Modélisation Mathématique

1 Modélisation Mathématique

Dans ce projet, nous abordons le **Split Delivery Vehicle Routing Problem (SD-VRP)** .il permet à un client d'être servi par plusieurs véhicules, sous réserve que la totalité de sa demande soit satisfaite. Ce problème, issu de l'optimisation combinatoire, présente des défis mathématiques et logistiques complexes, particulièrement dans des contextes où la flexibilité des livraisons et l'efficacité des ressources sont essentielles.

Pour résoudre ce problème, nous avons formulé un modèle mathématique précis qui permet de minimiser le coût total qui correspond au distance parcourue tout en respectant les contraintes spécifiques aux véhicules et aux demandes des clients.

1.1 Variables de décision

- x_{ij}^k: Variable binaire qui vaut 1 si le véhicule k emprunte l'arc (i, j), et 0 sinon. Cela permet de représenter les déplacements des véhicules sur le réseau.

- y_i^k: Quantité livrée au client i par le véhicule k.
  Cette variable garantit que la demande de chaque client est satisfaite et lie la quantité livrée aux déplacements des véhicules.

1.2 Fonction objectif

$$Minimiser \sum_{k=1}^{M} \sum_{i=0}^{n} \sum_{j=0}^{n} d_{ij} \cdot x_{ij}^k$$

Cette fonction objectif vise à minimiser la distance totale parcourue par tous les véhicules. Cela contribue directement à réduire les coûts logistiques, en optimisant les trajets des véhicules sur le réseau.

1.3 Contraintes

1. **Satisfaction des demandes :** Chaque client i doit recevoir exactement sa demande totale q_i. Cela garantit que toutes les demandes sont satisfaites sans surplus ni manque

$$\sum_{k=1}^{M} y_i^k = q_i, \quad \forall i \in \{1, \ldots, n\}$$

.

2. **Capacité des véhicules :** La charge totale transportée par un véhicule ne doit pas dépasser sa capacité maximale Q. Cette contrainte reflète les limitations physiques des véhicules.
$$\sum_{i=1}^{n} y_i^k \leq Q, \quad \forall k \in \{1, \ldots, M\}$$

3. **Conservation du flux :** Si un véhicule k entre dans un nœud i, il doit également en sortir. Cela garantit la cohérence des trajets des véhicules.
$$\sum_{j=0}^{n} x_{ij}^k = \sum_{j=0}^{n} x_{ji}^k, \quad \forall i \in \{0, \ldots, n\}, \forall k \in \{1, \ldots, M\}$$

4. **Départ du dépôt :** Chaque véhicule k peut quitter le dépôt au maximum une fois, ce qui reflète un début clair du trajet pour chaque véhicule.
$$\sum_{j=1}^{n} x_{0j}^k \leq 1, \quad \forall k \in \{1, \ldots, M\}$$

5. **Lien entre x et y :**
$$y_i^k \leq q_i \cdot \sum_{j=0}^{n} x_{ji}^k, \quad \forall i \in \{1, \ldots, n\}, \forall k \in \{1, \ldots, M\}$$

6. **Élimination des sous-tours :** Cette contrainte élimine les sous-tours, c'est-à-dire des cycles isolés qui ne passent pas par le dépôt. La variable u_i^k représente la position du nœud i dans la tournée du véhicule k.
$$u_i^k - u_j^k + n \cdot x_{ij}^k \leq n - 1, \quad \forall i, j \in \{1, \ldots, n\}, i \neq j, \forall k \in \{1, \ldots, M\}$$

Conclusion

Ce modèle mathématique est spécifiquement conçu pour répondre aux contraintes du SD-VRP :

- Minimisation des coûts logistiques (via la fonction objectif).
- Satisfaction intégrale des demandes des clients.
- Respect des limites de capacité des véhicules.
- Optimisation des trajets sans créer de sous-tours inutiles.

En utilisant ce modèle, nous avons exploité le solveur **CBC (Coin-or branch and cut)** pour produire des solutions optimisées adaptées aux scénarios logistiques complexes. Le modèle a été traduit dans une structure compatible avec les solveurs d'optimisation, spécifiquement au format LP (*Linear Programming*). Ce fichier, nommé `Case0_SDVRP.lp`, contient la formulation mathématique complète pour l'instance décrite dans le cas **Case0**.

La structure du fichier LP respecte les conventions standard et comprend les éléments suivants :

- **Minimize** : Décrit la fonction objectif, ici la minimisation de la distance totale parcourue par les véhicules.

- **Subject To** : Contient les contraintes définies pour le problème, notamment la satisfaction des demandes, les limites de capacité des véhicules, la conservation des flux, et l'élimination des sous-tours.
- **Bounds** : Définit les bornes des variables décisionnelles.
- **Binary** : Spécifie les variables binaires nécessaires pour modéliser les décisions.
- **End** : Indique la fin du fichier.

Le fichier `Case0_SDVRP.lp` est généré en exécutant le code ci-dessus et sert d'entrée au solveur **CBC** pour produire les solutions optimales.

```python
import math
import os

# Define the data for Case 0
clients = 3
vehicle_capacity = 10
demands = [4, 5, 6]
coordinates = [(0, 0), (2, 3), (4, 5), (6, 7)]  # Depot and clients

# Function to calculate Euclidean distances (rounded down as specified)
def euclidean_distance(i, j):
    x1, y1 = coordinates[i]
    x2, y2 = coordinates[j]
    return math.floor(math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2) + 0.5)

# Generate the LP file content
lp_content = []
lp_content.append("Minimize")
lp_content.append(" obj: " + " + ".join(
    f"{euclidean_distance(i, j)} x_{i}_{j}_k"
    for k in range(1, clients + 2)
    for i in range(clients + 1)
    for j in range(clients + 1) if i != j
))
lp_content.append("Subject To")

# Constraint: Satisfaction of demands
for i in range(1, clients + 1):
    lp_content.append(
        f" demand_{i}: " +
        " + ".join(f"y_{i}_{k}" for k in range(1, clients + 2)) +
        f" = {demands[i - 1]}"
    )

# Constraint: Vehicle capacity
for k in range(1, clients + 2):
    lp_content.append(
        f" capacity_{k}: " +
        " + ".join(f"y_{i}_{k}" for i in range(1, clients + 1)) +
```

```
        f" <= {vehicle_capacity}"
    )

# Constraint: Conservation of flow
for k in range(1, clients + 2):
    for i in range(clients + 1):
        lp_content.append(
            f" flow_{i}_{k}: " +
            " + ".join(f"x_{i}_{j}_{k}" for j in range(clients + 1) if i != j)
            ↪  +
            " - " +
            " - ".join(f"x_{j}_{i}_{k}" for j in range(clients + 1) if i != j)
            ↪  +
            " = 0"
        )

# Constraint: Departure from the depot
for k in range(1, clients + 2):
    lp_content.append(
        f" depot_departure_{k}: " +
        " + ".join(f"x_0_{j}_{k}" for j in range(1, clients + 1)) +
        " <= 1"
    )

# Constraint: Link between x and y
for i in range(1, clients + 1):
    for k in range(1, clients + 2):
        lp_content.append(
            f" link_{i}_{k}: y_{i}_{k} - " +
            f"{demands[i - 1]} " +
            " - ".join(f"x_{j}_{i}_{k}" for j in range(clients + 1) if j != i)
            ↪  +
            " <= 0"
        )

# Constraint: Sub-tour elimination
for k in range(1, clients + 2):
    for i in range(1, clients + 1):
        for j in range(1, clients + 1):
            if i != j:
                lp_content.append(
                    f" subtour_{i}_{j}_{k}: u_{i}_{k} - u_{j}_{k} + " +
                    f"{clients} x_{i}_{j}_{k} <= {clients - 1}"
                )

# Variable bounds and binary declarations
lp_content.append("Bounds")
for k in range(1, clients + 2):
    for i in range(clients + 1):
        for j in range(clients + 1):
            if i != j:
```

```
                lp_content.append(f" 0 <= x_{i}_{j}_{k} <= 1")
    for i in range(1, clients + 1):
        lp_content.append(f" 0 <= y_{i}_{k}")

lp_content.append("Binary")
for k in range(1, clients + 2):
    for i in range(clients + 1):
        for j in range(clients + 1):
            if i != j:
                lp_content.append(f" x_{i}_{j}_{k}")

lp_content.append("End")

# Generate the file in the current directory with a default name
file_name = "Case0_SDVRP.lp"
current_directory = os.getcwd()  # Get the current working directory
file_path = os.path.join(current_directory, file_name)

with open(file_path, "w") as file:
    file.write("\n".join(lp_content))

print(f"File generated and saved as {file_path}")
```