéaire, entière mixte, quadratique et autres. CPLEX est souvent utilisé pour résoudre des problèmes difficiles tels que le problème de tournées de véhicules (VRP).

Afin d'afficher une interface pour sélectionner les jeux de données que nous voulons résoudre, nous utilisons Python qui permettra de résoudre les problèmes et d'ainsi mieux comprendre la solution. Nous afficherons donc un graphique montrant le dépôt, les villes et les trajets empruntés par les différents véhicules, ainsi qu'un tableau, à côté, montrant les données envoyées par le solveur.

Pour faire cela, nous utilisons différentes API de Python telles que Tkinter et Cplex. Tkinter est une API permettant de réaliser des interfaces graphiques (IHM) mais aussi des graphiques (Canvas) en python.

L'API Cplex de python permet, quant à elle, de résoudre des problèmes d'optimisation de décisions soit par l'écriture des fonctions et contraintes en python, soit par l'utilisation d'un fichier lp exporter à partir de Cplex studio d'IBM.

III. II. Résultats

III. II. I. Code du CPLEX - Résolution du problème

```
//definire le nombre de nodes
int nbclients=51;
int nbvehicles=4;

// définir les parametres
// définir index
range clients=1..nbclients; //I l'ensemble des villes
range vehicles=1..nbvehicles; //I l'ensemble des vehicules

//c est le couts ou bien la distance entre deux villes; créer des nombres aléatoire
float d[i in clients][j in clients]=...;
float c[i in clients]=...;
float q[k in vehicles]=160;



//Variables de décision
dvar boolean x[clients][clients][vehicles]; //x est un variable boolean égale 1 si il visite de la ville i à j sinon 0
dvar int+ u[clients][vehicles]; // u est un variable pour éliminer les sous tours
//dvar int+ v[0][nbVehicles];

minimize sum(i,j in clients , k in vehicles)  d[i][j]*x[i][j][k];


//objective function
subject
to{
//constraints

//chaque nœud est entré une fois par un vehicule.
c1:
forall (j in clients: j >1) sum(i in clients) sum(k in vehicles) x[i][j][k]==1;


c2:
forall (i in clients: i > 1, k in vehicles) sum(j in clients)  x[i][j][k] == sum(j in clients)  x[j][i][k];

//Chaque véhicule quitte le dépôt et y retourne
c3:
forall(k in vehicles) sum (j in clients)  x[1][j][k]==1;

c4:
//Contrainte pour ne dépasser la capacité:
forall (k in vehicles) sum(i in clients) sum(j in clients ) c[i] * x[i][j][k] <= q[k];

//Contraintes pour éliminer les sous tours
c5:
forall (i,j in clients:  j > 1 , k in vehicles) u[i][k] - u[j][k] + (nbclients-nbvehicles) * x[i][j][k] <= (nbclients-nbvehicles-1);

}

execute {
    // Affichage des résultats
writeln("Déroulement des tournées : \n");

// Boucle sur les variables de décision x
  for (var k in vehicles) {
      var currentCity = 1; // Départ de la première ville après le dépôt
      writeln("Le vehicule ", k, " part du depot ");

      var remainingCapacity = q[k];
      var deliveredToDepot = false;
      var deliveredAnything = false; // Variable pour vérifier si quelque chose a été livré

      while (!deliveredToDepot) {
          for (var j in clients) {
              if (x[currentCity][j][k].solutionValue > 0.5) {
                  var nextCity = j;
                  if (nextCity == 1) {
                      deliveredToDepot = true;
                      if (!deliveredAnything) { // Si rien n'a été livré
                          writeln(" Le véhicule ", k ," ne livre rien et retourne au depot \n");
                      } else {
                          writeln(" Le véhicule ", k ," retourne au depot \n");
                      }
                      break;
                  } else {
                      deliveredAnything = true; // Indique qu'une livraison a été effectuée
                      writeln("Le véhicule ", k, " part du client ", currentCity-1, " vers le client ", nextCity-1);
                      currentCity = nextCity;
                      remainingCapacity -= c[currentCity];
                      writeln("    Quantite restante dans le vehicule ", k, ": ", remainingCapacity);
                      break;
                  }
              }
          }
      }
  }
}
```


Figure n°3 : Code CPLEX permettant de résoudre le problème VRP

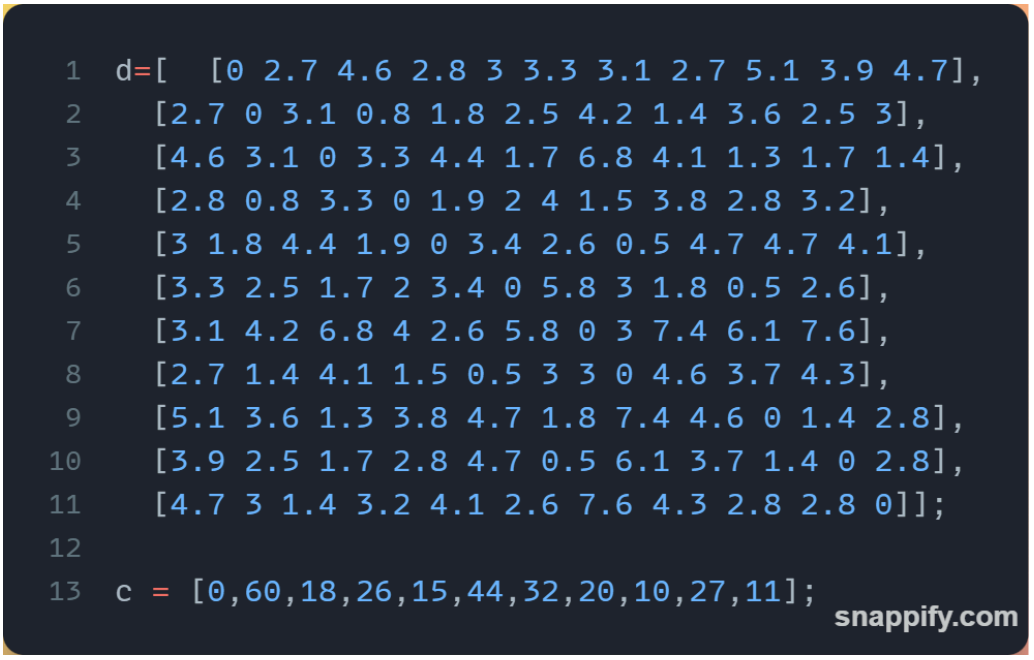


Figure n°4 : Fichier data de l'exemple

Notre code CPLEX est composé de la manière suivante, au début du code, nous définissons le nombre de véhicules et le nombre de clients. Ensuite, nous définissons l'ensemble des clients et des véhicules en fonction du nombre entré. Puis, nous récupérons les données du *.dat*, les distances entre les clients ainsi que les demandes de chacun. Par la suite, nous avons mis les différentes contraintes liées au problème VRP. Enfin, nous avons écrit notre *execute*.

III. II. II. Résultats du CPLEX

```
// solution (optimal) with objective 31.3
Déroulement des tournées :
Le vehicule 1 part du depot
Le véhicule 1 part du client 0 vers le client 6
    Quantite restante dans le vehicule 1: 68
Le véhicule 1 part du client 6 vers le client 4
    Quantite restante dans le vehicule 1: 53
Le véhicule 1 part du client 4 vers le client 7
    Quantite restante dans le vehicule 1: 33
 Le véhicule 1 retourne au depot

Le vehicule 2 part du depot
Le véhicule 2 part du client 0 vers le client 3
    Quantite restante dans le vehicule 2: 74
Le véhicule 2 part du client 3 vers le client 1
    Quantite restante dans le vehicule 2: 14
Le véhicule 2 part du client 1 vers le client 10
    Quantite restante dans le vehicule 2: 3
 Le véhicule 2 retourne au depot

Le vehicule 3 part du depot
Le véhicule 3 part du client 0 vers le client 2
    Quantite restante dans le vehicule 3: 82
Le véhicule 3 part du client 2 vers le client 8
    Quantite restante dans le vehicule 3: 72
Le véhicule 3 part du client 8 vers le client 9
    Quantite restante dans le vehicule 3: 45
Le véhicule 3 part du client 9 vers le client 5
    Quantite restante dans le vehicule 3: 1
 Le véhicule 3 retourne au depot

Le vehicule 4 part du depot
 Le véhicule 4 ne livre rien et retourne au depot
```

Figure n°5 : Résultat de l'exemple du problème VRP du sujet avec le solveur CPLEX

Lors de la création du CPLEX, nous avons créé un *execute* afin d'afficher correctement les données pour que nous puissions suffisamment bien les interpréter.

Nous avons donc décidé d'afficher les tournées de chaque véhicule, un par un, de son départ du dépôt à son retour. Nous avons également écrit une condition qui dit que si le véhicule ne part pas, il ne livre rien et reste au dépôt.

III. II. III. Résultats de l'exemple de 50 clients

Exemple avec 50 clients

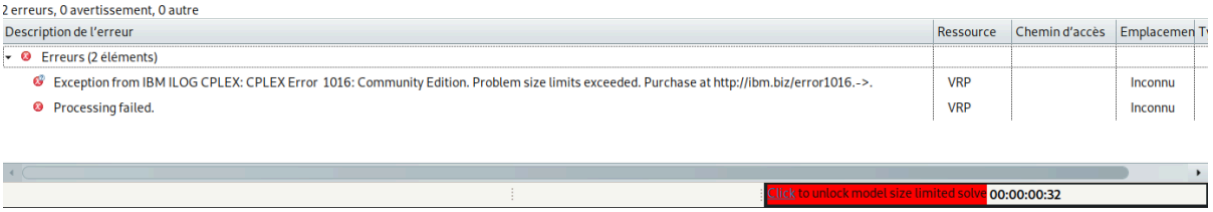


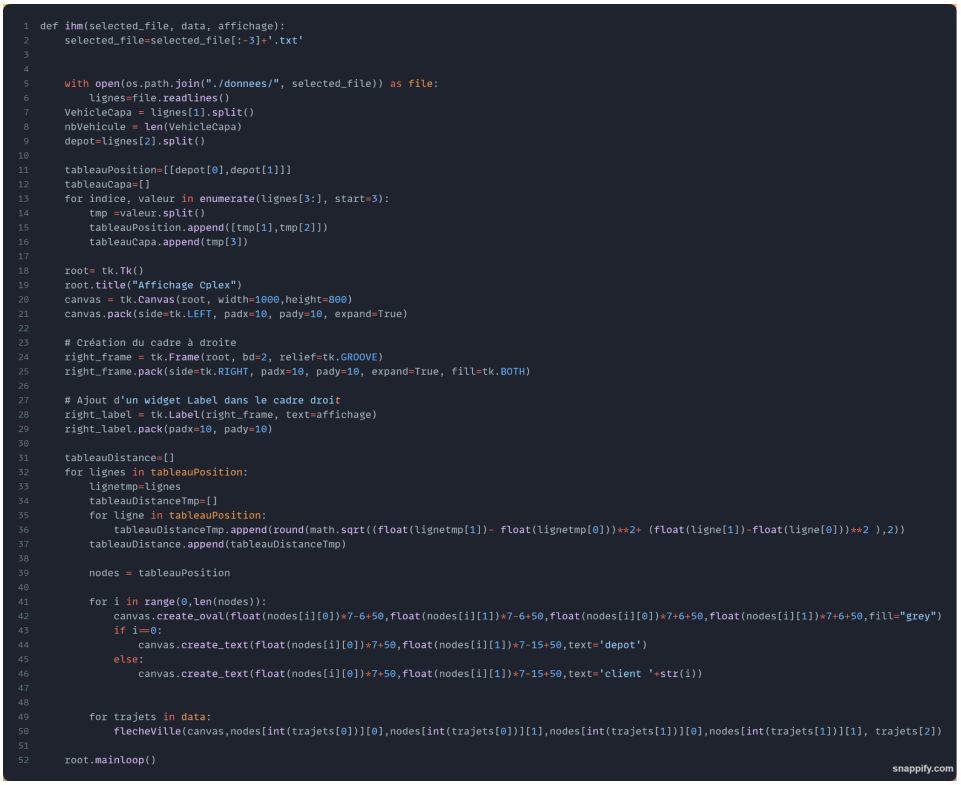
Figure n°6 : Résultat du problème VRP avec un exemple de 50 clients sur le solveur CPLEX

Dans cet exemple, on peut voir que nous obtenons une erreur à l'affichage. En effet, nous avons voulu interroger le code sur un problème proposé, composé de 50 clients, et nous avons obtenu cette erreur.

Celle-ci survient car nous ne possédons en réalité que la version gratuite du logiciel CPLEX qui limite le nombre de variables d'un programme, ce qui nous permet donc de trouver des solutions seulement à des problèmes moins massifs.

III. II. IV. Code de l'application Python

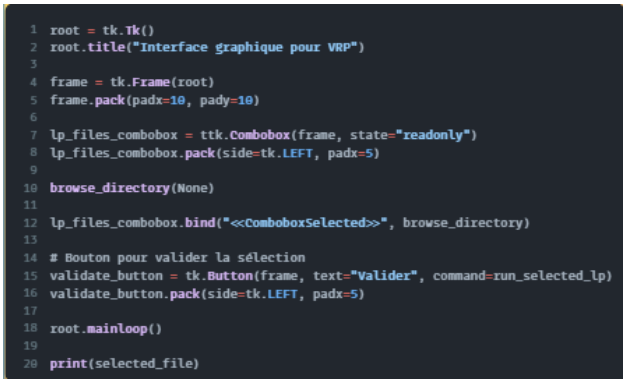
Fonctions permettant d'afficher les interfaces graphiques



```python
def ihm(selected_file, data, affichage):
    selected_file=selected_file[:-3]+'.txt'


    with open(os.path.join("./donnees/", selected_file)) as file:
        lignes=file.readlines()
    VehicleCapa = lignes[1].split()
    nbVehicule = len(VehicleCapa)
    depot=lignes[2].split()

    tableauPosition=[[depot[0],depot[1]]]
    tableauCapa=[]
    for indice, valeur in enumerate(lignes[3:], start=3):
        tmp =valeur.split()
        tableauPosition.append([tmp[1],tmp[2]])
        tableauCapa.append(tmp[3])

    root= tk.Tk()
    root.title("Affichage Cplex")
    canvas = tk.Canvas(root, width=1000,height=800)
    canvas.pack(side=tk.LEFT, padx=10, pady=10, expand=True)

    # Création du cadre à droite
    right_frame = tk.Frame(root, bd=2, relief=tk.GROOVE)
    right_frame.pack(side=tk.RIGHT, padx=10, pady=10, expand=True, fill=tk.BOTH)

    # Ajout d'un widget Label dans le cadre droit
    right_label = tk.Label(right_frame, text=affichage)
    right_label.pack(padx=10, pady=10)

    tableauDistance=[]
    for lignes in tableauPosition:
        lignetmp=lignes
        tableauDistanceTmp=[]
        for ligne in tableauPosition:
            tableauDistanceTmp.append(round(math.sqrt((float(lignetmp[1])- float(lignetmp[0]))**2+ (float(ligne[1])-float(ligne[0]))**2 ),2))
        tableauDistance.append(tableauDistanceTmp)

        nodes = tableauPosition

        for i in range(0,len(nodes)):
            canvas.create_oval(float(nodes[i][0])*7-6+50,float(nodes[i][1])*7-6+50,float(nodes[i][0])*7+6+50,float(nodes[i][1])*7+6+50,fill="grey")
            if i==0:
                canvas.create_text(float(nodes[i][0])*7+50,float(nodes[i][1])*7-15+50,text='depot')
            else:
                canvas.create_text(float(nodes[i][0])*7+50,float(nodes[i][1])*7-15+50,text='client '+str(i))


        for trajets in data:
            flecheVille(canvas,nodes[int(trajets[0])][0],nodes[int(trajets[0])][1],nodes[int(trajets[1])][0],nodes[int(trajets[1])][1], trajets[2])

    root.mainloop()
```


Figure n°7 : Fonction Python de l'affichage de la fenêtre graphique

Cette fonction permet d'afficher le graphique représentant les clients et les trajets en fonction des données renvoyées par cplex. on utilise des fonctions de tkinter pour le réaliser. Celle-ci permet également d'afficher le dérouler des tournées textuellement sur la droite de l'application.



```python
root = tk.Tk()
root.title("Interface graphique pour VRP")

frame = tk.Frame(root)
frame.pack(padx=10, pady=10)

lp_files_combobox = ttk.Combobox(frame, state="readonly")
lp_files_combobox.pack(side=tk.LEFT, padx=5)

browse_directory(None)

lp_files_combobox.bind("<<ComboboxSelected>>", browse_directory)

# Bouton pour valider la sélection
validate_button = tk.Button(frame, text="Valider", command=run_selected_lp)
validate_button.pack(side=tk.LEFT, padx=5)

root.mainloop()

print(selected_file)
```

Figure n°8 : Fonction Python permettant de créer la fenêtre de sélection de fichiers

Cette fonction permet d'ouvrir la première fenêtre de l'application qui permet de sélectionner un fichier *lp* contenant un jeu de données afin d'en afficher graphiquement et textuellement les tournées.

Fonctions permettant de résoudre un problème CPLEX

```python
# Fonction pour résoudre le fichier LP sélectionné
def run_selected_lp():
    selected_file = lp_files_combobox.get()
    if selected_file:
        try:
            print(selected_file)
            # Utilisation de l'API CPLEX pour lire et résoudre le fichier LP sélectionné
            c = cplex.Cplex()
            file_path = os.path.join("./donnees", selected_file)  # Chemin complet du fichier LP
            c.read(file_path)
            c.solve()
            affichage=""

            print("Solution pour ", selected_file[:-3], "\n")
            affichage+=str("Solution pour ")+ str(selected_file[:-3] + "\n\n")
            for i, value in enumerate(c.solution.get_values()):
                if value != 0:
                    names = c.variables.get_names(i).replace("#", " ");
                    tmp = names.split(" ")
                    tab.append(tmp)
                else:
                    continue

            trajets = [item[1:] for item in tab if item[0] == 'x']

            affichage+=affichageConsole(trajets)

            print("Valeur de la fonction objectif", round(c.solution.get_objective_value(),2))
            affichage+= str("Valeur de la fonction objectif : " + str(round(c.solution.get_objective_value(),2)))

            ihm(selected_file, trajets, affichage)

        except cplex.exceptions.CplexError as e:
            print("Aucunes solution trouvé", e)
    else:
        print("Aucun fichier sélectionné.")

# Fonction pour remplir la liste déroulante avec les fichiers LP du dossier 'donnees'
def browse_directory(event):
    folder_path = "./donnees"  # Chemin du dossier "donnees"
    if folder_path:
        lp_files = [f for f in os.listdir(folder_path) if f.endswith(".lp")]
        lp_files_combobox['values'] = lp_files
```

Figure n°9 : Fonction Python de récupération des résultats CPLEX

Cette fonction permet de récupérer les données du résultat renvoyé par CPLEX en utilisant la librairie cplex. Elle affiche ensuite l'ihm et textuellement le déroulé des tournées dans la console.

Fonctions permettant l'affichage console

```python
def trouver_trajets(depart, tableau_trie, numVoiture):
    text = ""
    for trajet in tableau_trie:
        if trajet[0] == depart:
            if trajet[0] == 0 and trajet[1] == 0:
                print("La voiture ", numVoiture, " reste au dépôt", "\n")
                text += "La voiture " + str (numVoiture) + " reste au dépôt" + "\n"
            elif trajet[1] == 0:
                print("La voiture ", numVoiture, " part du client ", trajet[0], " et retourne au dépot", "\n")
                text += "La voiture " + str (numVoiture) + " part du client " + str (trajet[0]) + " et retourne au dépot" + "\n"
            elif trajet[0] == 0:
                print("La voiture ", numVoiture, " commence son trajet du dépot et va au client ", trajet[1])
                text += "La voiture " + str (numVoiture) + " commence son trajet du dépot et va au client " + str (trajet[1]) + "\n"
            else:
                print("La voiture ", numVoiture, " va du client ", trajet[0]," au client ",trajet[1])
                text += "La voiture " + str (numVoiture) + " va du client " + str (trajet[0]) + " au client " + str (trajet[1]) + "\n"
            tableau_trie.remove(trajet)
            text += trouver_trajets(trajet[1], tableau_trie, numVoiture)
            break
    return text
```

Figure n°10 : Fonction Python permettant de trouver le trajet d'un véhicule

```python
# Fonction pour afficher les trajets dans la console
def affichageConsole(trajets):

    # Triez les trajets en fonction du numéro du véhicule
    trajets_tries = sorted(trajets, key=lambda x: int(x[2]))

    # Dictionnaire pour stocker les trajets par véhicule
    trajets_par_vehicule = {}

    # Remplir le dictionnaire avec les trajets pour chaque véhicule
    for trajet in trajets_tries:
        vehicule = trajet[2]
        if vehicule not in trajets_par_vehicule:
            trajets_par_vehicule[vehicule] = []
        trajets_par_vehicule[vehicule].append(trajet)


    tab_trajet_trie=[]

    # Afficher les trajets pour chaque véhicule
    for vehicule, trajets_vehicule in trajets_par_vehicule.items():
        tabtemp=[]
        for trajet in trajets_vehicule:
            tabtemp.append([trajet[0], trajet[1]])
        tab_trajet_trie.append(tabtemp)

    cpt=1

    text=""
    for trajets in tab_trajet_trie:
        trajets = [[int(element) for element in sublist] for sublist in trajets]
        tableau_trie = sorted(trajets, key=lambda x: int(x[0]))

        # Appel de la fonction pour trouver et afficher les trajets
        text += str (trouver_trajets(0, tableau_trie, cpt)) + "\n"
        cpt+=1
    return text
```

Figure n°11 : Fonction Python permettant d'afficher le trajet d'un véhicule

Ces deux fonctions permettent d'afficher dans les console et d'enregistrer dans un string les trajets réalisés par les véhicules triés dans l'ordre des véhicules.

III. II. V. Interface Python

Sélection du fichier .lp à résoudre

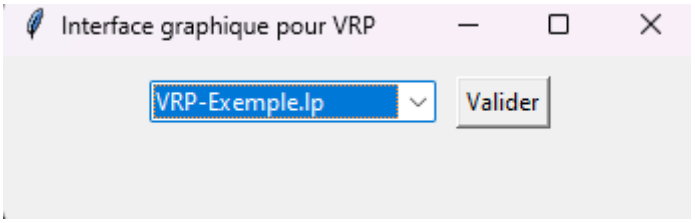


Figure n°12 : Fenêtre de l'application permettant de choisir un jeu de données

Notre application est faite d'une interface très simple comprenant une liste déroulante permettant de sélectionner des fichiers de jeux de données en .lp et d'un bouton valider permettant par la suite d'afficher la seconde interface.

Affichage de la tournée

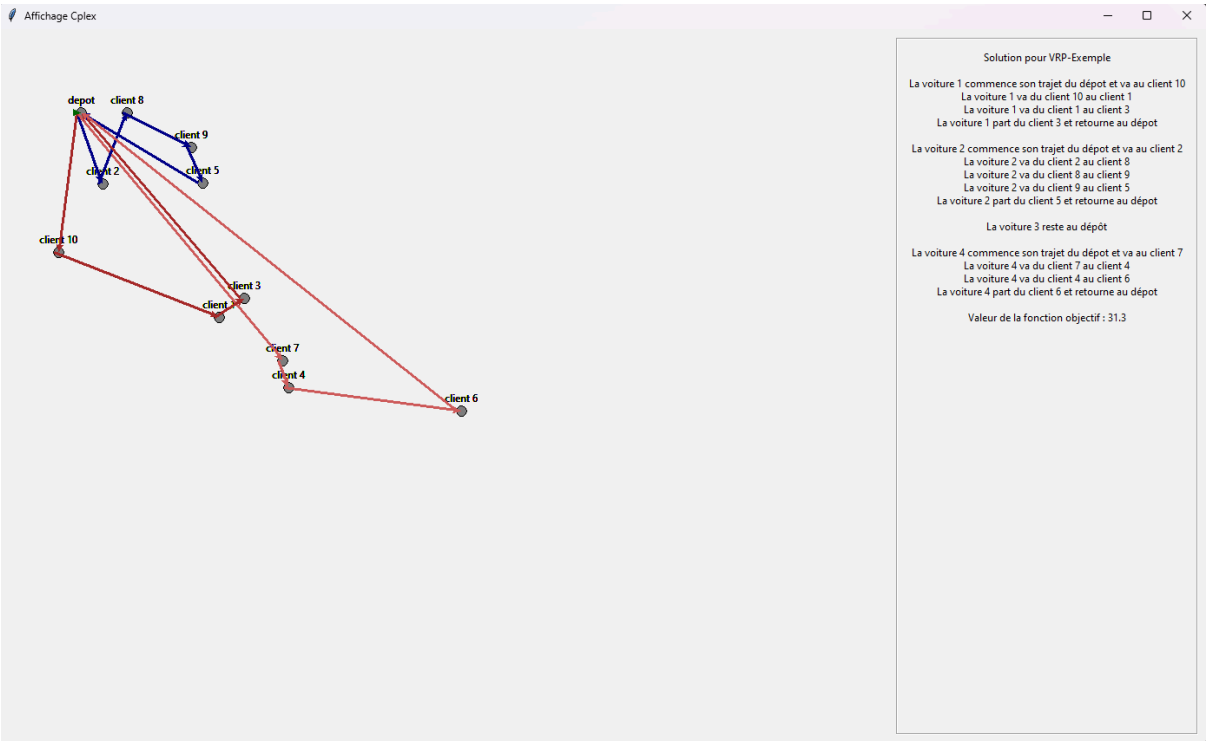


Figure n°13 : Seconde interface graphique permettant de visualiser les tournées

Sur cette seconde interface, nous trouvons le graphique des tournées avec des flèches ayant une couleur par véhicule, ainsi que le déroulement des tournées de manière textuelle.

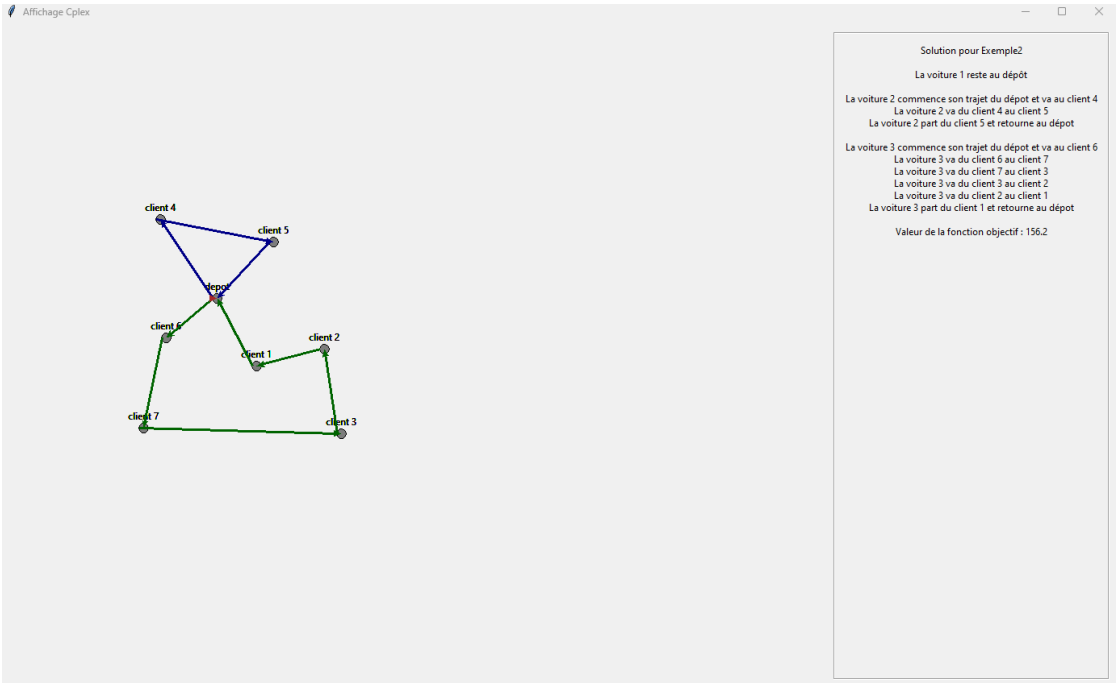


Figure n°14 : Seconde interface graphique permettant de visualiser les tournées

III. II. VI. Convertisseur de jeux de données

Pour convertir les jeux de données présents sur le site : mistic.heig-vd.ch qui était constitué d'une matrice de position pour chaque ville en jeux de données constitué d'une matrice de distance utilisable par cplex, nous avons créé un outil qui convertit les données et génère un fichier .dat pour cplex.

III. III. Points à améliorer

- On pourrait rajouter une fonctionnalité qui permet à l'utilisateur de rentrer les clients, leurs demandes ainsi que le nombre de véhicules et leur capacité directement sur l'application. Cela faciliterait l'utilisation de notre application. On pourrait imaginer une fonctionnalité où l'utilisateur rentre les différentes informations requises dans un tableau ou dans un formulaire ou bien que l'utilisateur créé directement les clients sur une interface graphique.
- On pourrait changer le support de l'application par exemple, un support web.
- On pourrait améliorer notre affichage actuel en ajoutant par exemple le nombre de km parcouru pour chaque véhicule.

III. IV. Problèmes rencontrés

III. IV. I. CPLEX - Trop de véhicules utilisés

Lorsque nous avions réalisé notre premier code CPLEX, nous pensions que nos contraintes étaient correctes car nous obtenions les résultats souhaités par les professeurs. Cependant, suite à une discussion avec un autre groupe, nous avons découvert que nous utilisions une voiture de plus alors que cela n'était pas le cas de base.

Nous avons donc cherché à modifier nos contraintes afin qu'elle fonctionne correctement et que notre code nous affiche alors la bonne solution au problème.

III. IV. II. Python et CPLEX - Problème d'affichage des trajets dans le bon ordre

Lors de notre développement de l'application avec le langage python et sur le logiciel Cplex, nous avons des difficultés pour afficher les trajets réalisés par chaque véhicule à chaque client dans le bon ordre (le trajet qui commence du dépôt en premier et le trajet qui termine au dépôt en dernier et le reste des trajets dans l'ordre logique de départ et d'arrivée pour chaque client)

III. IV. II. L'attribution des véhicules aux tournées par cplex change en fonction de la machine

Nous avons remarqué après plusieurs tests sur différentes machines de l'IUT et sur nos machines personnelles, que cplex attribuent les tournées à différents véhicules lorsqu'on change de machine quand le résultat est renvoyé sur python. Cela ne change rien au résultat trouvé mais nous ne savons pas d'où vient ce changement.

Conclusion

Pour conclure ce projet, notre équipe est globalement satisfaite du résultat de l'application, nous offrant l'opportunité de travailler sur un projet concret tout en respectant une deadline courte, en se développant dans de nouvelles technologies de manière plus approfondie comme l'outil CPLEX et le langage Python, ainsi que l'apprentissage de nouvelles API permettant de combiner les deux.

On reconnaît que plusieurs pistes d'améliorations sont envisageables pour l'application Python, comme plus de détails lors de l'affichage du graphique pour les véhicules ou les villes ou encore de rentrer les jeux de données à la main sur l'application plutôt que d'utiliser un fichier LP.