Rapport pour le projet de MOGPL

Pierre Mahé & François Thiré

24 novembre 2017

Table des matières

1 Première modélisation du problème

1.1 Question 1

La définition du programme linéaire $\mathcal{P}_0$ est donnée dans l'encadré ci-dessous :

Variables :

$$\underbrace{x_{i,j}}_{\text{Vaut 1 si l'agent } a_i \text{ reçoit le bien } b_j} \in \mathbb{B}\ 1 \leq i, j \leq n$$

Fonction objectif :

$$\textbf{max}\ \frac{1}{n} \sum_{1 \leq i,j \leq n} u_{i,j} x_{i,j}$$ (avec $u_{i,j}$ les coefficients d'utilité du bien i pour l'agent j)

Contraintes :

$$\sum_{i=1}^{n} x_{i,j} = 1,\ 1 \leq j \leq n \qquad \textit{1 agent par bien}$$

$$\sum_{j=1}^{n} x_{i,j} = 1,\ 1 \leq i \leq n \qquad \textit{1 bien par agent}$$

Programme linéaire $\mathcal{P}_0$

1.2 Question 2

Vous trouverez le code *Python* du modèle $\mathcal{P}_0$ dans le répertoire *Python/modelisation_P0*. L'implémentation du modèle et la génération des tests se fait dans le fichier **P0.py**. Le programme peut se lancer soit en utilisant la commande *Python2.7* s'il est installé ou bien en utilisant *gurobi.sh*. Afin de connaître les différentes options disponibles, il est possible d'utiliser l'option **-h**.
En particulier pour lancer le programme avec comme taille $n = 100$ et $M = 1000$ on utilisera la commande suivante :

```
gurobi.sh P0.py -n 100 -M 1000
```

Si vous choisissez d'utiliser les options pour enregistrer le modèle et écrire la solution dans un fichier, alors le programme va créer respectivement deux sous-dossiers [1] *modele* et *solutions* qui contiendra ces fichiers. Ces remarques s'appliquent aussi bien pour les autres implémentations demandées par le projet [2].

1.3 Question 3

Nous avons automatisé la création des tableaux par un script *bash* que vous trouverez dans le répertoire *Python/stats/question_3*. Ce programme va générer des fichiers *csv* qui pourront ensuite être importés dans un fichier LaTeXen utilisant le package *csvsimple*. Voici donc les résultats trouvés :

n	temps moyen	satisfaction moyenne	satisfaction minimum	satisfaction maximum
10	0	7.69641	4.68525984078	9.88840970812
50	0.022	8.91893	6.98608557939	9.96291213496
100	0.07	9.2037	7.37051412521	9.97975949946
500	5.837	9.64008	8.64648636468	9.99484313545
1000	40.595	9.74462	9.1191143598	9.99890280968

Résultats lorsque $M = 10$

n	temps moyen	satisfaction moyenne	satisfaction minimum	satisfaction maximum
10	0	7.72088	4.75539217734	9.75454874819
50	0.02	8.85833	7.024114648	9.94691281392
100	0.07	9.18897	7.69339053482	9.98682579342
500	7.649	9.63927	8.747011081	9.99636220016
1000	36.211	9.74641	9.17666459274	9.99687184259

Résultats lorsque $M = 100$

n	temps moyen	satisfaction moyenne	satisfaction minimum	satisfaction maximum
10	0	7.69587	4.77701102134	9.98019485689
50	0.021	8.90054	7.02335311664	9.98557572002
100	0.07	9.18693	7.06922191129	9.98258994542
500	5.844	9.64279	8.78818848224	9.99556314516
1000	33.023	9.74541	9.0636041877	9.99888703089

Résultats lorsque $M = 1000$

A noter que les temps indiqués sont les temps indiqués par gurobi.

1. à partir du dossier courant
2. A part les modélisations utilisant pygraph où il faut utiliser la commande *python2.7*

2 Approche égalitariste

2.1 Question 4

Avant de parler de la construction du graphe, nous mentionnons que cette question nous a posé quelques problèmes. En effet, le sujet ne spécifiait pas une version spécifique de Python a utiliser. Le projet a donc d'abord été implémenté en utilisant la version **3.4** de Python. Seulement, il se trouve que *gurobi* n'est pas compatible avec cette version. De plus, la librairie **pygraph** posait problème avec python **3.4** si nous voulions utiliser *gurobi*. Une première solution a donc été d'utiliser la librairie *graph_tools*. L'implémentation se trouve dans *Python/modelisation_graph/graph_tools.py*. Cependant cette implémentation souffre de deux problèmes :

— Les capacités utilisent directement les coefficients ;

— L'algorithme n'est pas au point, en particulier il n'utilise pas une recherche dichotomique ;

L'implémentation qui nous intéresse se situe à l'adresse *Python/modelisation_graph/approche_egalitariste.py*. La suite se décompose en deux parties. Dans une première partie on explicite la construction du graphe associé au problème ainsi que le dictionnaire de capacité associé au graphe. Et dans une seconde partie, nous donnons explicitement l'algorithme qui permet de résoudre le problème souhaité.

2.1.1 Description du graphe :

Soit n le nombre d'agents ainsi que le nombre de biens à répartir. Soit $u_{i,j}$ l'utilité du bien j pour l'agent i. On fait l'hypothèse que les coefficients de la matrice $u_{i,j}$ soient distincts [3]. On construit le graphe orienté suivant : $G = (V, E)$ où

— $V = \{s, a_0, a_1, \ldots, u_{n-1}, o_0, o_1, \ldots, o_{n-1}, t\}$

— $\forall i, j \in \{1, \ldots, n\}$

  — $(a_i, o_j) \in E$

  — $(s, a_i) \in E$

  — $(o_j, t) \in E$

3. L'hypothèse n'est pas très forte car il suffit sinon de tirer au hasard un $\varepsilon > 0$ pour différencier les coefficients égaux

Il reste à décrire les capacités sur chaque arc.

$$\begin{aligned}
\forall i,j \in \{1,\dots,n\},\ & c(a_i;o_j)=1 \text{ si } u_{i,j}-\lambda>0\\
\forall i,j \in \{1,\dots,n\},\ & c(a_i;o_j)=0 \text{ sinon}\\
\forall i \in \{1,\dots,n\},\ & c(s;a_i)=1\\
\forall j \in \{1,\dots,n\},\ & c(o_j;t)=1
\end{aligned}$$

2.1.2 Justification de la construction du graphe

En construisant le graphe de cette façon, on remarque qu'un flot maximal ne correspond pas forcément à un problème d'affectation. En effet, rien ne garantit que le flot sera composé qu'avec des capacités entières [4]. Cependant on a la propriété suivante [5] :

Proposition 1. *Le graphe $G=(V,E)$ construit précédemment à un flot maximal si et seulement s'il existe un flot maximal à coefficient entier, i.e. le coefficient de chaque arc est soit* 0 *soit* 1*.*

Démonstration. Le sens réciproque est évident. Il suffit de montrer le sens direct. Par la suite on fait l'hypothèse que chaque agent a reçu au moins un objet et réciproquement. Par maximalité, tous les $c(o_j,t)$ et les $c(s,a_i)$ sont à 1. Le seul cas intéressant se situe quand un objet est distribué entre plusieurs agents. Soit $\mathcal{F}$ un float maximum. Soit o un tel objet et a_k les agents assignés à o. Alors par maximalité du flot on a $\sum_{k\in K} c_{\mathcal{F}}(a_k,o)=1$. La construction consiste à choisir un agent k parmi les a_k et lui assigner cet objet en mettant sa capacité dans ce nouveau flot à 1. Par construction du graphe, on peut répartir les capacités des arc $(a_{-k};o)$ sur les autres objets assignés à l'agent k (on comble les trous). On peut itérer ce processus jusqu'à ce que toutes les arêtes soient des coefficients entiers. □

Cette construction ne garantit en rien la maximalité du flot vis à vis des utilités des agents. C'est l'algorithme décrit dans la partie suivante qui va s'en assurer.

2.1.3 Description de l'algorithme :

On ne prouve pas la correction de l'algorithme formellement. Par construction du graphe et des capacités, on a $\forall i \in \{1,\dots,n\},\ z_i(x)>\lambda$. Donc par la proposition 1, si l'algorithme de flot ne trouve pas de solution, alors c'est que le λ choisi est trop

4. En pratique cela n'arrive jamais si on utilise l'algorithme d'Edmonds-Karp
5. Nous avons appris après coup qu'il existait le théorème des valeurs des entières qui englobait cette proposition.

Input : $u_{i,j}$, la matrice des utilités
Output : $z^* = \mathbf{max}\,\mathbf{min}\, z_i(x)$
$\lambda \leftarrow 0$
$b^- \leftarrow \lambda$
$b^+ \leftarrow \mathbf{max}\{u_{i,j} | i, j \in \{1, ...n, \}\}$
$G \leftarrow G(V, E)$
while $\lambda \neq z^*$ **do**
  $C \leftarrow get\_capacites(G, \lambda)$
  $flot \leftarrow flot(G, C)$
  if *flot est valide* **then**
    $b^- \leftarrow \lambda$
  else
    $b^+ \leftarrow \lambda$
  end
  $\lambda \leftarrow \frac{b^- + b^+}{2}$
end
return λ

Algorithme 1 : Trouver le lambda maximum qui minimise l'utilité de l'agent le moins satisfait

grand. Sinon, alors on peut le faire grandir. Afin d'éviter d'itérer sur toutes les arêtes du graphe (dans le pire des cas), nous avons utilisé une recherche dichotomique. A noter que dans l'implémentation, nous avons implémenté le test d'égalité du **while** avec un compteur. Comme la recherche dichotomique met un temps logarithmique par rapport aux données, on a choisi le nombre d'itérations de telle sorte qu'il devrait être proportionnel à $2 \log_2 size$ [6]. Après quelques essais, nous avons mis comme coefficient de proportionnalité 2.

2.2 Question 5

On formule le programme linéaire suivant :

6. $size^2$ étant à peu prêt le nombre d'arêtes du graphe

Méthode	10	50	100
$\mathcal{P}_1$	.047	.184	.436
flot	.143	2.016	16.110

Temps moyens [7] entre les deux implémentations lorsque $M = 100$

Variables :

$$\underbrace{x_{i,j}}_{\text{Vaut 1 si l'agent } a_i \text{ reçoit le bien } b_j} \in \mathbb{B}\ 1 \leq i, j \leq n$$

$$\underbrace{z_{min}}_{\text{Satisfaction minimum d'un agent parmi tous les agents}} \in \mathbb{R}^+$$

Fonction objectif :

$$\textbf{max}\ z_{min} \qquad ()$$

Contraintes :

$$\sum_{i=1}^{n} x_{i,j} = 1,\ 1 \leq j \leq n \qquad \textit{1 agent par bien}$$

$$\sum_{j=1}^{n} x_{i,j} = 1,\ 1 \leq i \leq n \qquad \textit{1 bien par agent}$$

$$\sum_{1 \leq i,j \leq n} u_{i,j} x_{i,j} - z_{min} \geq 0,\ 1 \leq i \leq n \qquad z_{min} \textit{ doit être plus petit que la satisfaction de l'agent } i$$

Programme linéaire $\mathcal{P}_1$

2.3 Question 6

Vous trouverez les résultats dans le tableau 4

7. temps fournis par la commande time. Afin d'avoir une comparaison plus égalitaire.

Model	moyenne	val max	val min	difference max-min	ecrat type
P0	69.17	87.82	46.05	41.77	15.01
P1	66.33	82.29	53.00	29.28	10.61

comparatif du modèle max et maxmin pour n=5

Model	moyenne	val max	val min	difference max-min	ecrat type
P0	76.80	92.56	55.41	37.14	11.45
P1	73.58	89.60	60.72	28.88	8.89

comparatif du modèle max et maxmin pour n=10

On remarque dans ces résultats que l'algorithme utilisant les flots est plus lent. Pour autant, on ne peut pas dire que cette dernière méthode est plus lente. En effet, le modèle du programme linéaire de $\mathcal{P}_1$ est implémenté avec *gurobi* qui utilise le C/C++ qui est beaucoup plus rapide que le Python. Afin d'avoir des mesures un peu plus comparable, il faudrait implémenter l'algorithme de graphe en C/C++ efficacement !
Cependant, les résultats peuvent laisser penser que l'approche par programmation linéaire est plus rapide en pratique. Ceci n'est pas très étonnant. Gurobi est un outil payant et maintenu par quelques dizaines de personnes. L'algorithme du simplexe profite donc de nombreuses améliorations. A l'inverse, la librairie pygraph est en python, et son but premier n'est pas la performance pure, elle n'est donc certainement pas optimisée.

2.4 Question 7

D'après les tableaux 6, on peut observer que maximiser la satisfaction de l'agent le moins satisfait entraîne une répartition plus équitable des produits que lorsque l'on maximise la moyenne. A contrario, la moyenne des satisfactions par le programme $\mathcal{P}_1$ à tendance à être plus faible que la moyenne des satisfactions du programme $\mathcal{P}_0$.

2.5 Question 8

2.5.1 Modélisation de $\mathcal{P}_2$

Nous proposons la modélisation suivante pour $\mathcal{P}_2$:

Variables :

$$\underbrace{x_{i,j}}_{\text{Vaut 1 si l'agent } a_i \text{ reçoit le bien } b_j} \in \mathbb{B}\ 1 \leq i, j \leq n$$

$$\underbrace{z_{min}}_{\text{Satisfaction minimum d'un agent parmi tous les agents}} \in \mathbb{R}^+$$

Fonction objectif :

$$\textbf{max}\ z_{min} + \sum_{1 \leq i,j \leq n} \varepsilon u_{i,j} x_{i,j} \qquad ()$$

Contraintes :

$$\sum_{i=1}^{n} x_{i,j} = 1,\ 1 \leq j \leq n \qquad \textit{1 agent par bien}$$

$$\sum_{j=1}^{n} x_{i,j} = 1,\ 1 \leq i \leq n \qquad \textit{1 bien par agent}$$

$$\sum_{1 \leq i,j \leq n} u_{i,j} x_{i,j} - z_{min} \geq 0,\ 1 \leq i \leq n \qquad z_{min} \textit{ doit être plus petit que la satisfaction de l'agent } i$$

Programme linéaire $\mathcal{P}_2$

2.5.2 Comparaison de $\mathcal{P}_1$ avec $\mathcal{P}_2$

Afin de comparer les programmes $\mathcal{P}_1$ et $\mathcal{P}_2$, on se propose de modéliser le problème suivant :

> L'association ASCII est une association de récupération de matériel informatique. Elle vient de réparer deux ordinateurs. Le premier ordinateur est un Macbook pro de 2010 et l'autre ordinateur est un PC sorti en 2009 avec une configuration moyenne sous Debain. Deux personnes se sont prononcés pour récupérer ces ordinateurs :
>
> — Mr Michu souhaite un ordinateur pour faire de la bureautique. Il n'a pas de préférence personnel.

— Mr Suckerberg souhaite un ordinateur afin de l'aider dans le développement de sa nouvelle idée pour conquérir le monde. Bien sûr, il a une préférence pour l'ordinateur sous Debian.

La tâche de l'association ASCII est de distribuer les ordinateurs afin de satisfaire au maximum les besoin de Mr. Michu et de Mr. Suckerberg. L'association a donc défini une valeur qui correspond à l'utilité d'un des ordinateurs pour chacun des clients.

L'association a définit que l'utilité de Mr Michu pour les des deux ordinateurs est 2 puisque ce dernier n'a pas de préférence. Par contre, pour Mr Suckerberg, l'association à définit que l'utilité pour lui d'avoir le Macbook pro serait de 3 et de 10^6 s'il avait l'ordinateur sous Debian.

On peut évidemment modéliser ce problème comme un problème d'affectation. On obtient donc 4 variables à savoir :

— $x_{Michu,Mac}$;

— $x_{Michu,PC}$;

— $x_{Suckerberg,Mac}$;

— $x_{Suckerberg,Mac}$;

Leurs coefficients dans la fonction objectif sont donnés directement dans le problème. Par le programme $\mathcal{P}_1$, il y a deux solutions possibles :

1. $(x_{Michu,Mac}, x_{Michu,PC}, x_{Suckerberg,Mac}, x_{Suckerberg,PC})$=(1,0,0,1) ;
2. $(x_{Michu,Mac}, x_{Michu,PC}, x_{Suckerberg,Mac}, x_{Suckerberg,PC})$=(0,1,1,0) ;

En effet, dans la deux cas, la satisfaction associée à Mr Michu est de 2 et la satisfaction de Mr Suckerberg est plus grande.
A l'inverse, par le programme $\mathcal{P}_2$, il y a qu'une solution possible :

— $(x_{Michu,Mac}, x_{Michu,PC}, x_{Suckerberg,Mac}, x_{Suckerberg,PC})$=(1,0,0,1) ;

En effet, la satisfaction de Mr Suckerberg est plus grande dans la solution 1 que dans la solution 2. Cela entraîne donc que la fonction objectif est plus grande dans la solution 1 que dans la solution 2 car la satisfaction de Mr Michu n'a pas changé. On vient donc de trouver deux solutions différentes telles qu'avec le programme $\mathcal{P}_1$ la satisfaction de Mr Suckerberg soit strictement plus petite que dans la solution du programme $\mathcal{P}_2$ tandis que celle de Mr Michu n'a pas changé.
Les remarques qu'on peut en tirer c'est que maximiser seulement la satisfaction de l'agent le moins satisfait n'est pas suffisant. En effet, comme on le voit dans l'exemple précédent, il peut y avoir plusieurs solutions et alors on ne sait pas laquelle sera donnée par le simplexe. Afin de pallier ce problème, il faut dire au simplexe qu'il faut tout de même maximiser la satisfaction des agents sans pour autant outrepasser l'objectif premier de maximiser la satisfaction de l'agent le moins satisfait. Pour cela, on rajoute un facteur ε qui permet de différencier les différentes solutions possibles trouvées par le programme $\mathcal{P}_1$.

3 Approche égalitariste en regrets

3.1 Question 9

3.1.1 Modélisation de $\mathcal{P}_3$

Variables :

$$\underbrace{x_{i,j}}_{\text{Vaut 1 si l'agent } a_i \text{ reçoit le bien } b_j} \in \mathbb{B}\ 1 \leq i, j \leq n$$

$$\underbrace{r_{max}}_{\text{Satisfaction minimum d'un agent parmi tous les agents}} \in \mathbb{R}^+$$

Fonction objectif :

$$\textbf{max}\, r_{max} \qquad ()$$

Contraintes :

$$\sum_{i=1}^{n} x_{i,j} = 1,\ 1 \leq j \leq n \qquad \textit{1 agent par bien}$$

$$\sum_{j=1}^{n} x_{i,j} = 1,\ 1 \leq i \leq n \qquad \textit{1 bien par agent}$$

$$\sum_{1 \leq i,j \leq n} u_{i,j} x_{i,j} + r_{max} \geq z_i*,\ 1 \leq i \leq n \qquad r_{max} \textit{ doit être plus grand que le regret de l'agent } i$$

Programme linéaire $\mathcal{P}_3$

3.2 Question 10

L'algorithme utilisant les flots afin de procéder à l'approche égalitariste et l'approche par regret sont vraiment proches. En particulier, la construction du graphe sera équivalente. Avec deux changements notoires :

— les capacités (décrites ci-dessous) ;

— la recherche dichotomique se fait dans l'autre sens.

Les deux algorithmes étant similaires, nous n'explicitons pas le nouvel algorithme. Nous donnons juste le calcul des capacités :

$$\begin{aligned}
&\forall i,j \in \{1,\dots,n\} c(a_i;o_j) = 1 \text{ si } z_i(x)^* - u_{i,j} < \lambda \\
&\forall i,j \in \{1,\dots,n\} c(a_i;o_j) = 0 \text{ sinon} \\
&\forall i \in \{1,\dots,n\},\ c(s;a_i) = 1 \\
&\forall j \in \{1,\dots,n\},\ c(o_j;t) = 1
\end{aligned}$$

3.2.1 Description de l'algorithme :

3.3 Question 11

Vous trouverez les résultats dans le tableau 7

Méthode	10	50	100
$\mathcal{P}_1$	.058	.304	1.405
flot	.149	2.133	17.131

Comparaison des temps moyens (en seconde) entre les deux implémentations lorsque $M = 100$

Comme à la question 6, on s'aperçoit que l'implémentation par graphe semble plus lente que celle utilisant l'algorithme du simplexe. Ici aussi, les remarques sur la performance des langages s'appliquent.

4 Extension à l'affectation multiple

4.1 Question 12

L'affectation multiple est une généralisation du problème d'affectation simple. L'agent peut recevoir plusieurs objets et chaque objet est en différents exemplaires. Afin de considérer les changements, on va découper cette question en deux parties. Une première partie qui regardera seulement le côté programmation linéaire, et une seconde partie où l'on regardera l'approche par les graphes.

4.1.1 Approche par programmation linéaire

D'abord on va considérer les modifications à faire par rapport au programme $\mathcal{P}_0$. Ces modifications se répercuteront sur les deux autres approches. Ensuite on regardera les modifications à faire pour l'approche égalitariste et l'approche par regret.

Maximiser la moyenne des satisfactions : Dans le programme $\mathcal{P}_0$, nous avions deux types de contraintes. Le premier type de contraintes obligeait les agents à ne choisir qu'un objet. Cela correspond au cas où $\forall i \in \{1, .., n\},\ \alpha_i = 1$. Ce 1 se répercutait dans le membre de droite. Donc cela ce généralise très bien en remplaçant le membre de droite par α_i. On peut faire la même remarque pour les β_j qui correspondent au deuxième type de contraintes. On remarquera qu'on assouplit facilement la contrainte $m = n$. Cela se répercute juste sur le nombre de contraintes du problème. De plus, on remarquera que cette généralisation n'a aucune incidence sur la fonction objectif. La dernière chose à remarquer, est qu'en gardant des variables booléennes, on oblige à ce que chaque agent i choisissent un objet une seule fois. Or, si cet objet existe en plusieurs exemplaires, l'agent i voudra peut-être le prendre plusieurs fois. Il faut alors relâcher le domaine des variables. On obtient donc le programme linéaire $\mathcal{P'}_0$ suivant :

Variables :

$$\underbrace{x_{i,j}}_{\text{Vaut } n \text{ si l'agent } a_i \text{ reçoit } n \text{ fois le bien } b_j} \in \mathbb{N}\ 1 \leq i, j \leq n$$

Fonction objectif :

$$\textbf{max}\ \frac{1}{n} \sum_{1 \leq i,j \leq n,m} u_{i,j} x_{i,j}$$ (avec $u_{i,j}$ les coefficients d'utilité du bien i pour l'agent j)

Contraintes :

$$\sum_{i=1}^{n} x_{i,j} \leq \beta_j,\ 1 \leq j \leq m \qquad \beta_j \textit{ agent par bien}$$

$$\sum_{j=1}^{m} x_{i,j} \leq \alpha_i,\ \ 1 \leq i \leq n \qquad \alpha_i \textit{ bien par agent}$$

Programme linéaire $\mathcal{P}'_0$

Approche égalitariste : Dans l'approche égalitariste, les modifications faites précédemment ne changent pas. De plus, la notion de satisfaction ne change pas. Par conséquent le programme $\mathcal{P}_1$ se généralise simplement en appliquant les modifications précédentes.

Approche par regret : cette approche se généralise très bien à une petite différence prêt. $r_i(x)$ est défini en utilisant z_i^*. Seulement la définition de z_i^* s'applique seulement si l'agent choisit un seul objet. Il faut donc donner une nouvelle définition de z_i^*. Moralement z_i^* est un majorant de la satisfaction maximale que peut espérer l'agent i. Cependant si l'agent i peut recevoir α_i objets, alors $\alpha_i \times \textbf{max}\{u_{i,j} \mid j \in \{1, ..., m\}$ est bien un majorant mais pas une borne supérieure. On peut donc affiner la définition de z_i^* pour obtenir une borne supérieure.
Soit $\mathcal{U}_i$ le multiensemble des utilités de l'agent i :

$$\mathcal{U}_i = (U_i, m)$$

où

— $U_i = \{u_{i,j} \mid j \in \{1, \ldots, m\}\}$;

— $m(u_{i,j}) = \beta_j$;

Alors on peut définit z_i^* comme :

$$z_i^* = \sum_{i \in z_i^{\alpha_i}} i$$

où z_i^n est le multiensemble définit inductivement pour tout $n \in \mathbb{N}$:

$$\begin{aligned} z_i^0 &= \emptyset \\ z_i^{n+1} &= \mathbf{max}\{u_{i,j} \mid u_{i,j} \in \mathcal{U}_i \setminus z_i^n\} \cup z_i^n \end{aligned}$$

La définition de z_i^* donne directement un algorithme pour le calculer. Par conséquent, il suffit de reprendre la modélisation $\mathcal{P}_3$ en utilisant la nouvelle définition de z_i^*.

4.1.2 Approche par graphe

On s'intéresse ici aux algorithmes de graphe pour le problème d'affectation multiple lorsque l'on considère les deux approches précédentes à savoir :

— l'approche égalitariste ;

— l'approche par regret ;

Cependant nous n'avons pas trouvé de modélisation approprié pour ces deux approches. En effet, on peut avoir une bonne intuition des capacités sur les arrêtes sortantes de la source (α_i) et les arrêtes entrantes de la cible (β_j) mais sur les capacités du sous-graphe biparti, cela devient plus compliqué. En effet, l'approche par un algorithme glouton du même type qu'à la question 4 et 10 demanderait d'avoir une bonne heuristique sur la suppression des arrêtes dans le graphe. En particulier dans le problème de l'affectation simple, on était capable de garantir le fait que l'arrête choisit par l'agent le moins satisfait ne serait pas choisie s'il existait une solution meilleure. Dans le cadre de l'affectation multiple, un agent va pouvoir choisir plusieurs arrêtes. Mais cette fois on ne peut plus garantir que la plus petite arrête choisi par l'agent le moins satisfait ne pourra pas être choisi dans une autre solution. Car une nouvelle combinaison comprenant cette arrête pourrait améliorer nettement la satisfaction de l'agent.
Une idée *théorique* possible serait d'envisager plusieurs cas. Par exemple, de la même manière qu'on a calculer le regret maximal pour un agent, on pourrait calculer la satisfaction minimal d'un agent. Parmi les α_i arrêtes choisis par l'agent, on sait qu'il y en aura au moins une qui ne sera pas utilisé. Il faut donc envisager tous les cas où on supprime une arête $k \in \alpha_i$. Cela nous amène à parcourir un

arbre. En pratique cette méthode ne fonctionnera pas car l'arbre est exponentiel par rapport à la taille du problème. Afin d'avoir un algorithme efficace, cela revient donc à trouver une heuristique afin de choisir quels sont les noeuds intéressant à regarder.

5 Allocation équitable avec l'opérateur OWA

5.1 Question 13

Nous avons légèrement modifier $\mathcal{P}_{\mathcal{L}_k}$ afin de transformer la contrainte d'égalité en deux contraintes. On obtient alors le programme linéaire $\mathcal{D}_{\mathcal{L}_k}$ suivant :

Variables :

$$\underbrace{d^+_{seuil}} \in \mathbb{R}^+$$

$$\underbrace{d^-_{seuil}} \in \mathbb{R}^+$$

$$\underbrace{d_i} \in \mathbb{R}^+ \ 1 \leq i \leq n$$

Fonction objectif :

$$\textbf{max} \ k(d^+_{seuil} - d^-_{seuil}) - \sum_{1 \leq i \leq n} d_i \qquad ()$$

Contraintes :

$$d^+_{seuil} - d^-_{seuil} - d_i \leq z_i(x),\ 1 \leq i \leq n$$

Programme linéaire $\mathcal{D}_{\mathcal{L}_k}$

Ce qui est intéressant dans le dual c'est de remarquer que les $z_i(x)$ passent dans les contraintes. Là où dans le primal, $z_i(x)$ devaient obligatoirement être des constantes car c'était le coefficient associé à la variable y_{i_k}, alors que maintenant, on peut envisager $z_i(x)$ comme étant une variable.

On peut donc en déduire le programme linéaire suivant $\mathcal{L}_k(x)$ afin de calculer pour k fixé $\mathcal{L}_k(x)$:

Variables :

$$\underbrace{d^+_{seuil}}_{} \in \mathbb{R}^+$$

$$\underbrace{d^-_{seuil}}_{} \in \mathbb{R}^+$$

$$\underbrace{d_i}_{} \in \mathbb{R}^+ \; 1 \leq i \leq n$$

$$\underbrace{x_{i,j}}_{\text{Vaut 1 si l'agent } a_i \text{ reçoit le bien } b_j} \in \mathbb{B} \; 1 \leq i, j \leq n$$

Fonction objectif :

$$\textbf{max} \;\; k(d^+_{seuil} - d^-_{seuil}) - \sum_{1 \leq i \leq n} d_i \qquad (f_{\mathcal{L}_k})$$

Contraintes :

$$\sum_{1 \leq j \leq n} u_{i,j} x_{i,j} + d_i \geq d^+_{seuil} - d^-_{seuil}, \; 1 \leq i \leq n$$

Programme linéaire $\mathcal{L}_k$

L'intérêt de résoudre ce problème dans le contexte du partage équitable est d'assurer une répartition des satisfactions qui soient *équitables*. C'est à dire qu'il n'y ait pas seulement $n - k$ agents qui soit extrêmement contents même si cela peut-être profitable pour l'intérêt de tous, mais que les k agents soient aussi satisfaits. On pourrait aussi voir ça d'une certaine façon pour contrer le *principe de Pareto*.

5.2 Question 14

Afin de maximiser l'opérateur *OWA* nous considérons les $L_k(x)$ comme des *méta-variables* que seul le programme $\mathcal{L}_k$ peut comprendre. Une occurence d'une variable

$L_k(x)$ se traduit donc en quelque sort par une occurence du programme $\mathcal{L}_k$. Cela nous amène donc à considérer le programme linéaire $\mathcal{OWA}$ suivant :

Variables :

$$\underbrace{d_{k^+_{seuil}}}_{} \in \mathbb{R}^+ \; 1 \leq k \leq n$$

$$\underbrace{d_{k^-_{seuil}}}_{} \in \mathbb{R}^+ \; 1 \leq k \leq n$$

$$\underbrace{d_{k_i}}_{} \in \mathbb{R}^+ \; 1 \leq k, i \leq n$$

$$\underbrace{x_{i,j}}_{} \in \mathbb{B} \; 1 \leq i, j \leq n$$

Vaut 1 si l'agent a_i reçoit le bien b_j

Fonction objectif :

$$\textbf{max} \sum_{k=1}^{n-1} (w_k - w_{k+1}) f_{\mathcal{L}_k} + w_n f_{\mathcal{L}_n} \qquad (f_{\mathcal{L}_k} \text{ définit dans } \mathcal{L}_k)$$

Contraintes :

$$\sum_{i=1}^{n} x_{i,j} = 1, \quad 1 \leq j \leq n \qquad \textit{1 agent par bien}$$

$$\sum_{j=1}^{n} x_{i,j} = 1, \quad 1 \leq i \leq n \qquad \textit{1 bien par agent}$$

$$z_i(x) + d_{k_i} - (d_{k^+_{seuil}} - d_{k^-_{seuil}}) \geq 0, \; 1 \leq k, i \leq n$$

Programme linéaire $\mathcal{L}_k$

5.3 Question 15

Remarque : dans l'implémentation du modèle, la suite (w_i) est fixe et est définit comme $\forall i \in \{1, ..., n\}, \; w_i = n - i + 1$.

size	moyenne des satisfactions	temps moyen (en seconde)
10	77.10	.13
50	88.96	10.28
100	91.77	1234.92

Temps et valeur moyenne pour $M = 100$ avec l'opérateur OWA

Model	moyenne	val max	val min	difference max-min	ecrat type
OWA	68.70	84.77	50.99	33.78	12.12
P1	67.39	81.43	54.79	26.63	9.85

comparatif du modèle OWA et maxmin pour n=5

Model	moyenne	val max	val min	difference max-min	ecrat type
OWA	76.65	91.58	58.61	32.96	10.06
P1	73.22	88.75	61.06	27.69	8.76

comparatif du modèle OWA et maxmin pour n=10

Vous trouverez dans le tableau 8 les temps moyens pour $n = 10, 50, 100$. Nous faisons tout de même une remarque en remarquant que la disparité des temps pour $n = 100$ est assez impressionante. Et ce n'est pas tout. Au minimum le programme prenait 200 secondes, mais pour deux runs, il a prit près de 4000 secondes. Cela peut peut-être venir de l'ordinateur sur lequel nous avons fait tourné les tests.

5.4 Question 16

Comment on peut le constater sur les tableaux 10, la modélisation avec l'opérateur OWA tend à améliorer la moyenne au dépit de l'écart type. On a donc comme pour la première approche une disparité un peu plus grande des satisfactions. Les résultats obtenus étant similaires à ceux de la question 7, ce qui pourrait être intéressant, serait de faire la comparaison avec le programme $\mathcal{P}_0$.

5.5 Comparaison du modèle $\mathcal{OWA}$ avec $\mathcal{P}_0$

Cette fois on remarque que pour l'opérateur $\mathcal{OWA}$, la moyenne est un tout petit peu plus faible afin de favoriser l'écart-type par rapport $\mathcal{P}_0$. C'est pas étonnant que la moyenne soit plus basse pour le modèle $\mathcal{OWA}$, mais ce qui est intéressant de se rendre compte, c'est que cet opérateur favorise à la fois la moyenne et l'écart-type. C'est donc un bon compromis entre les modèles $\mathcal{P}_0$ et $\mathcal{P}_1$. Cependant, le modèle est beaucoup plus lourd à éxécuter.

Model	moyenne	val max	val min	difference max-min	ecrat type
OWA	69.28	85.96	51.37	34.58	12.58
P0	69.57	86.31	49.71	36.59	13.52

comparatif du modèle OWA et maxmin pour n=5

Model	moyenne	val max	val min	difference max-min	ecrat type
OWA	76.25	90.97	58.53	32.44	9.71
P0	76.05	92.80	56.85	35.94	11.17

comparatif du modèle OWA et maxmin pour n=10