Rapport des TPs 1-2
Recherche Opérationnelle

SADURNI Thomas
ROUX Thibault

Département Sciences du Numérique - Filière Image et Multimédia
2020-2021

Contents

List of Figures

1 Introduction

Ce rapport rend compte des séances de TP de Recherche Opérationnelle.
Parmi les problématiques d'optimisation émergeant en e-commerce, se trouve l'affectation de commandes de clients aux magasins ou entrepôts, compte-tenu des coûts associés à la livraison des colis, à la préparation des commandes et à la gestion des différents stocks. Nous nous intéresserons d'abord à la gestion de fluides émanant de demandes avec leurs coûts unitaires, en respectant les stocks de chaque entrepôt, et le prix d'expédition de colis sera pris en compte dans le dernier cas particulier.

Nous nous intéresserons ensuite, dans le cas d'un problème plus général, à l'affectation de commandes et tournées de véhicules pour différents magasins d'une même enseigne.

Tout au long de ces TP, nous avons utilisé Julia/JuMP pour modéliser nos problèmes.

2 Cas Particuliers

Comme expliqué en introduction, ces trois cas particuliers traitent du problème d'optimisation de commandes, par des clients, de fluides à des entrepôts. Chaque fluide a un prix unitaire, chaque entrepôt a une capacité de stockage, et enfin chaque magasin facture les coûts d'expéditions d'un colis associé à une demande (pour le cas particulier 3).

Nous avons voulu être aussi **génériques** que possible, c'est pourquoi nous avons ajouté différentes données dans notre code, à savoir :

- le nombre de fluides Nf
- le nombre d'entrepôts (ou magasins) Ne
- le nombre de demandes Nd

Nous pouvons ainsi modifier les données, et leur taille sans modifier notre code.

2.1 Cas Particulier 1

2.1.1 Variables et domaine

Notre unique variable est un tableau à deux dimensions (i,j), représentant la quantité de fluide i vendue par l'entrepôt j. Ces quantités sont positives ou nulles.

```
# define variables
@variable(model, quantites[1:Nf,1:Ne] >= 0)
```

2.1.2 Fonction objectif

La fonction à minimiser est le coût total, à savoir la somme sur tous les fluides et tous les entrepôts, du prix du fluide dans l'entrepôt, multiplié par la quantité vendue associée.

```
# define objective function
@objective(model, Min, sum(sum(p[f,e]*quantites[f,e] for f in 1:Nf) for e in 1:Ne))
```

2.1.3 Contraintes

Les stocks n'étant pas illimités, nous devons ajouter des contraintes pour les respecter. De plus, pour chaque fluide les quantités distribuées doivent être égales aux demandes des clients. Ces deux contraintes se traduisent sous le forme suivante:

Le premier cas particulier est ainsi modélisé.

```julia
# define constraints
for f in 1:Nf
    @constraint(model, sum(quantites[f,e] for e in 1:Ne) == sum(d[f,j] for j in 1:Nd))
    for e in 1:Ne
        @constraint(model, quantites[f,e] <= s[f,e])
    end
end
```

Nous pouvons le tester avec les données du sujet, qui sont les suivantes (et les mêmes pour les trois cas particuliers) :
Les données de l'énoncé :

```julia
# data
Nf = 2 # nombre de fluides
Ne = 3 # nombre d'entrepôts
Nd = 2 # nombre de demandes
p = [1 2 3;1 3 2] # prix de vente
s = [2.5 1 2;1 2 1] # stocks
d = [2 1;0 3] # demandes
```

```
Solution obtenue:
        coût total = 13.0
        quantité de fluide 1 à acheter dans l'entrepôt 1 = 2.0
        quantité de fluide 1 à acheter dans l'entrepôt 2 = 1.0
        quantité de fluide 1 à acheter dans l'entrepôt 3 = 1.0
        quantité de fluide 2 à acheter dans l'entrepôt 1 = 1.0
        quantité de fluide 2 à acheter dans l'entrepôt 2 = 1.0
        quantité de fluide 2 à acheter dans l'entrepôt 3 = 1.0
```

Figure 1: Solution obtenue pour le cas particulier 1

2.2 Cas Particulier 2

Le cas particulier 2 est le PLNE du cas particulier 1, la seule différence est le fait que les quantités sont désormais entières :

```julia
# define variables
@variable(model, quantites[1:Nf,1:Ne] >= 0, Int)
```

La solution obtenue est la suivante :

```
Solution obtenue:
        coût total = 13.0
        quantité de fluide 1 à acheter dans l'entrepôt 1 = 2.0
        quantité de fluide 1 à acheter dans l'entrepôt 2 = 1.0
        quantité de fluide 1 à acheter dans l'entrepôt 3 = 1.0
        quantité de fluide 2 à acheter dans l'entrepôt 1 = 1.0
        quantité de fluide 2 à acheter dans l'entrepôt 2 = 1.0
        quantité de fluide 2 à acheter dans l'entrepôt 3 = 1.0
```

Figure 2: Solution obtenue pour le cas particulier 2

On remarque que les solutions obtenues pour l'exemple du sujet pour le PL et le PLNE associé sont assez proches. Nous sommes conscients que ce n'est pas toujours le cas et que cela dépend du problème.

2.3 Cas Particulier 3

2.3.1 Variables

Le dernier cas particulier ajoute la gestion des coûts d'expédition des colis des magasins aux clients.
Les coûts d'expédition sont enregistrés dans la donnée exp.

Pour répondre au problème, nous avons ajouté une dimension à la variable $quantites$, pour prendre en compte le numéro de la demande, ce que nous ne faisions pas dans les cas particuliers précédents. Nous avons également ajouté une variable $offre$, qui est un tableau à 3 dimensions de Binaire (0 ou 1). $offre[dem, e]$ doit valoir 1 si l'entrepôt e vend des produits pour la demande dem, et 0 sinon.

2.3.2 Fonction objectif

```
@objective(model, Min, sum(sum(sum(p[f,e]*quantites[f,e,dem] for f in 1:Nf) for e in 1:Ne) for dem in 1:Nd) # produits
                     + sum(sum(offre[dem,e]*exp[dem,e] for dem in 1:Nd) for e in 1:Ne)) # expédition
```

2.3.3 Contraintes

```
for f in 1:Nf
        for dem in 1:Nd
            # la quantité vendue est bien égale à celle demandée
            @constraint(model, sum(quantites[f,e,dem] for e in 1:Ne) == d[f,dem])
        end
    end

for e in 1:Ne
    for dem in 1:Nd
        # Methode Big M, pour forcer offre à 0 quand le membre de droite est nul, et à 1 sinon
        @constraint(model,somme_stocks*offre[dem,e]>=sum(quantites[f,e,dem] for f in 1:Nf))
    end
    for f in 1:Nf
        # les quantités vendues doivent être inférieures aux stocks des magasins
        @constraint(model, sum(quantites[f,e,dem] for dem in 1:Nd) <= s[f,e])
    end
end
```

Enfin, la solution obtenue avec les données du problème est :

```
Solution obtenue:
        coût total = 17.0
        quantité de fluide 1 à acheter dans l'entrepôt 1 pour la demande 1 = 1.0
        quantité de fluide 1 à acheter dans l'entrepôt 1 pour la demande 2 = 1.0
        quantité de fluide 1 à acheter dans l'entrepôt 2 pour la demande 1 = 1.0
        quantité de fluide 1 à acheter dans l'entrepôt 2 pour la demande 2 = 0.0
        quantité de fluide 1 à acheter dans l'entrepôt 3 pour la demande 1 = 1.0
        quantité de fluide 1 à acheter dans l'entrepôt 3 pour la demande 2 = 0.0
        quantité de fluide 2 à acheter dans l'entrepôt 1 pour la demande 1 = 0.0
        quantité de fluide 2 à acheter dans l'entrepôt 1 pour la demande 2 = 1.0
        quantité de fluide 2 à acheter dans l'entrepôt 2 pour la demande 1 = 0.0
        quantité de fluide 2 à acheter dans l'entrepôt 2 pour la demande 2 = 1.0
        quantité de fluide 2 à acheter dans l'entrepôt 3 pour la demande 1 = 0.0
        quantité de fluide 2 à acheter dans l'entrepôt 3 pour la demande 2 = 1.0
        Offre de l'entrepôt 1 pour la demande 1 = 1.0
        Offre de l'entrepôt 1 pour la demande 2 = 1.0
        Offre de l'entrepôt 2 pour la demande 1 = 1.0
        Offre de l'entrepôt 2 pour la demande 2 = 1.0
        Offre de l'entrepôt 3 pour la demande 1 = 1.0
        Offre de l'entrepôt 3 pour la demande 2 = 1.0
```

Figure 3: Solution obtenue pour le cas particulier 3

3 Problème Général

L'objectif du problème général est de minimiser les trajets de livraisons de commandes pour les livreurs de magasins. Le PLNE comporte les données suivantes : les demandes à satisfaire, les niveaux de stocks des différents magasins et le temps de trajet entre les différents sites.

3.1 Variables

Nous introduisons trois variables pour ce problème:

1. La première est *arc_emprunte*, une variable binaire, qui s'initialise à 1 si l'arc est emprunté par le livreur, à 0 sinon.
2. La deuxième représente les *quantités* pour la demande d'un produit dans un certain magasin. Ces quantités sont nécéssairement positives.
3. La dernière *ordre* qui correspond à l'ordre de l'arc emprunté.

```
# variables
@variable(model,arc_emprunte[1:nb_noeuds,1:nb_noeuds,1:nb_mag],Bin)
@variable(model, quantites[1:nb_mag,1:nb_dem,1:nb_prod] >= 0, Int)
@variable(model, 0<=ordre[1:nb_noeuds,1:nb_noeuds+1,1:nb_mag]<=2*nb_noeuds,Int)
```

3.2 Fonction objectif

Nous devons minimiser les trajets du livreur, la fonction objective est donc assez simple, il faut juste penser à rajouter notre variable binaire *arc_emprunte* que l'on multiple à la distance entre chaque site.

```
# fonction objectif
@objective(model, Min, sum(sum(sum(arc_emprunte[i,j,m]*R[i][j] for i in 1:nb_noeuds) for j in 1:nb_noeuds) for m in 1:nb_mag))
```

3.3 Contraintes

Les contraintes sont clairement enoncées dans l'énoncé du sujet du TP, nous les rappelons ci dessous :

1. chaque magasin dispose de son propre livreur/camion qui sera en charge de livrer en une seule tournée tous les produits qui proviennent de son magasin (pas de limite de capacité sur les camions).
2. pour chaque magasin qui expédie au moins 1 produit, son livreur/camion débute sa tournée au magasin, visite une seule fois chacun des clients qu'il doit servir et retourne au magasin en fin de tournée.
3. chaque commande doit être satisfaite en totalité
4. une commande peut être satisfaite par un unique magasin qui livre tous les produits qui la composent, ou alors par plusieurs magasins, qui fournissent chacun une partie des produits
5. aucun magasin ne peut faire livrer plus de produits qu'il n'en possède en stock

Cette dernière contrainte, la plus simple à modéliser, est similaire aux cas particuliers : la somme des *quantites* pour chaque produit, chaque demande, chaque magasin doit etre inférieure ou égale à la *matrice_S* représentant l'ensemble des stocks de produit dans les magasins.

De la même façon, ces *quantites* doivent être égales à la *matrice_Q* représentant l'ensemble des quantitée de produit dans les différentes commandes. Nous les modélisons comme ceci :

```
    for p in 1:nb_prod
        for d in 1:nb_dem
            # Contrainte 3. Chaque commande doit être satisfaite en totalité
            @constraint(model, sum(quantites[m,d,p] for m in 1:nb_mag) == matrice_Q[d][p])
        end
        for m in 1:nb_mag
            # Contrainte 5. Quantites vendues <= Stocks des magasins
            @constraint(model, sum(quantites[m,d,p] for d in 1:nb_dem) <= matrice_S[m][p])
        end
    end
```

Nous procédons encore une fois avec la majoration du BigM avec la somme de *matrice_S* (stocks) pour notre variable *quantites*. Enfin, nous mettons à 1 notre variable binaire *arc_emprunte* si le livreur passe par un site, à 0 sinon.
De plus, un livreur ne doit pas pas s'arreter à un autre magasin. L'ensemble des contraintes est donc disponible ci dessous, avec une explications commentée pour chacune d'entre elles.

```
for m in 1:nb_mag
    # Contrainte 2.
    # Le livreur commence sa tournée au magasin
    @constraint(model, sum(arc_emprunte[m,j,m] for j in nb_mag+1:nb_noeuds) == 1 )
    # Le livreur finit sa tournée au magasin
    @constraint(model, sum(arc_emprunte[i,m,m] for i in nb_mag+1:nb_noeuds) == 1 )
    # Le livreur d'un magasin ne se déplace pas jusqu'à un autre magasin
    @constraint(model, sum(sum(arc_emprunte[i,j,m] for j in 1:m-1) for i in 1:nb_noeuds ) == 0)
    @constraint(model, sum(sum(arc_emprunte[i,j,m] for j in m+1:nb_mag) for i in 1:nb_noeuds ) == 0)
    # Le livreur d'un magasin ne se déplace pas à partir d'un autre magasin
    @constraint(model, sum(sum(arc_emprunte[i,j,m] for i in 1:m-1) for j in 1:nb_noeuds ) == 0)
    @constraint(model, sum(sum(arc_emprunte[i,j,m] for i in m+1:nb_mag) for j in 1:nb_noeuds ) == 0)
    for i in 1:nb_noeuds
        # Le livreur ne fait pas du sur place
        @constraint(model, arc_emprunte[i,i,m] == 0)
    end
    for i in nb_mag+1:nb_noeuds
        for j in 1:nb_noeuds
            # Le livreur doit passer par un noeud pour pouvoir en partir => ordre
            @constraint(model, 1+ordre[m,i]<=ordre[m,j] + nb_noeuds*(1-arc_emprunte[i,j,m]))
        end
    end
    for d in 1:nb_dem
        # Le livreur part de chaque noeud client et seulement les noeuds clients
        @constraint(model, somme_stocks*sum(arc_emprunte[nb_mag+d,j,m] for j in 1:nb_noeuds)>= sum(quantites[m,d,p] for p in 1:nb
        @constraint(model, sum(arc_emprunte[nb_mag+d,j,m] for j in 1:nb_noeuds)<= sum(quantites[m,d,p] for p in 1:nb_prod))
        # Le livreur arrive vers chaque noeud client et seulement vers les noeuds clients
        @constraint(model, somme_stocks*sum(arc_emprunte[i,nb_mag+d,m] for i in 1:nb_noeuds)>= sum(quantites[m,d,p] for p in 1:nb
        @constraint(model, sum(arc_emprunte[i,nb_mag+d,m] for i in 1:nb_noeuds)<= sum(quantites[m,d,p] for p in 1:nb_prod))

    end
end
```

3.4 Résultats

Les temps d'optimisation ne sont pas très rapides, mais voici les résultats pour différents fichiers :

```
Solution obtenue:
distance totale = 10221.0
        distance parcourue par le livreur du magasin 1 = 2410.0
        distance parcourue par le livreur du magasin 2 = 628.0
        distance parcourue par le livreur du magasin 3 = 747.0
        distance parcourue par le livreur du magasin 4 = 894.0
        distance parcourue par le livreur du magasin 5 = 1483.0
        distance parcourue par le livreur du magasin 6 = 414.0
        distance parcourue par le livreur du magasin 7 = 1595.0
        distance parcourue par le livreur du magasin 8 = 1028.0
        distance parcourue par le livreur du magasin 9 = 318.0
        distance parcourue par le livreur du magasin 10 = 704.0
```

Figure 4: Solution obtenue pour 5 demandes 10 magasins et 10 produits version 0

```
Solution obtenue:
distance totale = 12718.0
        distance parcourue par le livreur du magasin 1 = 1487.0
        distance parcourue par le livreur du magasin 2 = 1022.0
        distance parcourue par le livreur du magasin 3 = 1124.0
        distance parcourue par le livreur du magasin 4 = 2351.0
        distance parcourue par le livreur du magasin 5 = 778.0
        distance parcourue par le livreur du magasin 6 = 1340.0
        distance parcourue par le livreur du magasin 7 = 2087.0
        distance parcourue par le livreur du magasin 8 = 1473.0
        distance parcourue par le livreur du magasin 9 = 254.0
        distance parcourue par le livreur du magasin 10 = 802.0
```

Figure 5: Solution obtenue pour 5 demandes 10 magasins et 10 produits version 1

```
Solution obtenue:
distance totale = 11001.0
        distance parcourue par le livreur du magasin 1 = 468.0
        distance parcourue par le livreur du magasin 2 = 704.0
        distance parcourue par le livreur du magasin 3 = 1446.0
        distance parcourue par le livreur du magasin 4 = 940.0
        distance parcourue par le livreur du magasin 5 = 1510.0
        distance parcourue par le livreur du magasin 6 = 2142.0
        distance parcourue par le livreur du magasin 7 = 856.0
        distance parcourue par le livreur du magasin 8 = 368.0
        distance parcourue par le livreur du magasin 9 = 2009.0
        distance parcourue par le livreur du magasin 10 = 558.0
```

Figure 6: Solution obtenue pour 5 demandes 10 magasins et 10 produits version 2

4 Conclusion

Ce projet concret nous a permis, à notre échelle, de comprendre la gestion de livraison de grands magasins. Minimiser des prix, des trajets, des durées est essentiel dans la vie courante et ce projet nous a permis de comprendre l'utilité de tels programmes dans le commerce. C'est un projet intéréssant et avec un objectif concret que nous avons apprécié.