Projet ZZ1 - Affectation de projets aux ZZ1

Antony Vigouret et Hugo Bayoud

5 juin 2019

Table des matières

1 Introduction

Chaque année, il est demandé aux ZZ1 de participer en binôme à un projet en informatique. Pour ceux qui ont déjà une idée, ils peuvent proposer leur propre projet. **Pour ceux qui n'ont pas d'idée précise, une liste de projets** établis par les professeurs ou intervenants de l'ISIMA **est mise à disposition des binômes**. Il leur est demandé de **choisir parmi cette liste de sujets, 3 d'entre eux qui les intéresseraient de traiter et parmi ces 3, de les classer par ordre d'affinité.**
Une affectation est ensuite réalisée pour proposer à chacun de ces binômes un projet.

Il convient donc de formuler la problématique du sujet :

> Comment affecter un projet aux n binômes en évitant au maximum les déçus ; i.e pour chaque binôme, éviter de leur affecter un projet qu'ils n'ont pas choisi.

Nous faisons donc face à un **problème d'affectation** et décidons de représenter ce problème grâce un **programme linéaire**. Afin de résoudre ce programme linéaire, nous allons utiliser une bibliothèque d'optimisation appelée Gurobi que nous implémenterons dans un code écrit en C.

Afin de traiter au mieux le problème que nous voyons apparaître, nous verrons dans un premier temps quelles sont les **données** dont nous avons besoin **en entrée** de notre programme informatique pour renseigner correctement les composantes du programme linéaire ; et également quelles sont les **données de sortie** que nous souhaitons pour répondre correctement à la problématique. Nous expliciterons certaines notations **en généralisant par la suite le problème** et l'agrémenterons d'un **exemple simple** qui explicitera clairement la modélisation d'un programme linéaire. Nous pourrons alors présenter le **code source que nous avons écrit et les tests que nous avons fait sur ce dernier pour vérifier son bon fonctionnement.** Ceci permettra enfin de **conclure** quant à l'utilité, la performance de notre travail.

2 Fichiers d'entrée et sortie

2.1 Introduction des notations

Posons le problème de façon mathématique en introduisant les notations utiles.

On considère qu'il y a :

- n **binômes.**
- m **projets disponibles au total** (tel que $n \leq m$). On comprendra davantage leur utilité dans un exemple concret.
- **Chaque binôme peut faire k choix différents** et les classer.

Enfin, nous avons des poids qui vont pondérer les variables de décisions adéquates pour faire en sorte de pénaliser le mieux possible le fait d'affecter à un binôme, un projet qui n'est pas son 1^{er} voeu : plus le projet apparaît "loin" dans ses voeux, plus le poids sera grand. Il sera d'autant plus grand s'il ne fait même pas parti des choix initiaux.

- k entiers p_1, p_2, ..., p_k qui correspondent chacun à l'**importance donnée au voeu n° 1, n° 2, ..., n° k** (tels que $p_1 < p_2 < ... < p_k$).

Expliquons ce dont nous avons besoin comme données à l'entrée du programme pour permettre la résolution du programme linéaire et quel en sont les données de sortie utiles pour savoir à quel binôme est affecté quel projet.

2.2 Fichier d'entrée

En entrée, nous nous attendons à recevoir un tableau qui contient les informations suivantes :

Paramètres n, m et k
poids $p_1, p_2, ..., p_k$
N° du binôme et ses k choix de projets
N° du binôme et ses k choix de projets
...
...

Tableau que nous transformons en fichier *.txt* d'entrée pour notre programme de résolution. Fichier *.txt* que le programme transformera en matrice, bien plus exploitable et utile.

exemple :

```
3   5   3
10  20  30

1   1   5   3
2   4   2   5
3   4   1   3
```

Où,

- La **première ligne** correspond toujours à n, m, k.
- Le **deuxième ligne** correspond aux poids. (il y a k colonnes).
- Les **autres lignes correspondent chacune à un binôme**. La première colonne est le n° du binôme, la deuxième colonne, le 1^{er} choix du binôme, la troisième colonne, le 2^{ieme} choix, etc...

On attend également en donnée d'entrée, un fichier (au format *.txt*) qui est une **liste de l'ensemble des projets disponibles pour l'année**, qu'ils aient été choisis ou non par un binôme. Nous reviendrons sur son utilité un peu plus tard.

2.3 Sortie de l'algorithme

En sortie, nous aurons une matrice de solution. Néanmoins, nous ne présentons pas cette matrice comme résultat finale mais un tableau bien plus compacte et lisible qui **évite les informations inutiles** (ensemble de zéros).
Nous aurons un tableau de la sorte dans un fichier.*txt* :

N° du binôme	N° du projet affecté	Importance du projet pour le binôme
...	...	...
...	...	...
...	...	...

exemple :

```
1   5   2
2   4   1
3   3   3
```

En reprenant l'exemple du fichier d'entrée, on peut imaginer une affectation où le binôme n° 1 a obtenu son 2^{ieme} choix : le projet n° 5.

3 Généralisation du problème

Abordons ici la généralisation du problème. Utilisons les notations que nous avons introduit et réfléchissons sur les contraintes de notre problème linéaire. Nous rappelons que :

- n = nombre de binômes.
- m = nombre de projets tel que $n \leq m$.
- k = nombre de choix que peut faire un binôme.

3.1 Données d'entrée

Nous pouvons généraliser les fichiers d'entrée attendus de la façon suivante :

$$
\begin{array}{ccccc}
n & m & k & & \\
p_1 & p_2 & \cdots & p_k & \\
 & & & & \\
id_1 & v_{1,1} & v_{1,2} & \cdots & v_{1,k} \\
id_2 & v_{2,1} & v_{2,2} & \cdots & v_{2,k} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
 & & & \vdots & \\
id_n & v_{n,1} & v_{n,2} & \vdots & v_{n,k}
\end{array}
$$

Où,

$\forall i \in [\![1, k]\!]$, p_i est le poids attribué au i^{ieme} choix de projet.

$\forall i \in [\![1, n]\!]$, id_i est le n° du i^{ieme} binôme.

$\forall (i, j) \in [\![1, n]\!] \times [\![1, k]\!]$, $v_{i,j}$ est le n° de projet que le i^{ieme} binôme classé en j^{ieme} choix.

Comme nous l'avons dit précédemment, nous transformons ce fichier *.txt* en une matrice d'entrée pour la résolution. Nous avons donc besoin de matrices.

3.2 Matrice C

La matrice d'entrée (notée C) a n lignes et m colonnes et est construite à partir du fichier d'entrée vu précédemment.
Chaque coefficient $c_{i,j}$ de C correspond à une valeur d'un poids associée au choix que le projet n° j représente pour le binôme n° i. $c_{i,j}$ vaut p_t si le projet n° j a été choisis en t^{ieme} voeu par le binôme n° i.

Exemple : $c_{i,j}$ vaut p_1 si le binôme i a choisi le projet n° j en 1^{ier} voeu (*i.e.* que ce projet j se trouve dans la première colonne des choix dans le fichier d'entrée) et $c_{i,j}$ vaut 100 * p_k si le projet n° j n'a pas été choisi par le binôme n° i.

Note : Lorsqu'un projet n'est pas choisi, nous lui affectons un poids p_{k+1} qui vaut 100 fois le poids p_k. Nous allons le voir, nous cherchons à minimiser une somme, nous mettons donc une valeur élevée à un poids associé à un projet que le binôme ne souhaite pas.

Nous pouvons donc écrire la matrice C de la façon suivante :

$$C \in M_{n,m}(\mathbb{R}),\ \forall (i,j) \in [\![1,n]\!] \times [\![1,m]\!] \ /\ c_{i,j} \in \{p_1, p_2, ..., p_k, 100 * p_k\}$$

$$C = \begin{bmatrix} c_{1,1} & \dots & \dots & \dots & c_{1,m} \\ \vdots & c_{2,2} & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ c_{n,1} & \dots & \dots & \dots & c_{n,m} \end{bmatrix}$$

3.3 Matrice X

Le format de sortie des variables décisionnelles calculées par Gurobi peut être une matrice (notée X) à n lignes et m colonnes.
Chaque coefficient $x_{i,j}$ vaut 1 si le binôme i a obtenu le projet n° j, 0 sinon.

- Il ne faut pas qu'un binôme ait 2 projets ou plus ni qu'un projet soit affecté à 2 binômes ou plus. Dans la matrice X, **il ne faut donc pas qu'il y ait plus d'un coefficient différent de 0 par ligne et par colonne pour qu'elle soit valide.**
- Néanmoins, il faut que chaque binôme ait un projet. **Il faut donc qu'il y est au moins un coefficient à 1 par ligne.**

$$X \in M_{n,m}(\mathbb{R}),\ \forall (i,j) \in [\![1,n]\!] \times [\![1,m]\!] \ /\ x_{i,j} = \left\{ \begin{array}{l} 0 \\ 1 \end{array} \right.$$

$$X = \begin{bmatrix} x_{1,1} & \dots & \dots & \dots & x_{1,m} \\ \vdots & x_{2,2} & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ x_{n,1} & \dots & \dots & \dots & x_{n,m} \end{bmatrix}$$

Le but est de **minimiser la somme des coefficients de la matrice X pondérés par le coefficient équivalent de la matrice C**.

3.4 Programme linéaire

Ainsi, on peut généraliser le problème sous forme d'un programme linéaire où on cherche à minimiser

$$\sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij}$$

sous

$$\forall i \in [\![1,n]\!],\ \sum_{j=1}^{m} x_{ij} = 1 \qquad (1)$$

$$\forall j \in [\![1,m]\!],\ \sum_{i=1}^{n} x_{ij} \leq 1 \qquad (2)$$

avec

$$\forall (i,j) \in [\![1,n]\!] \times [\![1,m]\!] \ /\ x_{i,j} \in \{0,1\}$$

(1) : chaque binôme $i \in [\![1,n]\!]$ se voit affecter exactement 1 projet.

(2) : chaque projet $j \in [\![1,m]\!]$ est affecté au plus 1 fois à un projet.

4 Un exemple d'un cas simple $n = 3$, $m = 5$

Pour cet exemple, reprenons le fichier d'entrée que nous avions déjà introduit.

```
3   5   3
10  20  30

1   1   5   3
2   4   2   5
3   4   1   3
```

Modélisons-le comme pour un problème linéaire :

- $n = 3$, $m = 5$, $k = 3$.
- $p_1 = 10$, $p_2 = 20$, $p_3 = 30$

Soit C notre matrice d'entrée,

$$C = \begin{pmatrix} 10 & 3000 & 30 & 3000 & 20 \\ 3000 & 20 & 3000 & 10 & 30 \\ 20 & 3000 & 30 & 10 & 3000 \end{pmatrix}$$

Expliquons la première ligne :
comme nous l'avons dit lors de la généralisation, le fichier d'entrée donne le numéro des projets par ordre d'importance pour chaque binôme.
Ainsi,

- le binôme n° 1 a choisi le projet n° 1 comme premier choix, $c_{1,1} = p_1 = 10$.
- Le projet n° 2 n'étant pas été choisi par ce binôme, $c_{1,2} = p_k * 100 = 3000$.
- Le projet n° 3 est le troisième choix du binôme, $c_{1,3} = p_3 = 30$.
- Le projet n° 5 étant le deuxième choix du binôme, $c_{1,5}$ vaut $p_2 = 20$.

Nous avons X notre matrice de sortie dont il faut déterminer les coefficients (inconnues de notre PL = variables de décisions).

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} & x_{1,5} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} & x_{2,5} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} & x_{3,5} \end{pmatrix}$$

Où, $\forall (i,j) \in [\![1,n]\!] \times [\![1,m]\!]$, $x_{i,j} \in [\![0,1]\!]$

Avec les contraintes :

$$\begin{cases} x_{1,1} + x_{1,2} + x_{1,3} + x_{1,4} + x_{1,5} = 1 \\ x_{2,1} + x_{2,2} + x_{2,3} + x_{2,4} + x_{2,5} = 1 \\ x_{3,1} + x_{3,2} + x_{3,3} + x_{3,4} + x_{3,5} = 1 \end{cases}$$

et

$$\begin{cases} x_{1,1} + x_{2,1} + x_{3,1} \leqslant 1 \\ x_{1,2} + x_{2,2} + x_{3,2} \leqslant 1 \\ x_{1,3} + x_{2,3} + x_{3,3} \leqslant 1 \\ x_{1,4} + x_{2,4} + x_{3,4} \leqslant 1 \\ x_{1,5} + x_{2,5} + x_{3,5} \leqslant 1 \end{cases}$$

On peut enfin écrire la fonction à minimiser qui reprend les coefficients de la matrice C en les multipliant aux variables de décisions de la matrice X :

$\lambda = min(10.x_{1,1} + 3000.x_{1,2}30.x_{1,3} + 3000.x_{1,4} + 20.x_{1,5} + 3000.x_{2,1} + 20.x_{2,2} + 3000.x_{2,3} + 10.x_{2,4} + 30.x_{2,5} + 20.x_{3,1} + 3000.x_{3,2} + 30.x_{3,3} + 10.x_{3,4} + 3000.x_{3,5})$

5 Présentation du code source

5.1 Présentation

Avant d'étudier les résultats sur des instances de test, attardons nous sur l'agencement du code source. **Écrit en C, nous avons codé le programme à l'aide d'une bibliothèque externe : la bibliothèque** *Gurobi*.

Gurobi Optimization a pour but de résoudre des problèmes mathématiques. Nous l'utilisons ici comme outil pour résoudre notre programme linéaire que l'affectation des projets aux ZZ1 fait apparaître.

5.2 Installer Gurobi

Gurobi peut être installé sur Linux ou Windows mais nous n'expliquerons que l'installation sous Linux.

Voyons ça étapes par étapes :

1. S'inscrire sur le site avec une adresse mail universitaire tout en étant connecté au réseau universitaire (à l'ISIMA, EduSpot ou EduRoam fonctionnent) : http ://www.gurobi.com/registration/general-reg
2. Choisir son mot de passe et donner les informations utiles.
3. Se connecter sur le site ici : http ://www.gurobi.com/
4. Télécharger gurobi ici : http ://www.gurobi.com/downloads/gurobi-optimizer

   Le reste est spécifique à LINUX :

5. Extraire le tar.gz à l'emplacement de votre choix.
6. Récupérer une licence universitaire (clé d'activation) ici : http ://www.gurobi.com/downloads /licenses/license-center.
   L'exécutable *grbgetkey* se trouve dans <path>/gurobi8.1.0_linux64/gurobi810/linux64/bin
7. Tester l'installation et l'activation de la clé, aller dans : <path>/gurobi8.1.0_linux64/gurobi810 /linux64/examples/build
8. En ligne de commande, taper : make.

   Si vous avez des erreurs du type :
   gurobi8.1.0_linux64/gurobi810/linux64/examples/build/../ c++/callback_c++.cpp :158 : référence indéfinie vers « GRBModel : :GRBModel(GRBEnv const, std : :__cxx11 : :basic_string<char, std : :char_traits<char>, std : :allocator<char> > const) »

   (a) Aller dans : <path>/gurobi8.1.0_linux64/gurobi810/linux64/src/build

(b) En ligne de commande, taper : make

(c) Copier libgurobi_c++.a dans : <path>/gurobi8.1.0_linux64/gurobi810/linux64/lib

(d) Reprenez depuis l'étape de test d'installation (n° 7).

Essayer de lancer, pour tester, l'exécutable d'exemple *mip1_c* en tapant :
./mip1_c

En cas d'une erreur du type :
./mip1_c : error while loading shared libraries : libgurobi81.so : cannot open shared object file : No such file or directory

(a) Ouvrir le fichier bashrc, taper : ∼/.bashrc

(b) Ajouter à la fin de celui-ci :
export LD_LIBRARY_PATH=<path>/gurobi8.1.0_linux64/gurobi810/linux64/lib :$LD_LIBRARY_PATH

(c) Mettre à jour le fichier bashrc, taper en ligne de commande : source ∼/.bashrc

(d) Lancer à nouveau l'exécutable, taper : ./mip1_c

Note : L'ensemble du code est disponible sur : https ://github.com/hugobayoud/ProjetZZ1.git

5.3 Structure du code

Le programme qui permet l'affectation des choix de projet des ZZ1 demande en entrée un fichier texte (qui contient l'ensemble des choix des binômes) et un deuxième fichier texte (qui est une liste des tous les projets disponibles).

Le code est disposé dans le répertoire *resolutionSansGeneration* de la manière suivante :

```
|
|   main.c
|   Makefile
|
├──edition
|     └ // FICHIERS EDITION (.o)
|
├──include
|     └ // FICHIERS INCLUDES (.h)
|
└──src
|     └ // FICHIERS SOURCES (.c)
|
|   Instance1.txt
|   Instance2.txt
|   listeProjetInstance1.txt
|   listeProjetInstance2.txt
```

Nous laissons disponible dans ce répertoire deux instances de test qui sont la liste des binômes et projets d'années précédentes.
Note : Nous présentons ces instances de test plus tard dans le rapport.

La structure suivante permet d'enregistrer l'ensemble des informations du fichier d'entrée pour les traiter via gurobi.

```c
/*Structure qui regroupe l'ensemble
 des données d'entrée utiles*/

typedef struct donneesEntree
{
    int   nbBinome;
    int   nbProjet;
    int   nbChoix;
    int  *poids;
    int **choix;
    int **res;
    int **correspondance;
} Datas_t;
```

En pratique, les numéros de projets ne sont pas nécessairement des entiers de 1 à n. Or **gurobi a besoin d'entiers allant de 1 à n pour la résolution.** Nous créons donc un **tableau de correspondance** pour récupérer le vrai numéro de projet et donner un fichier de sortie facilement compréhensible.

5.4 Utilisation et Commandes

5.4.1 Programme principale

Les commandes sont les suivantes :

- Lancer la compilation : *"make"*.
- Lancer le nettoyage des fichiers *.o* dans *edition* : *"make clean"*.
- Lancer les tests du programme :
  "./prog <fichierListeChoixDesBinomes> <fichierListeProjetsDisponibles>"

Note : Des tests sont faits dans la fonction "main" pour être sûr qu'il y ait deux arguments lors de l'exécution de "./prog", .

Note 2 : le fichier "main.c" ne contient que les appels de génération de fichier d'entrée test et de lancement de gurobi pour la résolution.

5.4.2 Programme annexe

Lors de notre projet, nous avons eu besoin de générer des instances (**totalement aléatoires et uniquement utiles pour tester le programme de résolution**) car nous n'avions pas eu accès aux 2 instances directement.

Nous laissons donc ce programme dans le répertoire *resolutionAvecGeneration* qui peut être utile pour **comprendre le fonctionnement** de celui-ci et son **débogage** sans s'occuper de créer un fichier d'entrée.

Le programme se lance facilement :

1. après un *make* en ligne de commande,
2. lancer le programme avec *./prog*, il sera demandé les valeurs de n, m et k désirées et lancera la résolution.
3. Un fichier texte est créé, contenant la solution du programme.

5.5 Fonctions, prototypes

Dans cette partie nous détaillons les fonctions disponibles dans le code de *resolutionSansGeneration* bien qu'une grande partie se retrouve au final également dans *resolutionAvecGeneration*.

5.5.1 Fonctions liées aux Matrices

Comme nous l'avons plusieurs fois dit, les matrices sont essentielles pour permettre la résolution du programme. Ainsi, **nous avons créé une structure de donnée pour les matrices**. En voici les fonctions associées dont nous avons besoin que l'on retrouve dans "*matrice.c*".

Note : Les fonctions utilisent des éléments de la structure "Datas_t" qui a été défini plus haut.

La fonction "*FichierEnStructureDatas*" :
Affecter les données contenues dans le fichier d'entrée dans une structure.

Signature : `Datas_t* FichierEnStructureDatas(FILE *fic1, FILE *fic2);`

- Entrée :
  - fic1 (FILE *) : Pointeur du fichier d'entrée qui contient l'ensemble des données d'entrée.
  - fic2 (FILE*) : Pointeur du fichier d'entrée qui contient la liste des projets disponibles.
- Sortie
  - Datas (*Datas_t* *) : Pointeur vers *Datas_t*.

La fonction "*StructureEnMatrice*" :
Création de la matrice d'entrée (pour gurobi) à partir de la structure "*Datas_t*".

Signature : `int** StructureEnMatrice(Datas_t *Datas);`

- Entrée :
  - Datas (*Datas_t* *) : Pointeur de la structure de données.
- Sortie :
  - matrice (int **) : Double pointeur pour regrouper les données de la structure sous forme de matrice.

La fonction "$NumProjet\_BinomeChoix$" :
Donner le N° de projet du i^{me} choix du j^{ime} binôme.

Signature : `int NumProjet_BinomeChoix(Datas_t *Datas, int numBinome, int numChoix);`

— Entrée :
- Datas ($Datas\_t$ *) : Pointeur de la structure de données.
- numBinome (int) : N° du binôme auquel on veut connaître un choix de projet.
- numChoix (int) : N° choix du projet du binôme auquel on veut connaître le N° de projet associé.

— Sortie :
- res (int) : N° projet de "$numChoix$" choix du "$numBinome$" binôme.

La fonction "$NumChoix\_BinomeProjet$" :
Donner le N° d'un choix d'un projet pour un binôme donné.

Signature : `int NumChoix_BinomeProjet(Datas_t *Datas, int numBinome, int numProjet);`

— Entrée :
- Datas (Datas_t *) : Pointeur de la structure de données.
- numBinome (int) : N° du binôme auquel on veut connaître un N° de choix à partir d'un N° de projet.
- numProjet (int) : N° projet auquel on veut connaître le N° de choix associé pour un binôme donné.

— Sortie
- res (int) : N° projet de "$numChoix$"^{ieme} choix du "$numBinome$" binôme.

La fonction "*CreationFichierDeSortie*" :
Création du fichier de sortie sous forme de tableau
||N° de binôme | N° de projet obtenu | N° importance du choix||

Signature : `void CreationFichierDeSortie(Datas_t *Datas);`

— Entrée :
  - Datas (Datas_t *) : Pointeur de la structure de données.

— Sortie
  - void.

5.5.2 Générer un fichier d'entrée

La génération d'un fichier d'entrée pour la résolution est implémentée dans "*generer_entree.c*" et n'est disponible que dans le répertoire *resolutionAvecGeneration*. **La génération de fichier d'entrée permet uniquement de tester notre résolution via Gurobi.**

Note : Les fonctions utilisent des éléments de la structure "Datas_t" qui a été défini plus haut.

La fonction "*GenerateurDonneesEntree*" :
Génère automatiquement un fichier d'entrée exploitable par la suite pour gurobi en demandant n, m, i.

Signature : `void GenerateurDonneesEntree(int nbBinome, int nbProjet, int nbChoix);`

— Entrée :
  - nbBinome (int) : Nombre de binômes souhaités dans le fichier.
  - nbProjet (int) : Nombre de projets souhaités dans le fichier.
  - nbChoix (int) : Nombre de choix laissés à chaque binôme.

— Sortie
  - void.

5.5.3 Lancement de Gurobi

Une fonction, présente dans les deux répertoires, est disponible pour lancer la résolution avec gurobi. Cette fonction est implémentée dans : "*GUROBI.c*".

Note : Dans resolutionAvecGeneration, une fonction demande d'abord à l'utilisateur les paramètres n,m et i puis lance la résolution.

Note 2 : Nous mettons à disposition en annexe de ce rapport un petit tutoriel sur comment créer un environnement gurobi pour permettre une bonne utilisation de la bibliothèque.

La fonction "*LancementGurobi*" :
Permet de lancer la résolution avec gurobi. Que le fichier d'entrée doive être généré ou non, "*LancementGurobi*" est appelée.

Signature : `void LancementGurobi(char *Fichier1, char *Fichier2);`

- Entrée :
  - Fichier1 (char*) : fichier d'entrée contenant les choix des binômes.
  - Fichier2 (char*) : fichier contenant la liste de tout les projets.
- Sortie
  - void.

La fonction "*LancementGurobi2*" :
Permet de faire appel à la fonction "*GenerateurDoneesEntree*" puis de lancer gurobi via "*LancementGurobi*".

Signature : `void LancementGurobi2();`

- Entrée :
  - void.
- Sortie
  - void.

6 Instances de test

Expliquons désormais 2 instances de test récupérées auprès de Michelle Chabrol. Le but étant, avec Gurobi, d'obtenir un meilleur résultat (au minimum identique) que celui de Mme Chabrol.

A travers ces instances, **nous allons voir qu'il est possible que certains binômes ne peuvent avoir un des 3 projets qu'ils souhaitaient** car le programme linéaire ne peut résoudre le problème en affectant à tous les binômes un projet souhaité.
En effet, on peut imaginer **le cas où le nombre de binômes est supérieur au nombre de projets distincts choisis**. Il y aura donc obligatoirement des binômes qui se retrouveront avec un projet qui ne fait pas partie de leurs 3 choix.
Appelons un tel projet, un 4^{ieme} choix.

Note : Nous appelons ceci un 4^{ieme} choix parce que nous prenons $k = 3$. Le nom générique d'un tel projet serait plutôt un $(k+1)^{ieme}$ choix.

6.1 Instance de test N° 1

6.1.1 Présentation des résultats

L'instance N° 1 se présente de la manière suivante :
Il y a 47 projets pour 42 binômes ($n = 42$, $m = 47$). Néanmoins, uniquement 38 projets ont été choisis parmi les 3 choix des 42 binômes. **Ainsi, on constate qu'il y a moins de projets choisis que de binômes.**

Le premier tableau donne les résultats que trouve Mme Chabrol quand elle fait elle même l'affectation :

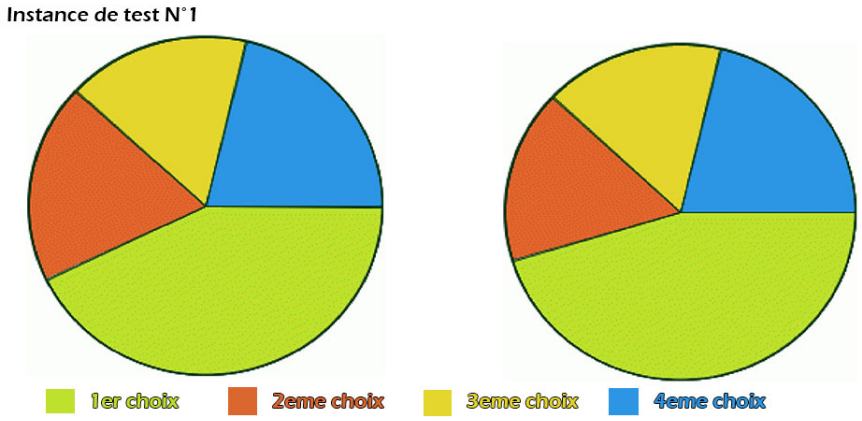
i^{ieme} choix affecté	Nb. de binômes affectés à leur i^{ieme} choix
1	18 (42,9%)
2	8 (19,0%)
3	7 (16,7%)
4	9 (21,4%)
TOTAL	42 (100%)

Note : Les résultats présentés ont été faits avec les poids suivants :
$p_1 = 1$, $p_2 = 2$, $p_3 = 3$.

Après le lancement du programme nous obtenons un fichier *txt* de sortie dont l'essentiel des informations à retenir est dans le tableau suivant :

i^{ieme} choix affecté	Nb. de binômes affectés à leur i^{ieme} choix
1	19 (45,2%)
2	7 (16,7%)
3	7 (16,7%)
4	9 (21,4%)
TOTAL	42 (100%)

Pour mieux se rendre compte de la différence entre les 2 résultats (celui de Mme Chabrol à gauche et le notre à droite), mettons-les côte à côte :



6.1.2 Conclusion de cette instance

Pour cette première instance, on constate **peu de différences entre les résultats de Mme Chabrol et les nôtres.** La résolution du programme linéaire par Gurobi n'a pas permis de réduire le nombre de déçus qui se voyaient attribuer un projet non choisis.
Puisqu'il y a 42 binômes ($n = 42$) et 38 projets choisis, on conclut dans un premier temps que **4 binômes étaient obligés d'avoir un** 4^{ieme} **choix.**

Note : On remarque que 4 binômes souhaitaient le projet 100 juste en premier choix, 9 binômes souhaitaient le projet N° 114 ou encore 10 binômes pour le N° 124.

6.2 Instance de test N° 2

6.2.1 Présentation des résultats

L'instance N° 2 se présente de la manière suivante :
Il y a 50 projets pour 35 binômes ($n = 35$, $m = 50$). Néanmoins, uniquement 36 projets ont été choisis parmi les 3 choix des 35 binômes. **Ainsi, on constate qu'il y a plus de projets choisis que de binômes.**

Le premier tableau donne les résultats que trouve Mme Chabrol quand elle fait elle même l'affectation :

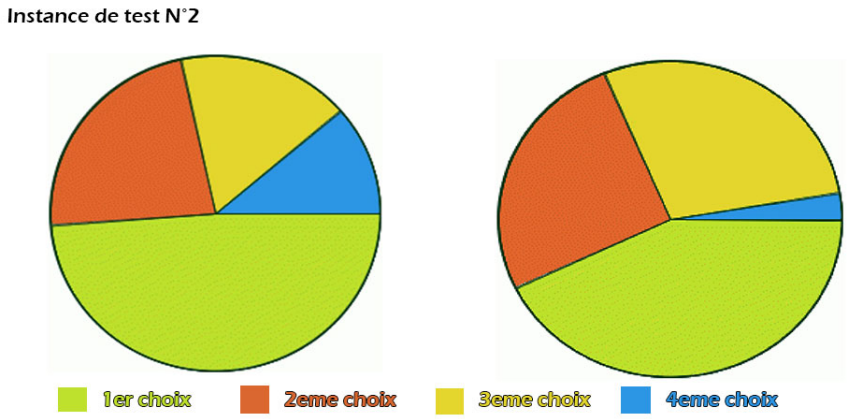
i^{ieme} choix affecté	Nb. de binômes affectés à leur i^{ieme} choix
1	17 (48,8%)
2	8 (22,8%)
3	6 (17,1%)
4	4 (11,4%)
TOTAL	35 (100%)

Note : Les résultats présentés ont été faits avec les poids suivants :
$p_1 = 1$, $p_2 = 2$ *et* $p_3 = 3$.

Par la résolution avec Gurobi nous obtenons le tableau suivant :

i^{ieme} choix affecté	Nb. de binômes affectés à leur i^{ieme} choix
1	15 (42,9%)
2	9 (25,7%)
3	10 (28,6%)
4	1 (2,8%)
TOTAL	35 (100%)

Pour mieux se rendre compte de la différence entre les 2 résultats (celui de Mme Chabrol à gauche et le notre à droite), mettons-les côte à côte :



6.2.2 Conclusion de cette instance

Pour cette deuxième instance, **on constate de fortes différences entre les résultats de Mme Chabrol et les nôtres.** La résolution du programme linéaire par Gurobi a permis de **réduire le nombre de déçus qui se voyaient attribuer un projet non choisi.**
De plus, nous pensons qu'avec de telles données d'entrée, i.e. le nombre de projets choisis et le nombre de binômes, il est difficilement possible de faire mieux (i.e. de n'avoir aucun 4^{ieme} choix).

6.3 L'importance des poids

Revenons rapidement sur l'importance de la valeur des poids p_i. Nous avons constaté que **pour la résolution, la valeur des poids ne permet pas de diminuer le nombre de 4^{ieme} choix**. Pour s'en rendre compte, faisons tourner l'instance de test n° 1 en ne modifiant que les poids (nous gardons $n = 42$, $m = 47$, $k = 3$).

Voici un tableau récapitulatif :

poids p_1 p_2 p_3	Nb. de binôme ayant obtenu leur i^{ieme} choix
1 4 9	choix 1 : 19 choix 2 : 7 choix 3 : 7 choix 4 : 9
1 10 100	choix 1 : 19 choix 2 : 7 choix 3 : 7 choix 4 : 9
1 100 100000	choix 1 : 19 choix 2 : 7 choix 3 : 7 choix 4 : 9

On constate ainsi que **la modification des poids ne permet pas de minimiser davantage le nombre de déçus, ni d'augmenter le nombre de 1^{ier} choix**.
Néanmoins, ce tableau ne veut pas dire que les résultats sont en tout point identiques. En effet, on constate que certains projets ont été attribués à d'autres binômes tout en gardant le même nombre de 1^{ier}, 2^{ieme} et 3^{ieme} choix.

7 Conclusion générale

7.1 Réduction du nombre de 4^{ieme} choix

Pour conclure sur ces 2 instances de test qui mettent à l'épreuve notre résolution linéaire avec Gurobi dans des cas concrets d'affectation de choix, **nous constatons qu'atteindre "0 binôme avec un 4^{ieme} choix" n'est pas forcément évident, même lorsque le nombre de projets choisis est supérieur au nombre de binômes.**

Néanmoins, nous avons pu vérifier que notre résolution fonctionnait et qu'elle permettait, **au pire, de faire aussi bien que Mme Chabrol à la main, au mieux, de réduire le nombre de binômes déçus.**

7.2 Gestion des binômes sans projet

Lorsque que le binôme se retrouve sans projet, c'est-à-dire avec un 4^{ieme} choix, le tableau qui contient la liste de tous les projets disponibles pour cette année devient utile.
En effet, **pour chaque binôme sans projet, nous regardons dans cette liste quels sont les projets qui n'ont pas été affecté et nous l'attribuons à ce binôme**.
Ainsi, en sortie de programme, chaque binôme, à un projet qui lui est attribué, que ce soit un projet qu'il désirait ou non.

8 Annexe

Petit tutoriel pour utiliser correctement la bibliothèque gurobi :

1. Lancer un environnement gurobi : *GRBloadenv()*.
2. Créer un modèle vide dans cet environnement : *GRBnewmodel()*.
3. Ajouter les variables de décision dans le tableau *obj* et donner le type de chaque variable dans le tableau *vtype*. *vtype[i]* correspond au type de la variable *obj[i]*.
   ex : GRB_BINARY pour une variable binaire,
4. Ajouter ces deux tableaux dans le modèle : *GRBaddvars()*.
5. Pour chaque contraintes :
   - ajouter les indices des variables concernés dans le tableau *ind*.
   - ajouter les coefficients associés à ces variables dans le tableau *val*.
6. Ajouter les contraintes dans le modèle : *GRBaddconstr()*.
7. Optimiser le modèle : *GRBoptimize()*.
8. Récupérer les valeurs des variables de décisions grâce au tableau *sol*, *sol[i]* correspondant à la valeur de la variable *obj[i]*.
9. Ne pas oublier de libérer l'environnement et le modèle : *GRBfreemodel()*, *GRBfreeenv()*.