Projet ZZ1 - Affectation des choix de projets pour les ZZ1

Antony Vigouret et Hugo Bayoud

15 mai 2019

Table des matières

1 Introduction

Chaque année, il est demandé aux ZZ1 de participer en binôme à un projet env 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 projet proposés par les professeurs ou intervenants de l'ISIMA dont mis à disposition des binôme sans sujet. Il leur ai demandé de choisir parmi cette liste de sujet, 3 d'entre eux qui les intéresserai de traiter et parmi ces 3, de les classer par ordre d'affinité.
Une affectation est ensuite réalisée pour proposer à chacun de ces binôme un projet.
Nous nous proposons ici de créer ce programme informatique qui permet d'affecter les choix de projet à chaque binôme en réduisant au maximum les déçus.

2 Modélisation

Nous cherchons dans un premier temps à **modéliser au mieux le problème que nous avons en entrée.**

2.1 Introduction des notations

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

On considère qu'il y a :

- n **binômes.**
- m **projets** (tel que $n \leq m$).
- **Chaque binôme peut faire i choix différents** et les classer (on considérera tout au long de ce projet que $i = 3$ comme c'est le cas actuel dans l'affectation des projets).
- Un entier p (qui prend des valeurs entières) pour chaque binôme qui permet de savoir, pour chaque projet choisi, quel est le degré d'importance **(poids)** dont sa valeur peut peut-être jouer un rôle dans l'affectation.

2.2 Problématique

Il convient désormais de répondre à la problématique du sujet :

Comment affecter un projet aux n binômes en évitant au maximum les déçus ; i.e proposer à plus de binôme possible leur 1^{ier} choix.

3 Fichiers d'entrée et sortie

Expliquons ce que nous avons besoin comme données à l'entrée du programme qui permet de résoudre le problème mathématique et quel en sont les données de sortie dont nous avons besoin pour savoir à quel binôme est affecté quel projet.

3.1 Fichier d'entrée

En entrée, nous nous attendons à recevoir un tableau de la sorte. Ce tableau que nous modéliserons ensuite en matrice pour pouvoir l'exploiter dans le programme.

N° Binôme	Choix projet 1 poids p1	Choix projet 2 poids p2	···	Choix projet i poids pi
1	20	14	···	2
2	15	5	···	20
···	···	···	···	···
···	···	···	···	···

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

```
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, i.
- Le **deuxième ligne** correspond aux poids. (il y a i 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 est été choisis ou non par un binôme. Nous reviendrons sur son utilité un peu plus tard.

3.2 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** (l'ensemble des 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
1	32	2
2	12	-1
...	...	...

Note : Ici, le binôme n° 1 a obtenu le projet n° 32, c'était son 2^{ieme} *choix. Le binôme n° 2 obtenu le projet n° 12, ce n'était pas pas un de ses i premier choix, il a fallut lui attribuer un projet qu'aucun binôme n'a choisi.*

4 Généralisation du problème

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

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

4.1 Données d'entrée

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

$$\begin{array}{ccccc} n & m & i & & \\ p_1 & p_2 & \cdots & p_i & \\ & & & & \\ id_1 & c_{1,1} & c_{1,2} & \cdots & c_{1,i} \\ id_2 & c_{2,1} & c_{2,2} & \cdots & c_{2,i} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ & & & \vdots & \\ id_n & c_{n,1} & c_{n,2} & & c_{n,i} \end{array}$$

Où,

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

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

$\forall (u, v) \in [\![1, n]\!] \times [\![1, i]\!]$, $c_{u,v}$ est le n° de projet que le u^{ieme} binôme classe en v^{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.

4.2 Matrice C

A partir du fichier d'entrée ; nous créons une **structure de données (écriture en C) pour permettre de construire une matrice qui sera utilisée pour la résolution** avec Gurobi.

Une matrice (notée C) à n lignes et m colonnes (**chaque coefficient de C correspond à un couple [N° binôme ; N° projet]**) qui contient un entier N (compris entre 0 et i) qui correspond au n° du projet choisi pour un binôme donné.
(ex : si chaque binôme peut faire 3 choix ($i = 3$), alors pour chaque ligne de C, il y aura un 1, un 2 et un 3, le reste : 0)

Il apparaît donc **plusieurs contraintes sur cette matrice**,

- La **somme de chaque ligne de C est égale à la somme des N premiers entiers.** (ex : pour $i = 3$, la somme de chaque ligne devra faire exactement $1+2+3 = 6$).
- La **somme de chaque colonne doit faire entre 0** (aucun des n binômes ne choisit ce projet) **et i*n** (tous les binômes souhaitent le même projet comme dernier choix par exemple).

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

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

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

4.3 Matrice X

Le format de sortie des données par Gurobi peut être une matrice (notée X) à n lignes et m colonnes qui contient l'affectation des projets à chaque binôme. Elle contient des 1 ou 0 en lien avec la matrice C.

- 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 C, **il ne faut donc pas qu'il y ait plus d'un coefficient différent de 1 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.**

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

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

Contraintes : $\forall i \in [\![1,n]\!],\ \sum_{j=1}^{m} x_{ij} = 1 \ \ et \ \ \forall j \in [\![1,m]\!],\ \sum_{i=1}^{n} x_{ij} \ \leq \ 1$

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

Fonction à MINIMISER :

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

5 Un exemple d'un cas simple $n = 5$

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 :

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

Soit C notre matrice d'entrée,

$$C = \begin{pmatrix} 1 & 0 & 3 & 0 & 2 \\ 0 & 2 & 0 & 1 & 3 \\ 2 & 0 & 3 & 1 & 0 \end{pmatrix}$$

Expliquons la première ligne. Comme nous l'avons dit 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. Ainsi, la $c_{1,1} = 1$. Le projet n° 2 n'étant pas été choisi par ce binôme, $c_{1,2} = 0$. Le projet n° 3 est le troisième choix du binôme, $c_{1,3} = 3$; le projet n° 5 étant le deuxième choix du binôme, $c_{1,5}$ vaut 2.

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

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

On peut enfin écrire la fonction à minimiser qui reprend les coefficients de la matrice C différents de 0 et les multipliant par le poids associé :

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

6 Présentation du code source

6.1 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 permettre 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.

6.2 Installer Gurobi

Gurobi peut être installer sur Linux ou Windows mais nous n'expliquerons que l'installation sous Linux.
On peut tout retrouver ici : http ://www.gurobi.com/documentation/8.1/quickstart$_{l}inux/obtaining_{a}g urobi_{l}ice$

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'executable, taper : ./mip1_c

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

6.3 Structure du code

Pour ce projet nous adoptons la structure suivante :

```
│
│   main.c
│   makefile
│   Test_NonG.txt
│
├──include
│     └─ // FICHIERS INCLUDES (.h)
│
└──src
      └─ // FICHIERS SOURCES (.c)
```

Le fichier "*Text_NonG.txt*" correspond au fichier d'entrée qui contient l'ensemble des informations sur les choix de projets que chaque binôme et de leur poids respectifs. Le fichier d'entrée est implémenté de la manière suivante dans

"*matrice.h*". Cette structure regroupe l'ensemble des informations utiles qui sont données en entrée :

```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 'Nombre_de_projet'. Or **gurobi a besoin d'entiers allant de 1 à 'Nombre_de_projet' pour la résolution.** Nous créons donc un **tableau de correspondances** pour récupérer le vrai numéro de projet et donner un fichier de sortie que Mme Chabrol peut lire facilement.

6.4 Utilisation et Commandes

Les commandes sont les suivantes :

- Lancer la compilation "*make*".
- Lancer les tests du programme "*./prog*"
  1. "*./prog*1" : lancement de gurobi avec le fichier d'entrée "*Test_NonG.txt*".
  2. "*./prog*2" : lancement de la génération d'un fichier "*Test_G.txt*" d'entrée en demandant nbBinome, nbProjet, nbChoix, puis lancement de gurobi.

Note : Des tests sont faits dans la fonction "main" pour être sûr que l'argument en entrée, lors de l'exécution de "./prog", soit bien un 1 ou un 2.

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.

6.5 Fonctions, prototypes

Dans cette partie nous allons détailler les fonctions disponibles dans le code.

6.5.1 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 dans ce rapport.

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

Signature : `Datas_t* FichierEnStructureDatas(FILE *fic);`

— Entrée :

- fil (FILE *) : Pointeur du fichier d'entrée qui contient l'ensemble des données d'entrée.

— 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|.

Signature : `void CreationFichierDeSortie(double *sol, Datas_t *Datas);`

— Entrée :
  - Datas (Datas_t *) : Pointeur de la structure de données.
  - sol (double *) : Pointeur de double contenant les solutions (après résolution par gurobi).

— Sortie
  - void.

6.5.2 Générer un fichier d'entrée

Le deuxième exécutable se présente à quelques choses près, comme le premier. Néanmoins, ce deuxième exécutable n'est là que pour permettre de tester gurobi en générant un fichier d'entrée de façon aléatoire. La génération d'un fichier d'entrée pour la résolution est implémentée dans "*generer_entree.c*". **La génération de fichier d'entrée permet uniquement de tester notre résolution via Gurobi. Elle n'est pas vouée à rester si les seuls fichiers d'entrée sont ceux donnés par Michelle Chabrol** une fois par an lors de l'affectation des choix de projets aux ZZ1.

Une fonction est disponible pour générer un fichier d'entrée avec le nombre de binôme, projet et choix par binôme souhaités pour ce fichier.

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

La fonction "*GenerateurDonneesEntree*" :
Génère automatiquement un fichier d'entrée exploitable par la suite pour gurobi.

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.

6.5.3 Lancement de Gurobi

Deux fonctions sont disponibles pour appeler un fichier d'entrée puis lancer la résolution avec gurobi. Ces fonctions sont implémentées dans : "$GUROBI.c$"

La fonction "$LancementGurobi1$" :
Permet de lancer la résolution avec gurobi. Que le fichier d'entrée doive être généré ou non, "$LancementGurobi1$" est appelée.

Signature : `void LancementGurobi1(char *Fichier);`

— Entrée :
  - void.

— Sortie
  - void.

La fonction "$LancementGurobi2$" :
Permet de faire appel à la fonction "$GenerateurDoneesEntree$" puis de lancer gurobi via "$LancementGurobi1$".

Signature : `void LancementGurobi2();`

— Entrée :
  - void.

— Sortie
  - void.

7 Instance 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 binôme ne peuvent avoir un des 3 projets qu'il souhaitait car le programme linéaire ne peux résoudre le problème en affectant à tous les binôme un projet.
En effet, on peut imaginer le cas ou le nombre de binôme est supérieurs au nombre de projet distincts choisis. Il y aura donc obligatoirement des binômes qui se retrouveront avec un projet qui ne fait pas partie de leurs 3 premiers choix.
Appelons un tel projet, un 4^{ieme} choix.

7.1 Instance de test N° 1

7.1.1 Présentation des résultats

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

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

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%)

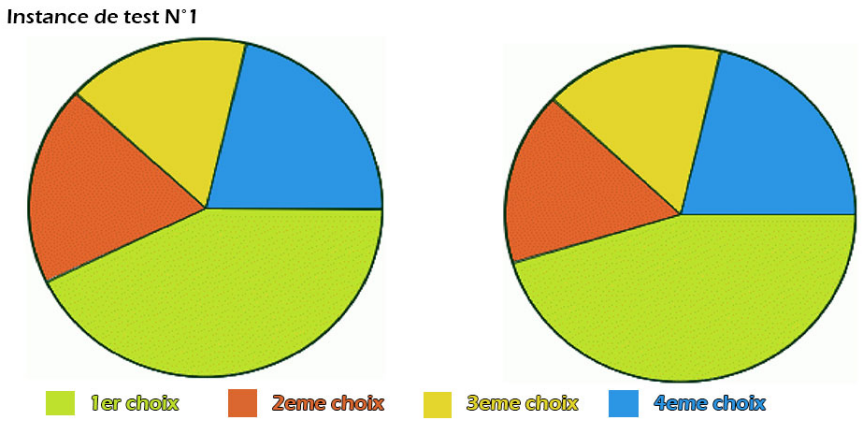
Note : Les résultats présentés ont été faits avec les poids suivants : 1 pour le 1^{ier} choix, 2 pour le 2^{ieme} et 3 pour le 3^{ieme}.

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	15 (45,2%)
2	9 (16,7%)
3	10 (16,7%)
4	1 (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 :



7.1.2 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 N° 4.
Puisqu'il y a 42 binômes ($n = 42$) et 38 projets choisis ($m = 38$), 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.

7.2 Instance de test N° 2

7.2.1 Présentation des résultats

L'instance N° 2 se présente de la manière suivante :
Il y a 50 projets pour 35 binômes ($n = 35$). Néanmoins, uniquement 36 projets ($m = 36$) 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 est fait elle même l'affectation :

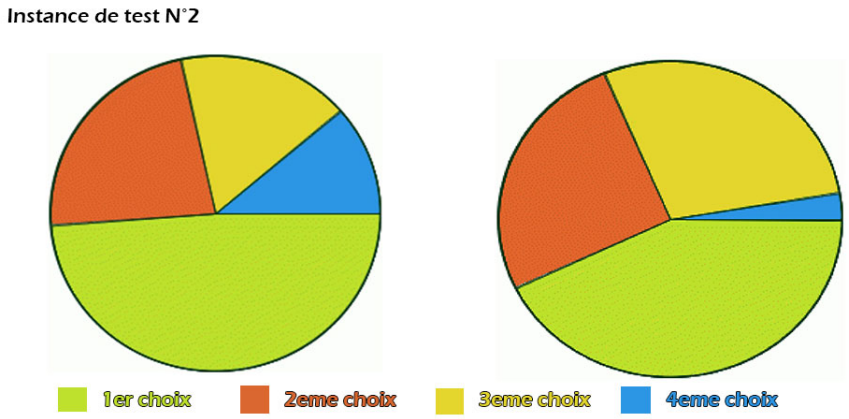
i^{ieme} choix affecté	Nb. de binômes affectés à leur i^{ieme} choix
1	17 (48,8%)
2	8 (22,8%)
3	6 (17,1%)
4	4 (11,4%)
TOTAL	35 (100%)

Par la résolution avec Gurobi nous obtenons le tableau suivant :

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

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 et le notre), mettons-les côte à côte :



7.2.2 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 N° 4 alors qu'il était possible de l'éviter.**
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).

7.3 L'importance des poids

Revenons rapidement sur l'importance de la valeur des poids p_i. Pour la résolution, la valeur des poids ne permet pas de diminuer le nombre de 4^{ieme} choix. En effet, on ne constate pas une grande différence de résultat suivant les valeurs des poids. Pour s'en rendre compte, faisons tourner l'instance de test n° 1 en ne modifiant uniquement les poids (nous gardons $n = 42$, $m = 47$, $i = 3$).

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 le 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 identique. En effet, on constate que certains projets ont été attribués à d'autres binôme tout en gardant le même nombre de 1^{ier}, 2^{ieme} et 3^{ieme} choix.

8 Conclusion générale

8.1 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.**

8.2 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 ce fichier quels sont les projets qui n'ont pas été affecté et nous l'attribuons à ce binôme.
Ainsi, en sortie de programme, chaque binôme, qu'il est ou avoir un projet qu'il désirait ou non, un projet lui est attribué.