École Nationale Supérieure
d'Informatique pour l'Industrie et
l'Entreprise

CORO

Projet 2024

Rapport

Gestion des commandes dans un centre de distribution

Élèves :
Mathéo Biard
Colin Coërchon

Enseignants :
Alain Faye
Ana Flávia Macambira

14 avril 2024

Table des matières

1 Introduction

L'objectif de ce projet consiste en l'optimisation de la gestion des commandes dans un centre de distribution. Il faut limiter le déplacement des robots qui amènent les objets et maximiser le nombre de tâche secondaire effectué pour améliorer la gestion des commandes.

Après avoir pris connaissance des informations, nous avons essayé de bien comprendre la modélisation mathématique du problème avant de s'atteler à la suite. En effet, le projet comporte deux parties :

Tout d'abord, nous allons **modéliser ce problème à l'aide de la PLNE** en Julia. Puis nous nous intéresserons à **la décomposition de Dantzig-Wolfe** de ce problème.

2 Résolution par la PLNE

2.1 Définition des variables

C'est la partie la plus importante du projet car une bonne définition des variables peut grandement aider à la résolution (notamment sur le nombre de contraintes à créer). Ainsi pour se faciliter la suite du projet, nous avons pris le temps de bien définir les différentes variables.

Voici donc **les données du problèmes** :

- N : Nombre de produits
- P : Nombre de travailleurs
- R : Nombre d'étagères
- O : Nombre de commandes
- RS : Nombre de produit différents dans une étagère
- $Q = [q_{io}]$: Nombre d'unité i dans la commande o (sous forme de matrice)
- $S = [s_{ir}]$: Nombre d'unité i dans l'étagère r
- $Capa = [C_p]$: Capacité du travailleur p

On notera aussi par ailleurs la décomposition de l'ensemble des commandes $[\![1, O]\!]$ en deux sous-ensembles :

- F : Ensemble des commandes prioritaires

- S : Ensemble des commandes secondaires

Et voici **les variables binaires principales** à optimiser dans notre problème :

- $\forall o \in [\![1, O]\!], \forall p \in [\![1, P]\!],$ $\boxed{x_{op} = \begin{cases} 1 & \text{si le personnel } p \text{ s'occupe de la commande } o, \\ 0 & \text{sinon.} \end{cases}}$
- $\forall r \in [\![1, R]\!], \forall p \in [\![1, P]\!],$ $\boxed{y_{rp} = \begin{cases} 1 & \text{si le personnel } p \text{ utilise l'étagère } r, \\ 0 & \text{sinon.} \end{cases}}$

Dans l'objectif de simplifier la résolution du problème, on définit également 2 autres variables dépendant des variables principales x_{op} et y_{rp} :

- $\forall o \in [\![1, O]\!], \quad v_o = \sum_{p=1}^{P} x_{op}$: Le nombre de personne s'occupant de la commande o
- $\forall r \in [\![1, R]\!], \quad u_r = \sum_{p=1}^{P} y_{rp}$: Le nombre de personne qui utilise l'étagère r

Évidemment, on remarque que ces variables appartiennent à l'ensemble $\{0, 1\}$ puisque les commandes ne sont réalisées par au plus qu'une personne, idem pour les étagères.

v_o et u_r sont donc également des **variables binaires**.

Implémentation en Julia :

```
model = JuMP.Model(Cbc.Optimizer)

N = Data.N
R = Data.R
O = Data.O
P = Data.P

S = Data.S
Q = Data.Q

Capa = Data.Capa
FO = Data.FO
SO = Data.SO

@variable(model, x[1:O, 1:P], Bin)
@variable(model, y[1:R, 1:P], Bin)
@variable(model, 0 <= v[1:O] <= 1)
@variable(model, 0 <= u[1:R] <= 1)
```

2.2 Définition des contraintes

Maintenant que toutes les données et les variables ont été définies, nous allons pouvoir écrire les différentes contraintes associées.

$$\begin{cases} \sum_{p=1}^{P} x_{op} = 1 & o \in F \quad (1) \\ \sum_{p=1}^{P} x_{op} = v_o & o \in S \quad (2) \\ \sum_{p=1}^{P} y_{rp} = u_r & r \in [\![1, R]\!] \quad (3) \\ \sum_{o=1}^{O} x_{op} \leq C_p & p \in [\![1, P]\!] \quad (4) \\ \sum_{r=1}^{R} s_{ir}\, y_{rp} \geq \sum_{o=1}^{O} q_{io}\, x_{op} & p \in [\![1, P]\!],\ i \in [\![1, N]\!] \quad (5) \\ x_{op},\ y_{rp} \in \{0, 1\} \quad (6) \\ v_o,\ u_r \in [0, 1] \quad (7) \end{cases}$$

Avec :

(1) Toutes les commandes prioritaires sont obligatoirement faites.

(2) Nombre de commandes secondaires traitées.

(3) Nombre de personnes qui utilisent l'étagère r.

(4) Aucun travailleur ne doit faire plus de commande que sa capacité.

(5) Chaque commande est complète (le contenu des étagères permet de remplir la commande).

(6) Les variables principales à optimiser (variables binaires)

(7) Les variables secondaires à optimiser. Ces variables sont nécessairement **binaires**, mais on les garde dans $[0, 1]$ pour accélérer le travail du solveur.

Implémentation en Julia :

```
@constraint(model, commande_prioritaire[o in FO], sum(x[o,p] for p in 1:P) == 1)
@constraint(model, commande_secondaire[o in SO], sum(x[o,p] for p in 1:P) == v[o])
@constraint(model, etagere_utilisee[r in 1:R], sum(y[r,p] for p in 1:P) == u[r])
@constraint(model, nombre_commandes[p in 1:P], sum(x[o,p] for o in 1:O) <= Capa[p])
@constraint(model, condition_necessaire_commande[p in 1:P, i in 1:N],
↪  sum(S[i][r]*y[r,p] for r in 1:R) >= sum(Q[i][o]*x[o,p] for o in 1:O))
```

2.3 Fonction objectif

Ainsi nous n'avons plus qu'à minimiser **la fonction objectif** suivante :

$$\min_{x,y} \sum_{r=1}^{R} (|S|+1)u_r - \sum_{o \in S} v_o$$

C'est de l'optimisation lexicographique.

Implémentation en Julia :

```
@objective(model, Min, sum([(length(SO)+1)*u[r] for r in 1:R]) - sum(v[o] for o in SO))
```

3 Premiers résultats

3.1 Présentation des résultats

Dans tous les fichiers du projet, il nous était donné un fichier *Excel* contenant une batterie assez folle de résultat : `resultat_test.xlsx`. Notre objectif a été d'en reproduire certains avec notre algorithme (notamment pour le tester).

À chaque appel de notre algorithme, voici **l'affichage de nos résultats** qui en ressort (exemple pour `N100_R50_O50_RS25` avec 5 travailleurs de Cp = 12). Nous en somme plutôt fier, car cela donne rapidement une bonne visualisation et interprétation des résultats d'une optimisation.

```
julia> include("Projet.jl")
Démarrage de l'optimisation...

Les données du problème sont les suivantes :
N : Nombre de produits = 100
P : Nombre de travailleurs = 5
R : Nombre d'étagères = 50
O : Nombre de commandes = 50
RS : Nombre de produit différents dans une étagère = 25
Capa : Capacité par travailleur = [12, 12, 12, 12, 12]
Nombre de commandes prioritaires = 15
Nombre de commandes secondaires = 35

Les valeurs optimales des variables sont les suivantes :

Travailleur 1 :
```

```
Commandes faites: 2, 7, 9, 10, 12, 15, 18, 29, 35, 43, 49, 50
Utilisation des étagères: 26

Travailleur 2 :
Commandes faites: 1, 3, 4, 6, 17, 22, 34, 44
Utilisation des étagères: 2

Travailleur 3 :
Aucune commande faite.
Aucune étagère utilisée.

Travailleur 4 :
Aucune commande faite.
Aucune étagère utilisée.

Travailleur 5 :
Commandes faites: 5, 8, 11, 13, 14, 16, 27, 30, 42, 45, 46, 48
Utilisation des étagères: 35, 41

Nombre total de commandes secondaires traitées : 17
Nombre total d'étagères utilisées : 4

La valeur de la fonction objectif est : 127.0
Le temps de calcul du solveur est de 25.021 secondes.
```

La majorité des données du problème sont rappelé, et les résultats sont triés selon chaque travailleur du centre de distribution.

Nous avons ensuite construit petit à petit notre ensemble de résultat. En voici un bref récapitulatif dans ce tableau LaTeX :

Instance	Pickers	FO	Solution Value	CPU Time
N100_R50_O50_RS25	5 p. Cp=12	5	79 (13 SO et 2 R)	1.19”
N100_R50_O50_RS25	5 p. Cp=12	10	106 (17 SO et 3 R)	7.36”
N100_R50_O50_RS25	5 p. Cp=12	15	127 (17 SO et 4 R)	24.47”
N100_R50_O50_RS25	5 p. Cp=12	20	135 (20 SO et 5 R)	100.39”
N100_R50_O50_RS25	5 p. Cp=12	25	134 (22 SO et 6 R)	126.15”
N100_R100_O100_RS25	5 p. Cp=12	5	173 (19 SO et 2 R)	241.85”
N100_R100_O100_RS25	5 p. Cp=12	10	247 (26 SO et 3 R)	776.03”
N100_R100_O100_RS25	5 p. Cp=12	15	311 (33 SO et 4 R)	1254.87”
N100_R100_O100_RS25	5 p. Cp=12	20	*Trop de temps d’attente*	
N200_R50_O50_RS25	5 p. Cp=12	5	168 (16 SO et 4 R)	3.93”
N200_R50_O50_RS25	5 p. Cp=12	10	224 (22 SO et 6 R)	6.17”
N200_R50_O50_RS25	5 p. Cp=12	15	231 (21 SO et 7 R)	20.94”
N200_R50_O50_RS25	5 p. Cp=12	20	226 (22 SO et 8 R)	20.04”
N200_R100_O100_RS25	5 p. Cp=12	5	173 (19 SO et 2 R)	7.55”
N200_R100_O100_RS25	5 p. Cp=12	10	333 (31 SO et 4 R)	50.19”
N200_R100_O100_RS25	5 p. Cp=12	15	402 (28 SO et 5 R)	262.84”

TABLE 1 – Résultats de l’optimisation

3.2 Analyse des résultats

Les résultats obtenus, comme présentés dans le Tableau (1), sont très corrects. Chaque instance, *ou presque*, caractérisée par un nombre spécifique de pickers, commandes, et étagères, a été traitée avec succès, montrant des variations dans les valeurs de solution et les temps de calcul en fonction de la complexité.

Les valeurs de solution obtenues et le nombre de commandes secondaires traitées (indiqué par SO dans le tableau) et le nombre d’étagères utilisées (indiqué par R) sont **très proches, sinon identiques, aux résultats fournis dans le fichier** `resultat_test.xlsx` de M. Faye. Cela confirme l’efficacité de notre modèle et sa capacité à reproduire fidèlement les résultats attendus.

Les temps de calcul varient avec la taille et la complexité des instances, mais restent cohérents avec les attentes pour des problèmes de grande taille, ce qui valide la logique de notre approche de traitement.

En conclusion, **notre approche par PLNE est validée comme étant efficace et précise**, offrant des solutions appropriées dans **des délais raisonnables**, adaptés aux exigences des instances traitées.

4 Décomposition simple de Dantzig-Wolfe

4.1 Concept de base

La décomposition de Dantzig-Wolfe est une technique de programmation linéaire pour résoudre des problèmes linéaires de grande taille qui peut être séparée en composantes plus petites. Cette méthode est particulièrement utile pour les problèmes où la structure de blocs est apparente, c'est-à-dire qu'une partie des contraintes ne relie que des variables spécifiques entre elles. En relaxant certaines contraintes qui lient les blocs ensemble, **on crée des sous-problèmes qui peuvent être résolus de manière indépendante**.

4.2 Lien avec le problème de gestion des commandes

Pour le problème donné, nous allons appliquer **la décomposition de Dantzig-Wolfe à la contrainte (5)**, qui impose une condition nécessaire à l'exécution des commandes. Cette relaxation nous permet de diviser le problème **en deux sous-problèmes** distincts. Le premier sous-problème traite de l'affectation des commandes aux travailleurs et à leur capacité, tandis que le second sous-problème traite de l'utilisation des étagères et de la satisfaction des commandes en fonction des stocks disponibles.

Nous divisons donc le problème en deux sous-problèmes :

Sous-problème 1 traite de l'affectation des commandes aux travailleurs :

$$\boxed{\min_{x} \left(-\sum_{o \in S} v_o + \sum_{o=1}^{O} \sum_{p=1}^{P} x_{op} \sum_{i=1}^{N} \alpha_{pi}\, q_{io} \right)} \tag{1}$$

sous contraintes (1), (2), (4).

Sous-problème 2 concerne l'utilisation des étagères et la satisfaction des commandes :

$$\boxed{\min_{y} \left(\sum_{r=1}^{R} (|\mathcal{S}| + 1)\, u_r + \sum_{r=1}^{R} \sum_{p=1}^{P} y_{rp} \sum_{i=1}^{N} (-\alpha_{pi})\, s_{ir} \right)} \tag{2}$$

sous contrainte (3).

Les variables α_{pi} représentent les multiplicateurs de Lagrange introduits suite à la relaxation lagrangienne de la contrainte (5).

4.3 Implémentation en Julia

Premièrement, nous avons défini deux sous-fonctions dans l'objectif d'implémenter cette décomposition de Dantzig-Wolfe. Il y a en effet deux sous-problèmes qui sont apparus

lors de la relaxation de la contrainte (5).

Nous avons ainsi créé les deux fonctions `configure_subproblem_1` et `configure_subproblem_2`.

Implémentation en Julia :

```
function configure_subproblem_1!(model, alpha)
    @variable(model, x[1:O, 1:P], Bin)
    @variable(model, 0 <= v[1:O] <= 1)

    @constraint(model, commande_prioritaire[o in FO], sum(x[o,p] for p in 1:P) == 1)
    @constraint(model, commande_secondaire[o in SO], sum(x[o,p] for p in 1:P) ==
    ↪  v[o])
    @constraint(model, nombre_commandes[p in 1:P], sum(x[o,p] for o in 1:O) <=
    ↪  Capa[p])

    @objective(model, Min, -sum(v[o] for o in FO) + sum(alpha[p,i] * Q[i,o] *
    ↪  x[o,p] for o in 1:O, p in 1:P, i in 1:N))
end

function configure_subproblem_2!(model, alpha)
    @variable(model, y[1:R, 1:P], Bin)
    @variable(model, 0 <= u[1:R] <= 1)

    @constraint(model, etagere_utilisee[r in 1:R], sum(y[r,p] for p in 1:P) == u[r])

    @objective(model, Min, sum((length(SO) + 1) * u[r] for r in 1:R) +
    ↪  sum(-alpha[p,i] * S[i,r] * y[r,p] for r in 1:R, p in 1:P, i in 1:N))
end
```

Avec l'aide du cours de Mme. Ana Flávia Macambira et de la description du projet par M. Faye, nous avons essayé de construire un **squelette de la fonction** responsable de la décomposition de Dantzig-Wolfe souhaitée.

Voici ci-dessous la boucle principale et cruciale de la fonction :

Implémentation en Julia :

```
# Boucle principale pour l'ajustement des multiplicateurs de Lagrange
for iteration in 1:max_iterations
    # Résolution du sous-problème 1
    optimize!(sp1_model)
    sp1_solution = ...
    sp1_obj_value = objective_value(sp1_model)
```

```
7
8       # Résolution du sous-problème 2
9       optimize!(sp2_model)
10      sp2_solution = ...
11      sp2_obj_value = objective_value(sp2_model)
12
13      # Mise à jour de la borne inférieure
14      current_lower_bound = sp1_obj_value + sp2_obj_value
15      if current_lower_bound > lower_bound
16          lower_bound = current_lower_bound
17      end
18
19      # Ajustement des multiplicateurs de Lagrange alpha
20      adjust_alpha!(alpha, sp1_solution, sp2_solution, eta, step_size)
21
22      # Vérification de la convergence et ajustement du step_size si nécessaire
23      # ...
24  end
```

4.4 Difficultés rencontrées lors de l'implémentation

Malgré de longues heures de travail et un engagement très sérieux dans le processus d'implémentation, **nous avons rencontré plusieurs défis notables qui ont empêché la réalisation de cette décomposition de Dantzig-Wolfe** en utilisant le langage de programmation Julia.

- Premièrement, la gestion des interactions entre les sous-problèmes et le problème maître a posé un défi majeur. Le mécanisme de mise à jour des multiplicateurs de Lagrange α_{pi} reste assez flou.

  Nous avons envisagé l'utilisation d'algorithmes de descente de gradient pour optimiser les α_{pi}, mais il n'était pas clair si cette approche était conforme à l'esprit de la décomposition de Dantzig-Wolfe.

- Deuxièmement, la compréhension et l'utilisation correcte des coefficients η_1 et η_2 ont été problématiques. Bien que nous ayons saisi leur importance théorique dans la formation de la solution convexe, l'implémentation pratique de ces variables duales en Julia a présenté des difficultés significatives.

- En outre, et peut-être le facteur le plus significatif, le cours dispensé par Mme. Macambira, bien qu'instructif, était complexe et parfois difficile à suivre, en grande partie à cause **de la barrière de la langue** et du format en visioconférence.

Malgré une participation active, je ne pense pas avoir pleinement assimilé les notions clés, jj'en suis désolé. Et cela a sans doute amplifié les défis liés à l'implémentation de la décomposition de Dantzig-Wolfe.

5 Conclusion

Dans le cadre de ce projet, nous avons d'abord abordé le problème en le comprenant et en l'analysant du mieux possible, avant de le résoudre par programmation linéaire en nombres entiers (PLNE) en utilisant Julia. L'implémentation de cette solution a été une réussite, et nous avons obtenu des résultats très corrects et similaires avec ceux donnés dans le fichier `resultat_test.xlsx`.

Cependant, notre tentative de mettre en œuvre la décomposition de Dantzig-Wolfe n'a pas abouti comme prévu. Cette partie du projet nous a posé des défis significatifs, et nous aurions bénéficié de travailler davantage ces aspects en travaux dirigés avec M. Faye. Quelques séances de TDs supplémentaires auraient peut-être éclairées certains des problèmes techniques que nous avons rencontrés, et nous auraient permis de mieux comprendre les mécanismes sous-jacents nécessaires à une implémentation réussie.

PS : Après avoir discuté avec presque tous les autres groupes, nous avons constaté que nous n'étions pas les seuls à éprouver des difficultés avec la décomposition de Dantzig-Wolfe. Bien que cela ne serve pas d'excuse, cela met en lumière la réelle complexité de la tâche qui nous était assignée.

Et très honnêtement, nous serions très curieux d'avoir accès à la correction une fois que tous les autres groupes auront rendus leur projet.