Projet ANDROIDE
Production et distribution intégrée

December 9, 2022

Monome: Xinyu HUANG(3803966)

Professeur: M.Pierre Fouilhoux

Contents

Résumé

Dans ce projet, on veut étuidier le problème de Production et Distribution Intégrée.

Tout d'aboard on va chercher à le résoudre en utilisant une heuristique gloutonne *Bin-Packing*.

Ensuite, on va essayer de le résoudre avec les méthodes exacte sans/avec la méthode de *Branch-and-cut*.

On résume le travail en faisant les tests et créant une petite GUI qui facilite la résolution du problème et la visualisation des résultats.

1 Introduction

Le problème de Production et Distribution Intégrée stables est un problème de planification opérationnelle, qui modélise la production d'un fabricant et sa distribution aux clientèles.

Le problème PDI a pour cadre applicatif la gestion des décisions de planification d'un fournisseur (Vendor Managed Inventory) par exemple un producteur de produits frais. Le fournisseur surveille les stocks de ses revendeurs (ou détaillants) et décide d'une politique d'approvisionnement pour chacun de ses revendeurs. Le fournisseur est alors le décideur central qui doit résoudre le problème PDI en ayant une connaissance complète des données (état des stocks, besoins,...) fournies par les revendeurs.

Le problème s'intègre deux problèmes très classiques d'optimisation combinatoire en Rechrche Opérationnelle : le problème de dimensionnement de lots (Lot-Sizing Problem ou LSP) et le problème de tournées de véhicules (Vehicle Routing Problem ou VRP) .

2 Travail réalisé

- Fonctions "outils"
  - Lecture de l'instance
  - Fonctions pour obtenir les sous-tours existant pour une affection des variables donnée.
- Méthode heuristique
  - PLNE pour résoudre le problème de LSP
  - Implémentation de la heuristique "Bin packing" pour résoudre le problème de VRP
- Méthode exacte
  - Implémentation de la méthode exacte avec les inégalités MTZ
  - Implémentation de la méthode exacte avec un algorithme de Branche and Cut
- Interface

2.1 Description des fichiers

- dossier PRP_instances : contenant les fichiers des instances prédéfinis ou crées.
- dossier data : contenant les graphes dessinées pour chaque instances
- read_data.jl : les fichiers contenant les fonctions pour lire un instance et le transformer en une structure facile à utiliser.
- PDI_resolution_heuristique.jl
- PDI_resolution_exacte.jl
- PDI_resolution_BnC.jl
- interface.jl : contenant les codes pour créer l'interface
- tool.jl : contenant les fonctions pour détecter les sous-tours existant pour une affectation des variables donnée

3 Résolution heurisitque

Pour la résolution heuristique, on décompose le problem PDI en deux sous-problème LSP et VRP. Vu que LSP est un problème facile donc on le résoudre directement avec une formulation de PLNE. ET VRP est reconnu comme un problème NP-difficile, donc pour faciliter la résolution, on applique une heuristique "Bin-Packing". Le problem PDI est résoulu en faisant passer les résultats obtenue par PLNE vers l'heuristique Bin-Packing.

3.1 Le Problème de Lot-Sizing (LSP)

La résolution de LSP est très simple en réalisant ces formules en Julia avec le Cplex.

$$
\begin{aligned}
\min \sum_{(i,j)\in A} c_{ij}x_{ij} & \\
\sum_{j=1}^{n} x_{0j} \leq m & \\
\sum_{i=1}^{n} x_{i0} \leq m & \\
\sum_{j=0}^{n} x_{ij} = 1 \qquad & \forall i \in \mathcal{N}_C, \\
\sum_{i=0}^{n} x_{ij} = 1 \qquad & \forall j \in \mathcal{N}_C, \\
w_i - w_j \geq d_i - (Q + d_i)(1 - x_{ij}) \qquad & \forall i \in \mathcal{N}_C, \forall j \in \mathcal{N}_\mathcal{C}, \\
0 \leq w_i \leq Q \qquad & \forall i \in \mathcal{N}_C \\
w_i \in I\!R \qquad & \forall i \in \mathcal{N}_C \\
x_{ij} \in \{0,1\} \qquad & \forall (i,j) \in A.
\end{aligned}
$$

"

3.2 Le problème de tournée de véhicules(VRP)

La résolution du VRP est implémenté par moi-même, le principe est de, à chaque pas de temps, on cherche les résultats obtenus par LSP pour savoir quelles sont les clients à livrer. En partant du noeud initial, on cherche le plus proche voisin possible, de façon à ne pas dépasser la capacité du véhicule. Si un véhicule est sur-chargé, on envoie un deuxième vehicule.....

Algorithm 1 Pin-Packing

```
—-Input —-
Q:Maximum charge capacity of a vehicule
clients_needed_to_visite:clients that x[client,t]==1
q:a array corresponding to demand for each client
—-Input —-
—-Output —-
Objectif_value
—-Output —-
vehicule_counter=0
charge_actual_vehicule=0
partition=Dict()
for client in clients_needed_to_visite do
    if charge_actual_vehicule==0 then
        charge_actual_vehicule=q[client]
        partition[vehicule_counter]=[]
    else if  charge_actual_vehicule+q[client] Q  then
        vehicule_counter++
        charge_actual_vehicule=q[client]
        partition[vehicule_counter]=[]
    else
        charge_actual_vehicule+=q[client]

    end if
    push!(partition[vehicule_counter],client)
end fortsp_partition=Dict()
if length(partition)!=0 then
    for i in 1:vehicule_counter do
        Generate order of partition by distance-¿tsp_partition
    end for
end if
Objectif_value=0
for key, value in tsp_partition do
    for j in 1:length(value)-1 do
        Objectif_value+=distance[j][j+1]
    end for
end for
return Objectif_value
```

3.3 Temps d'exécution et qualité du résultat

Voici les résultats trouvés:

Nom de l'instance	taille	valeur optimal	temps d'exécution
A_014_ABS1_15_1.prp	14	41853.0	29.94 sec
A_020_ABS1_50_1.prp	20	54602.0	28.88 sec
A_025_ABS1_50_1.prp	25	67126.0	29.43 sec
A_030_ABS1_50_1.prp	30	77972.0	29.37 sec
A_050_ABS1_50_1.prp	50	121424.0	30.79 sec
B_200_instance1.prp	200	2.34e6	185.18 sec

Interprétation : On a remarqué que l'approche utilisant l'heuristique Bin-packing est très puissant en terme de vitesse, même les instances de tailles 200 peuvent être résolus en temps raisonnable. On suppose cette approche propose une borne supérieure pour les méthodes exactes.

4 Résolution exacte

Pour la partie concernant les méthodes exactes, deux types de résolutions sont implémentés. Le premier est une PLNE avec les inégalités de MTZ, dont la formulation [1-16] de l'article [1]. L'autre est une PLNE contenant les formules [1-10],[13-16] en generant à chaque affectation des variables une méthodes de branch-and-cut avec les deux formules de l'élimination de sous-tours [17][19].

4.1 PLNE MTZ

Pour pouvoir résoudre le problème intégré, il faut surtout traité le problème de sous-tours, donc on a ajouté un variable wit qui représente la charge du vehicle avant passer par le client i à l'instant t.

Deux contraintes pour l'élimination de sous-tours:

- $w_{it} - w_{jt} \geq q_{it} - \tilde{M}_{it}(1 - x_{ijt}) \qquad \forall (i,j) \in A, \forall t \in T$
- $0 \leq w_{it} \leq Q z_{it} \qquad \forall i \in N_c, \forall t \in T$

4.2 PLNE Branch and cut

Pour pouvoir réaliser un algorithme de Branch-and-cut, il faut que je trouve une méthode pour détecter les sous-tours. Ici on défini les sous-tours comme les circuits qui ne passent pas le noeud 1 ou les noeuds dont les valeurs zit valent 1. J'ai implémenté deux fonctions pour satisfaire cette besoin.

Algorithm 2 detect_soustours

```
1: —-Input ——
2: n: nb de clients
3: xsep: une matrice qui corresponde à une affectation des valeurs de variable x
4: —-Input ——
5: —-Output ——
6: non_soustours : boolean vaut true s'il n'exist pas de sous-tours
7: soustours : ensemble contenant les sous-tours
8: —-Output ——
9: sousTours, his_visite deux ensemble vide
10: non_soustours = true
11: for i le numéro de noeud allant de 2 à nb_max do
12:     if i n'est pas visité then
13:         for i le numéro de noeud allant de 2 à nb_max do
14:             if xsep[i][j] >= 1 then
15:                 if n'a pas trouvé de circuit contenant l'arc i-j allant au noeud 1  then
16:                     non_soustours = false
17:                     ajouter le sous-tours trouvé dans l'ensmble sousTours
18:                     ajouter les noeuds dans le sous-tours dqns l'historique his_visite.
19:                 end if
20:             end if
21:         end for
22:     end if
23: end for
24: return non_soustours, sousTours
```

Principe : Le principe de cette fonction est simple : Pour une affectation des x données, on cherche s'il existe de sous-tour et pour chaque xsep[i][j] valant 1, on cherche s'il est dans un sous-tours en appellant la fonction suivante chercher_tournee.

Algorithm 3 chercher_tournee

```
—-Input ——
n: nb de clients
xsep: une matrice qui corresponde à une affectation des valeurs de variable x
i,j : deux noeuds non visités tel que le valeur xsep[i,j] égal à 1.
—-Input ——
—-Output ——
tournee : le sous-tour trouvé/peut être vide
—-Output ——
tournee = [j]
noeud_de_depart = i
noeud_init_trouvé = false
historique = set[i]
while n'a pas trouvé noeud de départ ou noeud 1 do
    ajouter noeud de depart dans la tournee et la historique
    for i allant de 1 à nb_max do
        if xsep[noeud_de_depart][]>=1 et i n'est pas dans la tournéé then
            noeud_de_depart=j
            on pass à l'itération suivante
        end if
        if noeud_de_depart est le noeud 1 then
            noeud_init_trouvé = true
            on rajoute le noeud dans la trournée
        end if
    end for
end while
return noeud_init_trouvé, tournéé
```

Principe : En lui proposant i,j deux noeuds tel que xij valant 1, cette fonction chercher l'existence de circuit et s'il en existe, détecter si c'est un sous-tours ou une tournée normale passant par le noeud 1.

4.3 Test

4.3.1 Test sur le PLNE MTZ

J'ai commencé avec les instances de grande taille(14/50), la résolution a apparru extrênement lente et donc j'ai décidé de commencer avec les instances de petits tailles et on veut savoir juste à quelle taille de l'instance le PLNE peut-il résoudre de façon "rapide".

Voici les résultats trouvés:

Nom de l'instance	taille	valeur optimal	temps d'exécution
A_005_ABS1_15_z.prp	5	21672.0	24.41 sec
A_006_ABS1_15_1.prp	6	23680.0	273.09 sec
A_007_ABS2_15_4.prp	7	17525.0	1915.77 sec

Interprétation : on peut bien remarquer que pour les instances de taille très petitte(7), la méthode a prise beaucoup de temps. Pour les instances de taille plus grande, il me semble impossible à les résoudre en temps raisonable.

4.3.2 Test sur le PLNE Branch and cut

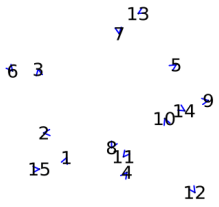
Pour le PLNE avec la méthode Branch and cut, la résolution des instances de taille 14 est instantané, donc on veut étuier le temps de l'exécution sur différents instances de taille 14.

Nom de l'instance	valeur optimal	temps d'exécution
A_014_ABS1_15_1.prp	40345.9	27.23 sec
A_014_ABS1_15_2.prp	40317.0	24.75 sec
A_014_ABS1_15_3.prp	34146.9	7.20 sec
A_014_ABS1_15_4.prp	41655.0	19.61 sec.

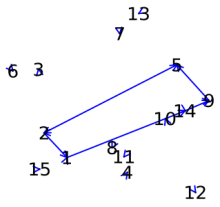
Interprétation:

On a fait plusieurs essais sur les instances de tailles 14 donc notre méthode fonctionne bien sur ces instances. Par contre, le temps de résolution dépend vraiment de l'instance: la résolution de l'instance3 est très rapide, soit de 7.20 secondes. En même temps, la résolution de l'instance2 est relativement lent, soit deux fois plus lent que cela de l'instance3.

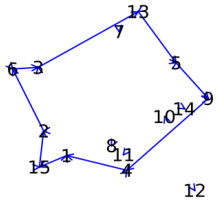
Notre résultat est bien valide et on peut regarder les graphes produits en-dessous, il n'existe pas de sous-tours et la somme des quantités d'approvisionnement est bien inférieur à la capacité de véhicule à chaque instant.



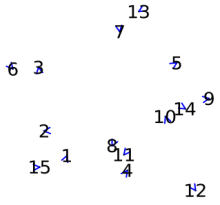
(a) t=1



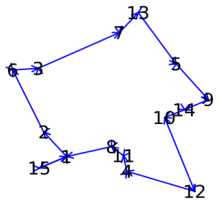
(b) t=2



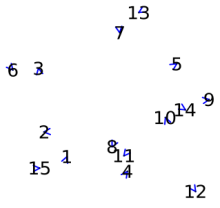
(c) t=3



(d) t=4



(e) t=5



(f) t=6

Figure 1: Les graphes produits par le PLNE Branch_and_cut de l'instance A_014_ABS1_15_1.prp.

Par la suite, j'ai essayé avec les instances de tailles 50, la résolution a l'air très lente et en même temps plus la borne inférieure augmente, plus le valeur de l'augmentation devient faible. Donc on veut essayer les instances de taille entre (14 et 50) pour vérifier la performance de notre méthode.

Nom de l'instance	taille	valeur optimal	temps d'exécution
A_014_ABS1_15_1.prp	14	40345.9	17.01
A_020_ABS1_50_1.prp	20	53048.0	54602.0
A_025_ABS1_50_1.prp	25	63715.0	67126.0
A_030_ABS1_50_1.prp	30	74837.9	121424.0
A_035_ABS1_50_1.prp	30	pas trouvée	très lente

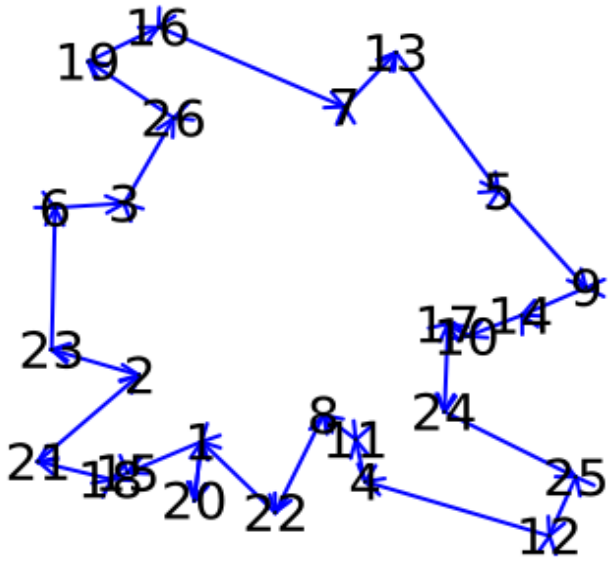


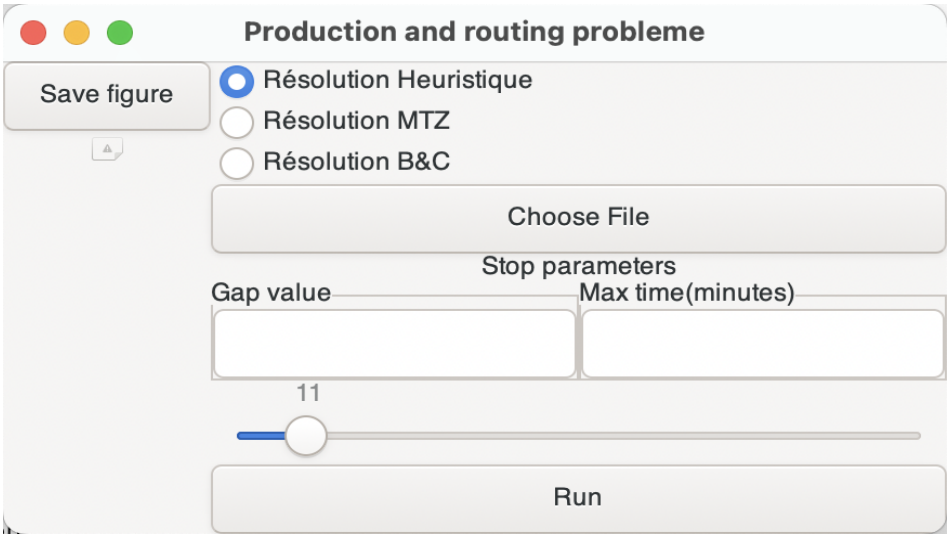
Figure 2: La graphe obtenue avec l'instance A_025_ABS1_50_1.prp pour t=5

Interprétation:

J'ai remarqué qu'à partir des instances de taille 25, le temps d'exécution est devenu très lent, cela peut être raisonné par notre façons de Branch-and-cut tel que le nombre de contraintes croît exponentiellement par rapport à la taille de l'instance.

5 Mode d'emploi de l'interface *PDI calculator*

L'interface *PDI calculator* est une interface pour faciliter à résoudre le problème de PDI.
Affichage :



“
Exécution : à partir de chemin courant, taper **julia interface.py**

Fonctionnalité: résoudre le problème de PDI en utilisant

- 1)Bin Packing
- 2)Résolution exacte MTZ
- 3)Résolution exacte B&C

Données utilisés: Les instances proposés par l'enseigant/crées par moi-même.

Étapes à suivre pour l'exécution

- taper julia interface pour exécuter le fichier
- choisir la méthode pour résoudre le problème
- exécuter l'instance en cliqant sur le bouton ”run”
- inspecter les graphes produits

6 Conclusion

Remarque 1 La valeur minimale des solutions obtenues par les heuristiques est une borne supérieure du problème, donc il est utile de coder plusieurs heuristiques pour trouver une meilleure borne qui peut accélérer la résolution exacte.

Voici les résultats de l'approche heuristique et la méthode exacte avec branch-and cut, on peut bien remarqué que les résultats de l'approche est bien une borne supérieure de notre résultats exacte.

Nom de l'instance	Méthode exacte BnC	Binpacking
A_014_ABS1_15_1.prp	40345.9	41853.0
A_020_ABS1_50_1.prp	53048.0	54602.0
A_025_ABS1_50_1.prp	63715.0	67126.0
A_030_ABS1_50_1.prp	74837.9	77972.0
A_035_ABS1_50_1.prp	pas trouvée	121424.0

Remarque 2 Pour un problème donné, il faut parfois chercher les propriétés du problème(ici, les sous-tours) et parfois, la vitesse des différentes formulations de la PL peut être très différente.

Remarque 3 Le temps de l'exécution pour les méthodes exactes varie largement en augmentant le nombre de clients, même pour les différents instances de même taille. En revanche, pour l'approche heuristique ici, le temps d'exécution augmente peu en ajoutant les clients, cela peut être expliqué par le fait que le problème de LSP est un problème simple et notre résolution du second problème est basé sur un algorithme glouton qui est efficace.

7 Bibliographie

[1] Y. Adulyasak and J-F Cordeau and R. Jans (2015). The production routing pro- blem : A review of formulations and solution algorithms Computers & Operations Research, 55 :141-152.