



# GP27

Three stage dynamic heuristic for multiple plants capacitated  
lot sizing with sequence-dependent transient costs

[Shang NI, Yiyang Tang]

Automne

# Sommaire

• Partie I	
• Résumé de l' article choisi	1
• Contributions des auteurs	1
• Explication du modèle	2
• Méthodologie de recherche	7
• Partie II	
• Contribution personnelle	8
• implémentation du code	8
• Répartition du travail	16

## 1. Résumé de l' article choisi

En raison de l'accent accru sur les clients, de la réduction du cycle de vie des produits et des avancées technologiques, la concurrence sur le marché mondial devient de plus en plus intense. Cela nécessite des stratégies de chaîne d'approvisionnement plus efficaces et efficientes pour répondre aux besoins du marché tout en améliorant la position sur le marché grâce à une meilleure gestion de la chaîne d'approvisionnement. Une entreprise pétrochimique collabore avec plusieurs filiales disposant de capacités de production fixes et capables de produire divers niveaux de produits. Les décisions de dimensionnement des lots visant à minimiser les coûts totaux des stocks et les coûts de réglage dans un réseau de production et de distribution ont été largement étudiées. Dans cet article, nous prenons en compte l'impact du transport entre les différents niveaux de la chaîne d'approvisionnement ainsi que les coûts d'achat des matières premières, afin de maximiser le profit total de la chaîne d'approvisionnement. L'intégration de tous ces éléments décisionnels dans un seul modèle mathématique augmente considérablement la complexité de la résolution du problème.

## 2. Contributions des auteurs

Tout d'abord, nous avons développé un modèle de programmation en nombres entiers mixtes pour une chaîne d'approvisionnement à quatre niveaux, intégrant la détermination de la taille des lots, la planification avec des coûts transitoires dépendant de la séquence, ainsi que les décisions de transport et d'entreposage entre les fournisseurs, les usines, les entrepôts et les clients. Deuxièmement, étant donné que le problème est NP-difficile, nous avons donc développé une heuristique itérative en trois étapes qui fournit des solutions de qualité avec un temps de calcul très acceptable.

## 2.1 Explication du modèle

Le modèle développé répondra aux questions liées à la quantité de matières premières à acheter auprès de chaque fournisseur, au plan de production pour chaque filiale, à l'affectation des grades à produire par chaque filiale, à la taille des lots et à la séquence dans laquelle ces grades doivent être produits, aux niveaux de stock des différents grades dans chaque entrepôt, ainsi qu'à la sélection de l'entrepôt qui répondra à chaque demande client.

Le CLSP avec des coûts de configuration dépendants de la séquence a été prouvé comme étant NP-difficile (Florian, Lenstra et Rinnooy Kan, 1980). De même, le problème de séquençement avec des coûts de configuration dépendants de la séquence a également été prouvé comme étant NP-difficile par Coleman (1992). En combinant ces deux problèmes, on obtient un CLSP avec des coûts de configuration dépendants de la séquence, ce qui est également NP-difficile. Afin de calculer une solution faisable proche de l'optimal en un temps raisonnable, nous proposons une méthode heuristique itérative en trois étapes, illustrée dans la figure. Ici, nous n'expliquerons pas en détail les trois étapes de cette heuristique.

Ensuite, nous allons analyser en détail le modèle de programmation en nombres entiers mixtes, qui comprend les paramètres, les variables, la fonction objectif et les contraintes.

### Indices:

- $s$  : Spplier,  $s=1, \dots, S$
- $r$  : RawMaterial,  $r=1, \dots, R$
- $a$  : Affiliate,  $a=1, \dots, A$
- $g, h$  : Grade,  $g=1, \dots, G$
- $w$  : Warehouse,  $w=1, \dots, W$
- $k$  : Customer,  $k=1, \dots, K$
- $t$  : Period,  $t=1, \dots, T$

## Notation :

$C_{sar}^{trans}$  : Coût de transport par tonne de matière première  $r$  du fournisseur  $s$  à l'affilié  $a$

$C_{awg}^{trans'}$  : Coût de transport par tonne de grade  $g$  de l'affilié  $a$  à l'entrepôt  $w$

$C_{wkg}^{trans''}$  : Coût de transport par tonne de grade  $g$  de l'entrepôt  $w$  au client  $k$

$C_{sar}^{raw}$  : Coût d'achat par tonne de matière première  $r$  du fournisseur  $s$  à l'affilié  $a$

$P_{agt}^{prime}$  : Prix par tonne de grade premier  $g$  dans l'affilié  $a$  pendant la période  $t$

$C_{agt}^{nprime}$  : Coût de production par tonne de matériau transitionnel formé entre les grades  $g$  et  $h$  dans l'affilié  $a$  pendant la période  $t$

$C_{wgt}^{inv}$  : Coût de stockage par tonne de grade  $g$  dans l'entrepôt  $w$  pendant la période  $t$

$\lambda_{sr}$  : Disponibilité de la matière première  $r$  en tonnes du fournisseur  $s$  par période, constante sur toutes les périodes

$\lambda'_{at}$  : Capacité de production de l'affilié  $a$  pendant la période  $t$

$O_{aght}$  : Quantité de matériau transitionnel en tonnes formée entre les grades  $g$  et  $h$  dans l'affilié  $a$  pendant la période  $t$

$\beta_{arg}$  : Quantité de matière première  $r$  nécessaire pour produire une tonne de grade  $g$  dans l'affilié  $a$

$D_{gkt}$  : Demande du grade  $g$  pour le client  $k$  pendant la période  $t$

$F_{sart}$  : Tonnes de matière première  $r$  expédiées du fournisseur  $s$  à l'affilié  $a$  pendant la période  $t$

$F'_{awgt}$  : Tonnes de grade  $g$  expédiées de l'affilié  $a$  à l'entrepôt  $w$  pendant la période  $t$

$F''_{wkg}$  : Tonnes de grade  $g$  expédiées de l'entrepôt  $w$  au client  $k$  pendant la période  $t$

$X_{agt}$  : Tonnes de grade  $g$  produites dans l'affilié  $a$  pendant la période  $t$

$Z_{aght}$  : Une variable binaire égale à 1 si une transition est effectuée entre les grades  $g$  et  $h$  dans l'affilié  $a$  pendant la période  $t$ , 0 sinon

$\alpha_{agt}$ : Une variable binaire égale à 1 si le grade  $g$  est produit en premier dans l'affilié  $a$  pendant la période  $t$ , 0 sinon

$V_{agt}$ : Variable auxiliaire du grade  $g$  dans l'affilié  $a$  pendant la période  $t$  utilisée pour valider la contrainte d'élimination des sous-tournées

$I_{wgt}$ : Niveau de stock du grade  $g$  dans l'entrepôt  $w$  pendant la période  $t$

## Fonction objectif:

$$\begin{aligned} \text{Max} \left[ \sum_{a=1}^A \sum_{g=1}^G \sum_{t=1}^T P_{agt}^{\text{prime}} * X_{agt} \right] \\ - \left[ \sum_{s=1}^S \sum_{a=1}^A \sum_{r=1}^R \sum_{t=1}^T C_{sar}^{\text{trans}} * F_{sart} + \sum_{a=1}^A \sum_{w=1}^W \sum_{g=1}^G \sum_{t=1}^T C_{awg}^{\text{trans}'} * F'_{awgt} \right. \\ \left. + \sum_{w=1}^W \sum_{k=1}^K \sum_{g=1}^G \sum_{t=1}^T C_{wkg}^{\text{trans}''} * F''_{wkg t} \right] - \left[ \sum_{s=1}^S \sum_{a=1}^A \sum_{r=1}^R \sum_{g=1}^G C_{sar}^{\text{raw}} * F_{sart} \right] \\ - \left[ \sum_{a=1}^A \sum_{g=1}^G \sum_{h=1, h \neq g}^G \sum_{t=1}^T C_{aght}^{\text{nprime}} * O_{aght} * Z_{aght} \right] - \left[ \sum_{w=1}^W \sum_{g=1}^G \sum_{t=1}^T C_{wgt}^{\text{inv}} * I_{wgt} \right] \quad (1) \end{aligned}$$

## Contraintes:

$$\sum_{g=1}^G \beta_{arg} * X_{agt} = \sum_{s=1}^S F_{sart} \quad \forall a, r, t \quad (2)$$

$$\sum_{a=1}^A F_{sart} \leq \lambda_{sr} \quad \forall s, r, t \quad (3)$$

$$\sum_{g=1}^G X_{agt} + \sum_{g=1}^G \sum_{h=1, h \neq g}^G O_{aght} * Z_{aght} \leq \lambda'_{at} \quad \forall a, t \quad (4)$$

$$X_{agt} \leq \lambda'_{at} * \left( \sum_{h=1, h \neq g}^G Z_{ahgt} + \alpha_{agt} \right) \quad \forall a, g, t \quad (5)$$

$$\sum_{h=1, h \neq g}^G Z_{ahgt} + \alpha_{agt} \leq 1 \quad \forall a, g, t \quad (6)$$

$$\sum_{g=1}^G \alpha_{agt} = 1 \quad \forall a, t \quad (7)$$

$$Z_{aght} \leq X_{aht} \quad \forall a, g, h, t \quad (8)$$

$$\alpha_{agt} + \sum_{h=1, h \neq g}^G Z_{ahgt} = \alpha_{agt+1} + \sum_{h=1, h \neq g}^G Z_{aght} \quad \forall a, g, t \quad (9)$$

$$V_{agt} + (N * z_{aght}) - (N - 1) - (N * \alpha_{agt}) \leq V_{aht} \quad \forall a, g, h \neq g, t \quad (10)$$

$$X_{agt} = \sum_{w=1}^W F'_{awgt} \quad \forall a, g, t \quad (11)$$

$$I_{wgt-1} + \sum_{a=1}^A F'_{awgt} = I_{wgt} + \sum_{k=1}^K F''_{wkgt} \quad \forall w, g, t \quad (12)$$

$$\sum_{w=1}^W F''_{wkgt} = D_{gkt} \quad \forall g, k, t \quad (13)$$

## Explication des contraintes :

La fonction objectif (1) maximise le bénéfice net en soustrayant les différents coûts des revenus. Les coûts pris en compte sont les coûts de transport entre les quatre niveaux de la chaîne d'approvisionnement, le coût des matières premières, le coût des matériaux de transition et le coût de stockage.

Les contraintes (2) et (3) garantissent que la quantité requise de chaque matière première  $rr$ , nécessaire pour produire le grade  $gg$  dans la filiale  $aa$ , sera transportée depuis les différents fournisseurs  $ss$  à chaque période  $tt$ , sans dépasser la capacité des fournisseurs.

La contrainte (4) garantit que la quantité totale des grades premiers produite à chaque période ne peut pas dépasser la capacité de la filiale.

La contrainte (5) garantit que chaque grade dans chaque filiale peut être produit à la période  $tt$  uniquement si la machine est d'abord configurée pour ce grade ou s'il existe au moins une configuration pour ce grade.

La contrainte (6) garantit que chaque grade dans chaque filiale à chaque période ne peut avoir qu'une seule configuration, soit issue de la première configuration, soit par une transition depuis un autre grade.

La contrainte (7) garantit que la configuration au début de chaque période ne peut être effectuée que pour un seul grade à la fois dans chaque filiale.

La contrainte (8) garantit qu'il ne peut y avoir de transition vers un grade donné que si ce grade est produit.

La contrainte (9) représente la configuration reportée (carry-over setup).

La contrainte (10) garantit que le modèle évite les sous-tournées déconnectées et maintient la faisabilité, ce qui est couramment utilisé dans le Problème du Voyageur de Commerce et a été proposé par Nemhauser et Wolsey (1988). Cette contrainte garantit que chaque grade peut être produit une seule fois par période dans chaque filiale.

La contrainte (11) garantit que tous les grades produits dans chaque filiale à chaque période doivent être transportés vers les entrepôts.

La contrainte (12) représente l'équilibre des stocks pour chaque grade dans chaque entrepôt à chaque période.

La contrainte (13) indique que la quantité totale du grade  $gg$  transportée depuis tous les entrepôts doit être égale à la demande de chaque client  $kk$  à chaque période  $tt$ .



## **2.2 Méthodologie de recherche**

L'entreprise dispose de plusieurs entrepôts, et les matières premières nécessaires à la production sont fournies par différents fournisseurs. Le passage d'un grade à un autre entraîne des coûts de configuration dépendants de la séquence. Cet article propose un modèle de programmation linéaire en nombres entiers (MILP) pour résoudre les problèmes de planification et d'ordonnancement multi-périodes. Il développe un modèle complet de chaîne d'approvisionnement à quatre niveaux, incluant les fournisseurs, les usines, les entrepôts et les clients.

Comparé aux études précédentes qui se concentrent uniquement sur un ou plusieurs niveaux spécifiques de la chaîne d'approvisionnement, ce modèle couvre l'ensemble du réseau de la chaîne d'approvisionnement, offrant ainsi une meilleure capacité à résoudre des problèmes complexes rencontrés dans la pratique. De plus, il intègre plusieurs caractéristiques dynamiques clés des coûts de la chaîne d'approvisionnement, ce qui permet non seulement de répondre aux besoins de production, mais aussi d'optimiser ces coûts afin d'améliorer la rentabilité globale de la chaîne d'approvisionnement.

### 1. Contribution personnelle

Nous avons reproduit dans une certaine mesure les résultats de l'article via Python.

### 2. implémentation du code

• **Introduction:** Dans le contexte donné par l'article, programmer en Python en utilisant Gurobi pour concevoir les paramètres, les variables de décision, la fonction objectif et les contraintes. Ensuite, résoudre le problème pour trouver la solution optimale en utilisant la méthode de programmation linéaire.

• **Détail du code:**

*# Importez les bibliothèques requises*

```
import gurobipy as gp
```

```
from gurobipy import GRB
```

```
import numpy as np
```

*#Initialisation du modèle*

```
model = gp.Model("SupplyChainOptimization")
```

```
S = 4 # Supplier
```

```
R = 4 # RawMaterial
```

```
A = 4 # Affiliate
```

```
G = 4 # Grade
```

```
W = 4 # Warehouse
```

```
K = 10 # Customer
```

```
T = 6 # Period
```

*# Définir des variables de décision*

*# F\_sart: Tonnes de matières premières r expédiées des fournisseurs à l'affilié a au cours du temps période t*

```
F_sart = model.addVars(S, A, R, T, vtype=GRB.CONTINUOUS, lb=0, name="F_sart")
```

*# F\_awgt: Tonnes de qualité g expédiées de la filiale a à l'entrepôt w pendant la période t*

```
F_awgt = model.addVars(A, W, G, T, vtype=GRB.CONTINUOUS, lb=0, name="F_awgt")
```

```

# F_wkgt: Tonnes de qualité g expédiées de l'entrepôt w au client k pendant la période t
F_wkgt = model.addVars(W, 10, G, T, vtype=GRB.CONTINUOUS, lb=0, name="F_wkgt")
# X_agt: Tonnes de qualité g produites dans la filiale a pendant la période t
X_agt = model.addVars(A, G, T, vtype=GRB.CONTINUOUS, lb=0, name="X_agt")
# Z_aght: Une variable binaire est égale à 1 si une transition est effectuée entre les grades g et
h dans affilié a pendant la période t, 0 sinon

Z_aght = model.addVars(A, G, G, T, vtype=GRB.BINARY, name="Z_aght")
# alpha_agt: Une variable binaire est égale à 1 si le grade g est produit en premier dans l'affilié
a pendant période de temps t, 0 sinon
alpha_agt = model.addVars(A, G, T, vtype=GRB.BINARY, name="alpha_agt")
# V_agt: Variable auxiliaire de grade g dans l'affilié a pendant la période t qui sert à
valider la contrainte d'élimination des sous-tours
V_agt = model.addVars(A, G, T, vtype=GRB.CONTINUOUS, lb=0, name="V_agt")
# I_wgt: Niveau de stock de qualité g dans l'entrepôt w pendant la période t
I_wgt = model.addVars(W, G, T, vtype=GRB.CONTINUOUS, lb=0, name="I_wgt")

# Définir des paramètres via des tableaux multidimensionnels

# Transportation cost per ton of raw material r from supplier s to affiliate a
c_trans_sar = np.random.rand(4, 4, 4)

# Transportation cost per ton of grade g from affiliate a to warehouse w
c_trans_awg = np.random.rand(4, 4, 4)
# Transportation cost per ton of grade g from warehouse w to customer k
c_trans_wkg = np.random.rand(4, 10, 4)
# Purchasing Cost per ton of raw material r from supplier s to affiliate a
c_raw_sar = np.random.rand(4, 4, 4)
# Price per ton of prime grade g in affiliate a during time period t
c_prime_agt = np.random.randint(500, 600, size=(4, 4, 6))
# Production cost per ton of transitional material that is formed in transition between grade
g and h in affiliate a during time period t
c_nprime_aght = np.random.rand(4, 4, 4, 6)
# inventory carrying cost per ton of grade g in warehouse w during time period t

```

```

c_inv_wgt = np.random.rand(4, 4, 6)
# Availability of raw material r in tons from supplier s per period which is constant across all periods

lambda_sr = np.random.rand(4, 4) * 1000

# Production capacity of affiliate a during time period t
lambda_prime_at = np.random.rand(4, 6) * 2000
# Amount of transitional material in tons that is formed in transition between grade g and h in affiliate a during time period t
O_aght = np.random.rand(4, 4, 4, 6)
# Quantity needed of raw material r to produce one ton of grade g in affiliate a
beta_arg = np.random.rand(4, 4, 4)
# Demand of grade g for customer k during time period t
D_gkt = np.random.rand(4, 10, 6) * 50

# Définir la fonction objectif
objective = 0

# Première partie : Revenu total
for a in range(4): # 4 个附属工厂
    for g in range(4): # 4 个等级
        for t in range(6): # 6 个时间周期

            objective += c_prime_agt[a, g, t] * X_agt[a, g, t]

# Deuxième partie : coût du transport
# Coûts de transport des matières premières des Suppliers vers les affiliates
for s in range(4): for a in range(4): for r in range(4): for t in range(6):
    objective -= c_trans_sar[s, r, a] * F_sart[s, a, r, t]

# Coûts de transport des produits des usines affiliées aux entrepôts
for a in range(4): for w in range(4): for g in range(4): for t in range(6):
    objective -= c_trans_awg[a, g, w] * F_awgt[a, w, g, t]

# Coûts d'expédition du produit de l'entrepôt au client
for w in range(4): for k in range(10): for g in range(4): for t in range(6):
    objective -= c_trans_wkg[w, k, g] * F_wkgt[w, k, g, t]

# Coût d'achat des matières premières
for s in range(4): for a in range(4): for r in range(4): for g in range(4):

```

```

objective -= c_raw_sar[s, r, a] * F_sart[s, a, r, g]

# Coût de la conversion de niveau
for a in range(4):for g in range(4):for h in range(4):

if h != g: # Signifier la conversion de niveau
for t in range(6):
    objective -= (c_nprime_aght[a, g, h, t] * O_aght[a, g, h, t] * Z_aght[a, g, h, t])

# Coût de détention
for w in range(4): for g in range(4): for t in range(6):
    objective -= c_inv_wgt[w, g, t] * I_wgt[w, g, t]

# Définir la fonction objective sur le modèle
model.setObjective(objective, GRB.MAXIMIZE)

# utiliser le fonction de 'quicksum' pour réaliser les contraintes

# Contrainte1: la quantité totale de matière première r requise pour produire un grade g dans
une filiale a est égale à la somme des matières premières transportées depuis tous les
fournisseurs s vers cette filiale a, pour chaque période t.

model.addConstrs(
    ( gp.quicksum(beta_arg[a, r, g] * X_agt[a, g, t] for g in range(G)) ==
      gp.quicksum(F_sart[s, a, r, t] for s in range(S))
    for a in range(A) for r in range(R) for t in range(T)
    ),name="MaterialBalance")

#Contrainte2: la quantité totale de matière première r transportée depuis un fournisseur
s ne dépasse pas la capacité disponible de ce fournisseur ( $\lambda_{sr}$ ) pour chaque période t.

model.addConstrs(
    ( gp.quicksum(F_sart[s, a, r, t] for a in range(A)) <= lambda_sr[s, r]
      for s in range(S) for r in range(R)for t in range(T) ),name="SupplierCapacity")

#Contrainte3: la quantité totale produite des grades ne dépasse pas la capacité maximale des
filiales pour chaque période

model.addConstrs(
    ( gp.quicksum(X_agt[a, g, t] for g in range(G)) +

```

```

gp.quicksum(O_aght[a, g, h, t] * Z_aght[a, g, h, t]
            for g in range(G) for h in range(G) if h != g) <= lambda_prime_at[a, t]
for a in range(A) for t in range(T) ), name="AffiliateCapacity")
# Contrainte4: un grade peut être produit dans une filiale et une période données seulement si
une configuration pour ce grade est déjà mise en place
model.addConstrs(
    (X_agt[a, g, t] <= lambda_prime_at[a, t] *
    (gp.quicksum(Z_aght[a, g, h, t] for h in range(G) if h != g) + alpha_agt[a, g, t])
    for a in range(A) for g in range(G) for t in range(T)
    ),
    name="ProductionLimit"
)
# Contrainte5: un seul changement ou maintien de configuration est possible par période pour
chaque grade dans une filiale
model.addConstrs(
    (gp.quicksum(Z_aght[a, g, h, t] for h in range(G) if h != g) + alpha_agt[a, g, t] <= 1
    for a in range(A) for g in range(G) for t in range(T)), name="SingleTransitionLimit")
# Contrainte6: Permet qu'une configuration initiale soit activée pour un seul grade dans une
filiale donnée et une période donnée
model.addConstrs(
    (gp.quicksum(alpha_agt[a, g, t] for g in range(G)) == 1
    for a in range(A) for t in range(T)
    ), name="SingleSetupPerPeriod")
# Contrainte7: Vérifie qu'aucune transition vers un grade ne peut avoir lieu à moins que ce
grade ne soit effectivement produit
model.addConstrs(
    ( Z_aght[a, g, h, t] <= X_agt[a, h, t]
    for a in range(A) for g in range(G) for h in range(G) if h != g for t in range(T)
    ), name="TransitionRequiresProduction")
# Contrainte8: Gère la transition des configurations d' une période à l'autre
model.addConstrs(
    (alpha_agt[a, g, t] + gp.quicksum(Z_aght[a, h, g, t] for h in range(G) if h != g) ==

```

```

alpha_agt[a, g, t + 1] + gp.quicksum(Z_aght[a, g, h, t] for h in range(G) if h != g)
    for a in range(A) for g in range(G) for t in range(T - 1) # 注意 t 的范围是 T-1
), name="CarryOverSetup")

# Contrainte9:Empêche les sous-tours inutiles (similaires au problème du voyageur de
commerce) et garantit que chaque grade est configuré au plus une fois par période dans chaque
filiale.N = 10000
model.addConstrs(

    (V_agt[a, g, t] + N * Z_aght[a, g, h, t] - (N - 1) - N * alpha_agt[a, g, t] <= V_agt[a, h, t]
    for a in range(A) for g in range(G) for h in range(G) if h != g for t in range(T)
),name="SubTourElimination")

# Contrainte10: toutes les quantités produites de chaque grade dans chaque filiale pour une
période donnée doivent être transportées vers les entrepôtsmodel.addConstrs(
    ( X_agt[a, g, t] == gp.quicksum(F_awgt[a, w, g, t] for w in range(W))
    for a in range(A) for g in range(G) for t in range(T)
), name="ProductionToWarehouse")

# Contrainte11:l'équilibre des stocks dans chaque entrepôt pour chaque grade à chaque période.
Les quantités reçues (production et inventaire précédent) doivent être égales aux quantités
expédiées aux clients et au stock restant

model.addConstrs(
    ( I_wgt[w, g, t] == (I_wgt[w, g, t - 1] if t > 0 else 0) +
    gp.quicksum(F_awgt[a, w, g, t] for a in range(A)) -
    gp.quicksum(F_wkgt[w, k, g, t] for k in range(K))
    for w in range(W) for g in range(G) for t in range(T)
), name="InventoryBalance")

# Contrainte12:la quantité totale d'un grade g transportée depuis tous les entrepôts
correspond exactement à la demande des clients pour ce grade à chaque période
model.addConstrs(
    ( gp.quicksum(F_wkgt[w, k, g, t] for w in range(W)) == D_gkt[g, k, t]
    for g in range(G) for k in range(K) for t in range(T)
),name="CustomerDemand")

# Commencez à optimiser

model.optimize()

```



## •Résultat

```

D:\python\python.exe "D:\python supply\new.py"
Set parameter Username
Academic license - for non-commercial use only - expires 2025-10-31
Objective function defined and added to the model.
Gurobi Optimizer version 11.0.3 build v11.0.3rc0 (win64 - Windows 11.0 (22631.2))

CPU model: AMD Ryzen 7 6800H with Radeon Graphics, instruction set [SSE2|AVX|AVX2]
Thread count: 8 physical cores, 16 logical processors, using up to 16 threads

Optimize a model with 1520 rows, 2496 columns and 7824 nonzeros
Model fingerprint: 0x77c92c47
Variable types: 2016 continuous, 480 integer (480 binary)
Coefficient statistics:
  Matrix range      [6e-04, 1e+04]
  Objective range   [2e-04, 6e+02]
  Bounds range      [1e+00, 1e+00]
  RHS range         [3e-01, 1e+04]
Presolve removed 22 rows and 176 columns
Presolve time: 0.03s
Presolved: 1498 rows, 2320 columns, 7527 nonzeros
Variable types: 2016 continuous, 304 integer (304 binary)
Found heuristic solution: objective 8752926.3481
  
```

```

Root relaxation: objective 1.044245e+07, 792 iterations, 0.02 seconds (0.01 work units)

  Nodes | Current Node | Objective Bounds | Work
Expl Unexpl | Obj Depth IntInf | Incumbent BestBd Gap | It/Node Time
-----
0 0 1.0442e+07 0 57 8752926.35 1.0442e+07 19.3% - 0s
0 0 1.0442e+07 0 50 8752926.35 1.0442e+07 19.3% - 0s
0 0 1.0442e+07 0 54 8752926.35 1.0442e+07 19.3% - 0s
0 0 1.0442e+07 0 54 8752926.35 1.0442e+07 19.3% - 0s
0 0 1.0441e+07 0 51 8752926.35 1.0441e+07 19.3% - 0s
0 0 1.0441e+07 0 61 8752926.35 1.0441e+07 19.3% - 0s
0 0 1.0441e+07 0 61 8752926.35 1.0441e+07 19.3% - 0s
0 0 1.0441e+07 0 60 8752926.35 1.0441e+07 19.3% - 0s
0 0 1.0441e+07 0 59 8752926.35 1.0441e+07 19.3% - 0s
0 0 1.0441e+07 0 59 8752926.35 1.0441e+07 19.3% - 0s
0 0 1.0441e+07 0 55 8752926.35 1.0441e+07 19.3% - 0s
0 0 1.0441e+07 0 65 8752926.35 1.0441e+07 19.3% - 0s
0 0 1.0441e+07 0 62 8752926.35 1.0441e+07 19.3% - 0s
0 0 1.0440e+07 0 57 8752926.35 1.0440e+07 19.3% - 0s
0 0 1.0440e+07 0 56 8752926.35 1.0440e+07 19.3% - 0s
0 0 1.0440e+07 0 65 8752926.35 1.0440e+07 19.3% - 0s
0 0 1.0440e+07 0 65 8752926.35 1.0440e+07 19.3% - 0s
0 2 1.0440e+07 0 65 8752926.35 1.0440e+07 19.3% - 0s
  
```



```

python supply  Version control  Current File  Run  new  variable.py  objective_function.py  TutoGurobi.ipynb

Run  new
...
0 0 1.0440e+07 0 65 8752926.35 1.0440e+07 19.3% - 0s
0 2 1.0440e+07 0 65 8752926.35 1.0440e+07 19.3% - 0s
H 269 133 8779527.5276 1.0438e+07 18.9% 18.8 1s
H 479 221 8783475.9044 1.0438e+07 18.8% 20.0 1s
H 503 221 8793193.8906 1.0438e+07 18.7% 20.2 1s
H 539 221 8919731.2625 1.0438e+07 17.0% 20.3 1s
H 1012 302 8934891.4601 1.0437e+07 16.8% 24.3 1s
H 1219 316 8938326.4182 1.0436e+07 16.8% 24.4 1s
H 1232 316 8940509.6558 1.0436e+07 16.7% 24.5 1s
H 1240 316 8940693.3299 1.0436e+07 16.7% 24.6 1s
H 1244 318 8940715.2043 1.0436e+07 16.7% 24.6 1s
H 1247 318 8940740.8944 1.0436e+07 16.7% 24.7 1s
H 3943 475 9859105.6684 1.0198e+07 3.44% 24.0 3s
H 4723 362 9865150.1056 1.0198e+07 3.38% 22.0 3s
H 4727 339 9865159.1632 1.0198e+07 3.38% 22.0 3s
H 4739 313 9865160.4847 1.0198e+07 3.38% 21.9 3s
15767 961 infeasible 52 9865160.48 1.0198e+07 3.38% 13.3 5s

Cutting planes:
Learned: 11

Explored 53878 nodes (550847 simplex iterations) in 9.62 seconds (6.69 work units)
Thread count was 16 (of 16 available processors)
python supply > new.py 78:1 CRLF UTF-8 4 spaces D:\python\python.exe

```

```

python supply  Version control  Current File  Run  new  variable.py  objective_function.py  TutoGurobi.ipynb

Run  new
...
H 1232 316 8940509.6558 1.0436e+07 16.7% 24.5 1s
H 1240 316 8940693.3299 1.0436e+07 16.7% 24.6 1s
H 1244 318 8940715.2043 1.0436e+07 16.7% 24.6 1s
H 1247 318 8940740.8944 1.0436e+07 16.7% 24.7 1s
H 3943 475 9859105.6684 1.0198e+07 3.44% 24.0 3s
H 4723 362 9865150.1056 1.0198e+07 3.38% 22.0 3s
H 4727 339 9865159.1632 1.0198e+07 3.38% 22.0 3s
H 4739 313 9865160.4847 1.0198e+07 3.38% 21.9 3s
15767 961 infeasible 52 9865160.48 1.0198e+07 3.38% 13.3 5s

Cutting planes:
Learned: 11

Explored 53878 nodes (550847 simplex iterations) in 9.62 seconds (6.69 work units)
Thread count was 16 (of 16 available processors)

Solution count 10: 9.86516e+06 9.86516e+06 9.85911e+06 ... 8.78348e+06

Optimal solution found (tolerance 1.00e-04)
Best objective 9.865160484612e+06, best bound 9.865160484612e+06, gap 0.00000%

Process finished with exit code 0
python supply > new.py 78:1 CRLF UTF-8 4 spaces D:\python\python.exe

```

•Le programme a obtenu le meilleur résultat ( $9.865 \times 10^6$ )en 9,62 secondes.

Donc, le profit maximum est de  $9.865 \times 10^6$ .

## **2.Répartition du travail**

Yiyang TANG:

- Programmation des parties des variables de décision et des fonctionnelles objectives
- Exécution et débogage du code

Shang NI:

- Programmation des parties des contraintes et des paramètres
- Exécution et débogage du code