FST
FACULTÉ DES SCIENCES ET TECHNIQUES
DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

AFFECTATION ET TRANSPORT

RÉALISÉ PAR :

Brahim Anougmar
Ismail Iaich
Wissal Hattab

DEMANDÉ PAR :

Mr.J.Abouir

Remerciement

Nous tenons à exprimer nos sincères remerciements à **M. J. Abouir** pour son soutien précieux et ses conseils avisés tout au long du module de recherche opérationnelle 2. Son expertise et son engagement ont grandement enrichi ce projet.

Notre sincère remerciement s'adresse à l'ensemble du personnel de FST Mohammedia, en particulier à nos professeurs. Enfin, nous remercions tous ceux et celles qui ont contribué de près ou de loin à la réalisation de ce modeste travail.

Table des matières

Chapitre 1

Le problème d'Affectation et de Transport

1.1 Introduction

Le problème d'affectation et de transport représente l'un des défis fondamentaux en optimisation combinatoire et en recherche opérationnelle. Ces problèmes, bien que distincts dans leur formulation, partagent des caractéristiques communes et des applications étendues dans une multitude de domaines, allant de la logistique à l'économie, en passant par la planification urbaine et la gestion des ressources humaines. Dans cette introduction, nous allons explorer en profondeur ces deux problèmes, en mettant en lumière leur importance, leur complexité et leurs applications pratiques.

1.2 Importance et Contexte

Le problème d'affectation consiste à attribuer un ensemble d'objets à un ensemble de destinations de manière à minimiser ou maximiser une certaine fonction de coût ou de bénéfice. D'autre part, le problème de transport concerne la distribution optimale de ressources limitées depuis un ensemble de sources vers un ensemble de destinations, en tenant compte des coûts associés au transport de ces ressources. Bien que distincts, ces deux problèmes partagent une structure mathématique similaire et sont souvent résolus à l'aide de techniques d'optimisation similaires.

1.3 Objectifs du Rapport

Ce rapport vise à fournir une compréhension approfondie du problème d'affectation et de transport, en explorant leurs concepts fondamentaux, leurs modèles mathématiques, leurs méthodes de résolution, ainsi que leurs applications dans divers domaines. En outre, nous discuterons des approches avancées pour la résolution de ces problèmes, des tendances émergentes et des défis restants. Enfin, nous illustrerons l'importance pratique de ces problèmes à travers des exemples réels et des études de cas.

En résumé, ce rapport servira de guide exhaustif pour quiconque souhaite comprendre, résoudre et appliquer efficacement les problèmes d'affectation et de transport dans divers contextes.

1.4 Contexte Historique et Développement

Le problème d'affectation trouve ses origines dans les travaux du mathématicien hongrois Dénes Kőnig dans les années 1930, tandis que le problème de transport remonte aux travaux du mathématicien français Gaspard Monge à la fin du 18ème siècle. Depuis lors, ces problèmes ont suscité un intérêt considérable dans la recherche opérationnelle et ont été appliqués avec succès dans de nombreux domaines.

L'évolution des techniques de résolution et l'émergence de nouveaux domaines d'application ont contribué à enrichir notre compréhension de ces problèmes et à étendre leur portée dans divers contextes pratiques.

Chapitre 2

Modélisation Mathématique

Dans ce chapitre, nous examinerons de plus près les modèles mathématiques pour le problème d'affectation et le problème de transport, ainsi que les méthodes de résolution associées.

2.1 Modèle Mathématique pour le Problème Transport

Le problème de transport peut être formulé en utilisant des variables de décision pour représenter les quantités transportées de chaque source à chaque destination. Supposons qu'il y ait p sources et q destinations. Soit xij la quantité transportée de la source i à la destination j. Le problème de transport peut alors être formulé comme suit :

$$\text{Minimiser } \sum_{i=1}^{p} \sum_{j=1}^{q} c_{ij} x_{ij}$$

sous les contraintes :

$$\begin{aligned} &\sum_{j=1}^{q} x_{ij} \leq s_i, \quad \forall i \\ &\sum_{i=1}^{p} x_{ij} \geq d_j, \quad \forall j \\ &x_{ij} \geq 0, \quad \forall i, j \end{aligned}$$

Où si est l'offre de la source i et dj est la demande de la destination j.

La première contrainte représente la capacité de chaque source i. Elle indique que la somme des quantités transportées depuis la source i vers toutes les destinations ne peut pas dépasser l'offre si de la source.

La deuxième contrainte représente la demande de chaque destination j. Elle indique que la somme des quantités transportées vers la destination j depuis toutes les sources doit être au moins égale à la demande dj de la destination.

Les variables xij représentent les quantités transportées de la source i à la destination j, et doivent être positives ou nulles :

$$x_{ij} \geq 0, \quad \forall i, j$$

Ces contraintes définissent le modèle mathématique complet du problème de transport, où l'objectif est de minimiser le coût total du transport tout en respectant les capacités des sources et les demandes des destinations.

2.2 Modèle Mathématique pour le Problème d'Affectation

Le problème d'affectation peut être formulé de manière mathématique en utilisant des variables binaires pour représenter les décisions d'affectation. Supposons qu'il y ait n personnes et m tâches. Soit xij une variable binaire qui vaut 1 si la personne i est affecté à la tâche j, et 0 sinon. Le problème d'affectation peut alors être décrit comme suit :

$$\text{Minimiser } \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

sous les contraintes :

$$\sum_{j=1}^{m} x_{ij} = 1, \quad \forall i$$
$$\sum_{i=1}^{n} x_{ij} = 1, \quad \forall j$$
$$x_{ij} \in \{0, 1\}, \quad \forall i, j$$

La première contrainte assure qu’une seule personne est affecté à chaque tâche. Elle garantit que chaque personne est affectée exactement à une tâche, sans qu'il y ait de tâche non affectée.

Cette contrainte assure qu'une seule tâche est attribuée à chaque agent. Elle garantit que chaque tâche est attribuée exactement à un agent, sans qu'il y ait de tâche non attribuée.

Les variables xij représentent les affectations, et doivent être des valeurs binaires (0 ou 1) :

$$x_{ij} \in \{0, 1\}, \quad \forall i, j$$

Chapitre 3

Problème d'Affectation

3.1 Algorithme de Résolution du Problème

1. **Initialisation :** Initialisez une matrice de coûts C représentant les coûts d'affectation entre les agents et les tâches. Convertissez le problème de minimisation en un problème de maximisation en inversant les coûts.
2. **Étape 1 : Réduction des Lignes :** Pour chaque ligne de la matrice C, soustrayez le plus petit élément de la ligne à tous les autres éléments de cette ligne.
3. **Étape 2 : Réduction des Colonnes :** Pour chaque colonne de la matrice résultante, soustrayez le plus petit élément de la colonne à tous les autres éléments de cette colonne.
4. **Sélection des Affectations :** Sélectionnez des affectations initiales en choisissant le plus petit élément non barré dans chaque ligne et colonne. Si une ligne ou une colonne a plus d'un élément non barré, choisissez arbitrairement.
5. **Marquage des Lignes et Colonnes :** Marquez les lignes et les colonnes des éléments sélectionnés. Si toutes les lignes et colonnes sont marquées, passez à l'étape 7. Sinon, passez à l'étape 6.
6. **Mise à Jour des Marques :** Trouvez la plus petite valeur non marquée dans la matrice. Soustrayez cette valeur de tous les éléments non marqués et ajoutez-la à tous les éléments doublement marqués. Retournez à l'étape 5.
7. **Calcul de la Solution :** Les affectations sélectionnées correspondent à la solution optimale. Si le problème initial était un problème de minimisation, il suffit de soustraire les éléments sélectionnés des coûts initiaux.

3.2 Pseudocode de l'algorithme

```
Fonction MethodeHongroise(C)
    RéduireLignes(C)
    RéduireColonnes(C)
    AffectationsInitiales = SélectionAffectationsInitiales(C)
    Tant que il existe des affectations non complètes dans AffectationsInitiales
        MarquerLignesEtColonnes(AffectationsInitiales)
        MettreÀJourMarques(C)
    Fin Tant que
    Retourner AffectationsInitiales
Fin Fonction

Fonction RéduireLignes(C)
    Pour chaque ligne i de C
        minimum = trouverMinimumDeLaLigne(C, i)
        Pour chaque élément C[i][j] de la ligne i
            C[i][j] -= minimum
    Fin Pour chaque
Fin Fonction

Fonction RéduireColonnes(C)
    Pour chaque colonne j de C
        minimum = trouverMinimumDeLaColonne(C, j)
        Pour chaque élément C[i][j] de la colonne j
            C[i][j] -= minimum
    Fin Pour chaque
Fin Fonction

Fonction SélectionAffectationsInitiales(C)
    affectations = []
    Pour chaque ligne i de C
        Pour chaque colonne j de C
            Si C[i][j] est égal à 0 et ni i ni j n'ont été affectés
                affectations.ajouter((i, j))
                MarquerLaLigneEtLaColonneCommeAffectées(i, j)
    Fin Pour chaque
    Retourner affectations
Fin Fonction

Fonction MarquerLignesEtColonnes(AffectationsInitiales)
    lignesMarquées = set()
    colonnesMarquées = set()
    Pour chaque affectation (i, j) dans AffectationsInitiales
        Marquer la ligne i
        Marquer la colonne j
    Fin Pour chaque
Fin Fonction

Fonction MettreÀJourMarques(C)
    minimumNonMarqué = trouverMinimumNonMarqué(C)
    Pour chaque élément C[i][j] non marqué
        C[i][j] -= minimumNonMarqué
    Pour chaque élément C[i][j] doublement marqué
        C[i][j] += minimumNonMarqué
Fin Fonction

```

3.3 Code en Python

```python
import numpy as np

def MethodeAffectation(C):
    # Création d'une copie de la matrice de coûts
    C_copy = np.copy(C)

    # Réduction des lignes
    C_copy -= np.min(C_copy, axis=1, keepdims=True)
    # Réduction des colonnes
    C_copy -= np.min(C_copy, axis=0, keepdims=True)

    n, m = C_copy.shape
    affectations = []

    while len(affectations) < n:
        # Sélection des affectations initiales
        row_indices, col_indices = np.where(C_copy == 0)
        row_covered = set()
        col_covered = set()
        for i, j in zip(row_indices, col_indices):
            if i not in row_covered and j not in col_covered:
                affectations.append((i, j))
                row_covered.add(i)
                col_covered.add(j)

        # Si toutes les lignes sont couvertes, sortir de la boucle
        if len(affectations) == n:
            break
```

```python
        # Marquer les lignes et les colonnes
        marked_rows = set([row for row, _ in affectations])
        marked_cols = set([col for _, col in affectations])

        # Trouver la plus petite valeur non marquée
        min_unmarked = np.inf
        for i in range(n):
            if i not in marked_rows:
                for j in range(m):
                    if j not in marked_cols:
                        min_unmarked = min(min_unmarked, C_copy[i, j])

        # Soustraire la plus petite valeur non marquée des éléments non marqués
        # et l'ajouter aux éléments doublement marqués
        for i in range(n):
            for j in range(m):
                if i in marked_rows and j in marked_cols:
                    C_copy[i, j] += min_unmarked
                elif i not in marked_rows and j not in marked_cols:
                    C_copy[i, j] -= min_unmarked

    # Calcul du coût total des affectations à partir de la matrice initiale
    cout_total = sum(C[affectation] for affectation in affectations)

    return affectations, cout_total

```

3.4 Un exemple d'application

```
56 # Exemple d'utilisation
57 C = np.array([[4, 5, 6],
58                [3, 2, 1],
59                [6, 5, 4]])
60
61 affectations, cout_total = MethodeAffectation(C)
62 print("Affectations optimales:", affectations)
63 print("Coût total des affectations:", cout_total)
64
```

```
Affectations optimales: [(0, 0), (1, 1), (2, 2)]
Coût total des affectations: 10
```

Chapitre 4
Problème de Transport

Pour déterminer la solution réalisable de ce problème on a deux méthodes du coin Nord-Ouest et du Moindre cout.

4.1 Algorithme de la méthode du Coin Nord-Ouest

1. **Initialisation :** Démarrez à partir du coin nord-ouest de la matrice d'offre-demande.
2. **Sélection de la Case :** Sélectionnez la case avec la plus grande offre ou demande disponible.
3. **Affectation :** Affectez autant de marchandises que possible à la case sélectionnée sans dépasser l'offre ou la demande disponible.
4. **Mise à Jour :** Réduisez l'offre ou la demande disponible et mettez à jour les quantités restantes dans les cases correspondantes. Si l'offre ou la demande atteint zéro, éliminez la ligne ou la colonne correspondante de la matrice.
5. **Répétition :** Répétez les étapes 2 à 4 jusqu'à ce que toutes les offres ou demandes soient épuisées.

4.2 Pseudocode de la méthode

```
Fonction MethodeCoinNordOuest(offres, demandes, coûts)
    Tant que il reste des offres non affectées ou des demandes non satisfaites
        Si nombre d'offres non affectées > nombre de demandes non satisfaites
            Sélectionner la case (i, j) telle que offres[i] est maximale
        Sinon
            Sélectionner la case (i, j) telle que demandes[j] est maximale
        Fin Si

        quantité_affectée = min(offres[i], demandes[j])
        affectations[i][j] = quantité_affectée

        offres[i] -= quantité_affectée
        demandes[j] -= quantité_affectée

        Si offres[i] == 0
            Supprimer la ligne i de la matrice des coûts
        Fin Si

        Si demandes[j] == 0
            Supprimer la colonne j de la matrice des coûts
        Fin Si
    Fin Tant que

    Retourner affectations
Fin Fonction
```

4.3 Code en Python

```python
import numpy as np

def coin_nord_ouest(offres, demandes, couts):
    lignes, colonnes = len(offres), len(demandes)
    affectations = np.zeros((lignes, colonnes))

    i, j = 0, 0

    while i < lignes and j < colonnes:
        # Calcul de la quantité à affecter (minimum entre l'offre et la demande)
        quantite_affectee = min(offres[i], demandes[j])

        # Affectation de la quantité
        affectations[i][j] = quantite_affectee

        # Mise à jour des offres et demandes restantes
        offres[i] -= quantite_affectee
        demandes[j] -= quantite_affectee

        # Passage à la prochaine ligne ou colonne si l'offre ou la demande est épuisée
        if offres[i] == 0:
            i += 1
        if demandes[j] == 0:
            j += 1

    return affectations

```

4.4 Un exemple d'application

```
28 # Exemple d'utilisation
29 offres = np.array([20, 30, 40])
30 demandes = np.array([30, 50, 30])
31 couts = np.array([[5, 8, 7],
32                  [7, 12, 10],
33                  [8, 6, 4]])
34
35 affectations = coin_nord_ouest(offres, demandes, couts)
36 print("Affectations optimales :\n", affectations)
37 # Calcul du coût total des affectations
38 cout_total = np.sum(affectations * couts)
39 print("Coût total des affectations :", cout_total)
40
41
```

```
Affectations optimales :
 [[20.  0.  0.]
 [10. 20.  0.]
 [ 0. 30. 10.]]
Coût total des affectations : 630.0
```

4.5 Algorithme de la méthode du Moindre Cout

1. **Initialisation** : Commencez avec toutes les offres et demandes non satisfaites.
2. **Sélection de la Case :** Recherchez la cellule avec le coût le plus bas dans la matrice des coûts parmi les cellules correspondant aux offres et aux demandes non satisfaites.
3. **Affectation :** Affectez autant de marchandises que possible à cette cellule sans dépasser l'offre ou la demande disponible.

4. **Mise à Jour :** Réduisez l'offre ou la demande disponible et mettez à jour les quantités restantes dans les cases correspondantes. Si l'offre ou la demande atteint zéro, éliminez la ligne ou la colonne correspondante de la matrice.
5. **Répétition :** Répétez les étapes 2 à 4 jusqu'à ce que toutes les offres ou demandes soient épuisées.

4.6 Pseudocode de la méthode

```
Fonction MethodeCoutMinimum(offres, demandes, couts)
    Tant que il reste des offres non affectées ou des demandes non satisfaites
        min_cout, i_min, j_min = trouverCelluleMin(couts, offres, demandes)

        quantite_affectee = min(offres[i_min], demandes[j_min])

        affectations[i_min][j_min] = quantite_affectee

        offres[i_min] -= quantite_affectee
        demandes[j_min] -= quantite_affectee

        Si offres[i_min] == 0
            Supprimer la ligne i_min de la matrice des coûts
        Fin Si

        Si demandes[j_min] == 0
            Supprimer la colonne j_min de la matrice des coûts
        Fin Si
    Fin Tant que

    Retourner affectations
Fin Fonction
```

4.7 Code en Python

```python
import numpy as np
def trouver_cellule_min(couts, offres, demandes):
    min_cout = np.inf
    i_min, j_min = -1, -1

    for i in range(len(offres)):
        for j in range(len(demandes)):
            if offres[i] > 0 and demandes[j] > 0:
                if couts[i][j] < min_cout:
                    min_cout = couts[i][j]
                    i_min, j_min = i, j
    return min_cout, i_min, j_min
def methode_cout_minimum(offres, demandes, couts):
    lignes, colonnes = len(offres), len(demandes)
    affectations = np.zeros((lignes, colonnes))
    while np.any(offres > 0) and np.any(demandes > 0):
        min_cout, i_min, j_min = trouver_cellule_min(couts, offres, demandes)
        if min_cout == np.inf:
            break
        quantite_affectee = min(offres[i_min], demandes[j_min])

        affectations[i_min][j_min] = quantite_affectee

        offres[i_min] -= quantite_affectee
        demandes[j_min] -= quantite_affectee
    return affectations
```

4.8 Un exemple d'application

```python
# Exemple d'utilisation
offres = np.array([20, 30, 40])
demandes = np.array([30, 50, 30])
couts = np.array([[5, 8, 7],
                  [7, 12, 10],
                  [8, 6, 4]])

affectations = methode_cout_minimum(offres, demandes, couts)
print("Affectations optimales :\n", affectations)

# Calcul du coût total des affectations
cout_total = np.sum(affectations * couts)
print("Coût total des affectations :", cout_total)
```

```
Affectations optimales :
 [[20.  0.  0.]
 [10. 20.  0.]
 [ 0. 10. 30.]]
Coût total des affectations : 590.0
```

Chapitre 5

Résoudre un problème réel

5.1 Problème D’affectation

Problème :

Une entreprise de logistique doit affecter des chauffeurs à des livraisons pour s'assurer que toutes les commandes sont livrées dans les délais impartis. Chaque chauffeur a une capacité limitée de livraisons qu'il peut effectuer dans une journée. Les coûts de transport varient en fonction de la distance et du temps requis pour chaque livraison. L'objectif est de minimiser le coût total de livraison tout en respectant les capacités de chaque chauffeur.

Données :

Chauffeurs :

Chauffeur A peut effectuer jusqu'à 5 livraisons par jour.

Chauffeur B peut effectuer jusqu'à 4 livraisons par jour.

Livraisons :

Livraison 1 : De l'entrepôt au client A (distance : 10 km, temps de livraison : 1 heure).

Livraison 2 : De l'entrepôt au client B (distance : 15 km, temps de livraison : 1.5 heures).

Livraison 3 : De l'entrepôt au client C (distance : 20 km, temps de livraison : 2 heures).

Livraison 4 : De l'entrepôt au client D (distance : 12 km, temps de livraison : 1.2 heures).

Livraison 5 : De l'entrepôt au client E (distance : 18 km, temps de livraison : 1.8 heures).

Coûts :

Coût par kilomètre : 0.5 $.

Coût par heure : 10 $.

Résolution :

Nous pouvons utiliser l'algorithme du coût minimum pour résoudre ce problème. Nous allons créer une matrice de coûts où chaque ligne représente un chauffeur et chaque colonne représente une livraison. Les coûts seront calculés en fonction de la distance et du temps de livraison pour chaque livraison. Ensuite, nous utiliserons l'algorithme du coût minimum pour affecter les livraisons à chaque chauffeur tout en respectant leurs capacités.

Code Python :

Voici un exemple de code Python pour résoudre ce problème :

```python
import numpy as np

# Données
chauffeurs = {'A': 5, 'B': 4}
livraisons = {'1': {'distance': 10, 'temps': 1},
              '2': {'distance': 15, 'temps': 1.5},
              '3': {'distance': 20, 'temps': 2},
              '4': {'distance': 12, 'temps': 1.2},
              '5': {'distance': 18, 'temps': 1.8}}
cout_kilometre = 0.5
cout_heure = 10

# Calcul des coûts
couts = np.zeros((len(chauffeurs), len(livraisons)))
for i, (chauffeur, capacite) in enumerate(chauffeurs.items()):
    for j, (livraison, details) in enumerate(livraisons.items()):
        couts[i][j] = (details['distance'] * cout_kilometre) + (details['temps'] * cout_heure)

# Résolution du problème
from scipy.optimize import linear_sum_assignment
affectations = linear_sum_assignment(couts)
print("Affectations optimales :\n", affectations)

```

```
Affectations optimales :
 (array([0, 1], dtype=int64), array([0, 3], dtype=int64))
```

Puisque les affectations optimales sont trouvées, on peut utiliser ces résultats pour planifier les livraisons et minimiser les coûts tout en respectant les contraintes des chauffeurs.

```python
# Affectations optimales
chauffeurs_indices, livraisons_indices = affectations

# Planification des livraisons
planification = {}

for i, chauffeur_idx in enumerate(chauffeurs_indices):
    chauffeur = list(chauffeurs.keys())[chauffeur_idx]
    livraison = list(livraisons.keys())[livraisons_indices[i]]
    if livraison not in planification:
        planification[livraison] = [chauffeur]
    else:
        planification[livraison].append(chauffeur)

# Affichage de la planification des livraisons
print("Planification des livraisons :")
for livraison, chauffeurs in planification.items():
    print(f"Livraison {livraison} : Chauffeurs {', '.join(chauffeurs)}")

```

```
Planification des livraisons :
Livraison 1 : Chauffeurs A
Livraison 4 : Chauffeurs B
```

5.2 Problème du transport

Problème

Une entreprise de distribution alimentaire dispose de trois entrepôts situés dans différentes villes (A, B et C). Cette entreprise doit livrer différents types de produits alimentaires à cinq supermarchés situés dans la même région. Chaque entrepôt a une capacité de stockage différente et chaque supermarché a une demande spécifique pour chaque produit.

L'objectif est de minimiser les coûts de transport tout en satisfaisant les demandes de chaque supermarché.

Données :

Entrepôts :

Entrepôt A : Capacité de stockage de 100 unités.

Entrepôt B : Capacité de stockage de 150 unités.

Entrepôt C : Capacité de stockage de 200 unités.

Supermarchés :

Supermarché 1 : Demande de 80 unités de produit A, 50 unités de produit B et 70 unités de produit C.

Supermarché 2 : Demande de 100 unités de produit A, 120 unités de produit B et 90 unités de produit C.

Supermarché 3 : Demande de 70 unités de produit A, 60 unités de produit B et 80 unités de produit C.

Supermarché 4 : Demande de 60 unités de produit A, 80 unités de produit B et 100 unités de produit C.

Supermarché 5 : Demande de 90 unités de produit A, 70 unités de produit B et 110 unités de produit C.

Coûts de Transport :

Coût d'expédition par unité de produit de l'entrepôt à chaque supermarché.

Résolution :

Nous pouvons résoudre ce problème en utilisant l'algorithme du coût minimum pour trouver les affectations optimales des produits de chaque entrepôt à chaque supermarché, tout en respectant les capacités de stockage de chaque entrepôt et les demandes de chaque supermarché.

Nous utiliserons ensuite les résultats pour planifier les expéditions de chaque entrepôt vers chaque supermarché de manière à minimiser les coûts de transport tout en satisfaisant les demandes des supermarchés.

Code Python :

Nous voulons maintenant résoudre ce problème en utilisant Python avec la bibliothèque scipy.optimize pour résoudre le problème d'assignation (transport). Nous allons utiliser les données fournies pour les entrepôts, les supermarchés et les coûts de transport, et ensuite nous planifierons les expéditions en utilisant les affectations optimales obtenues.

```python
import numpy as np
from scipy.optimize import linear_sum_assignment
# Données
entrepots = {'A': 100, 'B': 150, 'C': 200}
supermarches = {'1': {'A': 80, 'B': 50, 'C': 70},
                '2': {'A': 100, 'B': 120, 'C': 90},
                '3': {'A': 70, 'B': 60, 'C': 80},
                '4': {'A': 60, 'B': 80, 'C': 100},
                '5': {'A': 90, 'B': 70, 'C': 110}}
couts_transport = np.array([[5, 4, 6],
                            [7, 6, 8],
                            [6, 5, 7]])
# Matrices d'offres et de demandes
offres = np.array([entrepots[entrepot] for entrepot in entrepots.keys()])
demandes = np.array([sum(supermarches[supermarche].values())
                     for supermarche in supermarches.keys()])
# Résolution du problème d'assignation (transport)
affectations = linear_sum_assignment(couts_transport)
# Planification des expéditions
planification = {}
for i, entrepot_idx in enumerate(affectations[0]):
    supermarche_idx = affectations[1][i]
```

```
    supermarche_idx = affectations[1][i]
    entrepot = list(entrepots.keys())[entrepot_idx]
    supermarche = list(supermarches.keys())[supermarche_idx]
    quantites = {produit: supermarches[supermarche][produit]
                 for produit in supermarches[supermarche].keys()}
    if entrepot not in planification:
        planification[entrepot] = {supermarche: quantites}
```

Conclusion

En conclusion, les problèmes d'affectation et de transport sont des défis d'optimisation essentiels pour maximiser l'efficacité opérationnelle et minimiser les coûts dans divers domaines. En comprenant les contraintes et en utilisant des méthodes telles que l'algorithme du coût minimum pour l'affectation et l'algorithme du transport pour le transport, les organisations peuvent améliorer leur planification logistique, leur allocation de ressources et leur satisfaction client. En résolvant ces problèmes de manière efficace, les entreprises peuvent réaliser des économies substantielles tout en maintenant des opérations fluides et efficaces.