TME, semaines 1 à 4

1 Problème et affectation

On s'intéresse au problème d'affection des étudiants dans le master informatique de Sorbonne Université. Ce master comporte 9 parcours : ANDROIDE, BIM, DAC, IMA, RES, SAR, SESI, SFPN, STL. Vous pouvez aller sur le site du master, à l'adresse `www-master.ufr-info-p6.jussieu.fr/`, pour avoir une description de ces différents parcours. Le recrutement dans chacun de ces parcours se fait selon les dossiers, et selon les vœux exprimés par les étudiants. L'objectif de cet exercice est d'utiliser l'algorithme de Gale-Shapley pour déterminer des affectations stables dans ce problème. Commencer par récupérer les fichiers "PrefSpe.txt" et "PrefEtu.txt" sur le Moodle de l'UE.

Fichier PrefSpe.txt : il correspond aux préférences des parcours sur un groupe fictif de 32 étudiants. La première ligne contient le nombre d'étudiants (ici 11). La deuxième contient les capacités d'accueil des parcours (2 pour ANDROIDE, 1 pour BIM, ...). Les 9 lignes suivantes correspondent aux préférences des parcours sur les étudiants. Par exemple, la ligne "0 ANDROIDE 7 9 [...] 2" signifie que l'étudiant 7 est classé premier par le parcours numéroté 0 (qui est ANDROIDE), que l'étudiant 9 est classé deuxième par ce même parcours, etc et que l'étudiant 2 est le dernier. Les parcours sont numérotés de 0 à 8.

PrefEtu.txt : il correspond aux préférences des étudiants. La première ligne contient le nombre d'étudiants (ici 11). Puis, le fichier contient une ligne par étudiant donnant ses préférences sur les parcours. Par exemple, la ligne "0 Etu0 5 [...] 4" signifie que l'étudiant numéro 0, nommé Etu0, a classé le parcours SAR (numéro 5) premier, etc et que le parcours RES (numéro 4) est le dernier de son classement.

Q1. En langage Python, écrivez une fonction lisant le fichier des préférences des étudiants sur les masters (PrefEtu.txt) et qui retourne une matrice des préférences correspondantes (ou une liste de listes). Écrivez une autre fonction pour les préférences des spécialités sur les étudiants.

Note : deux fichiers contenant quelques instructions utiles en Python vous sont fournis ("main.py" et "exemple.py"). Si vous n'avez jamais utilisé Python, lisez la section 4.

Q2. Codez l'extension de l'algorithme de Gale-Shapley au problème des hôpitaux "côté étudiants", et appliquez l'algorithme sur les deux fichiers tests.

Q3. Proposez, puis codez, une adaptation de Gale-Shapley "côté parcours". Appliquez l'algorithme sur les deux fichiers tests.

Q4. Écrivez une méthode prenant en entrée une affectation (et les matrices de préférences), et renvoyant la liste des paires instables. Vérifiez que les affectations obtenues dans les questions précédentes sont stables.

2 Evolution du temps de calcul

Q5. Ecrivez deux méthodes prenant en paramètre un nombre n d'étudiants :
- l'une générant un tableau des préférences de ces n étudiants sur les 9 parcours du master (préférences aléatoires),
- l'autre générant un tableau des préférences des 9 parcours du master sur les n étudiants. Les préférences seront aléatoires. Pensez à définir les capacités d'accueil des parcours (la somme devant faire n). On pourra générer des capacités de manière déterministe (et par exemple à peu près équilibrées entre les parcours).

Q6. Mesurez le temps de calcul de votre algorithme de Gale-Shapley pour différentes valeurs de n. Vous ferez plusieurs tests pour chaque valeur de n pour avoir une valeur significative). On pourra par exemple faire varier n de 200 à 2000 par pas de 200, et faire 10 tests pour chaque valeur de n. On pourra tracer une courbe représentant le temps de calcul (moyen) en fonction de n.

Q7. Quelle complexité obtient-on ? Est-ce cohérent avec la complexité théorique ?

Q8. (facultatif) Faites de même avec le nombre d'itérations. Combien fait-on d'itérations en moyenne ? Est-ce cohérent avec une analyse théorique de l'algorithme ?

3 Equité et PL(NE)

On souhaite maintenant trouver une solution (pas forcément stable) qui maximise l'utilité minimale des étudiants. Autrement dit, on veut trouver le plus petit k tel qu'il existe une affectation où tout étudiant a un de ses k premiers choix (utilité au moins $m - k$ pour chaque étudiant).

Q9. Ecrivez, pour un certain k, un PLNE permettant de savoir s'il existe une affectation où tout étudiant a un de ses k premiers choix.

Q10. Sur l'exemple, et pour $k = 3$, écrivez une méthode permettant de générer un fichier .lp correspondant au PLNE, cf section 5.

Q11. Résolvez le PLNE à l'aide de Gurobi (cf section 5). Existe-t-il une solution ?

Q12. Trouvez le plus petit k tel qu'il existe une solution. Décrivez une solution maximisant l'utilité minimale. Quelle était l'utilité minimale des solutions fournies par l'algorithme de Gale-Shapley (côté étudiants et côté parcours) ?

Q13. (facultatif) Ecrivez un PLNE maximisant l'utilité totale des étudiants, puis résolvez-le avec Gurobi. Quelle utilité moyenne obtient-on ?

Q14. (facultatif) Soit k^* le plus petit k trouvé à la question 15. Ecrire un PLNE maximisant l'utilité totale des étudiants parmi les solutions où chaque étudiant a un de ses k^* premiers choix.

Q15. (facultatif) Comparez les différentes solutions obtenues (GS côté étudiant, GS côté parcours, solutions des questions 15, 16 et 17) : stabilité, utilité moyenne, utilité minimale. Si vous avez fait la question 5, comparez aussi le nombre de paires instables dans chacune des solutions calculées.

4 Vous n'avez jamais utilisé Python ?

Pas d'inquiétude. Pour cette partie nous n'aurons besoin que de notions basiques d'algorithmique, et vous saurez facilement adapter la syntaxe (écriture de boucles, tests, écriture de fonctions,...) à ce langage proche d'un pseudo-code.

Afin de vous aider, vous pouvez télécharger les fichiers `main.py`, `exemple.py` et `test.txt`.

Lancez alors `Spyder`. Ouvrez et lisez les deux fichiers `.py` téléchargés précédemment.

Placez-vous sur le fichier `main.py`, puis cliquez sur le petit triangle (*Exécutez le fichier*). En bas à droite de la fenêtre apparaissent les affichages du code. Ce fichier fait appel aux fonctions définies dans le fichier `exemple.py` (commande `import exemple.py`). Vous trouverez dans ce fichier quelques-unes des commandes utiles pour votre projet. Pour le reste, je vous fais confiance !

5 Fichiers .lp et Gurobi

5.1 Fichier .lp

Le "format .lp" est un format standard d'écriture d'un programme linéaire.

En voici un exemple :

```
 Maximize
obj: x1 + 2 x2 + 3 x3 + x4
Subject To
c1: - x1 + x2 + x3 + 10 x4 <= 20
c2: x1 - 3 x2 + x3 <= 30
c3: x2 - 3.5 x4 = 0
Bounds
0 <= x1 <= 40
-2 <= x4 <= 3
Binary
x3 x4
End
```

Aide : pour des variables x_{ij} à deux indices, pensez à séparer les indices dans vos noms de variables (par exemple x_{2_3}). Attention également à bien mettre des espaces (entre coefficients, variables, ...) comme dans l'exemple ci-dessus.

5.2 Solveur

Nous allons utiliser le solveur Gurobi pour résoudre le(s) PLNE. Gurobi prend en entrée un fichier (par exemple au format lp). Avant de pouvoir l'utiliser directement, taper les 3 commandes ci-dessous :

```
export GUROBI_HOME="/opt/gurobi801/linux64"
```

```
export PATH="${PATH}:${GUROBI_HOME}/bin"
```

```
export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:${GUROBI_HOME}/lib"
```

Maintenant, en vous plaçant dans le répertoire contenant votre fichier `fichierpl.lp`, vous pouvez taper la commande :

```
/opt/gurobi801/linux64/bin/gurobi_cl ResultFile=affectation.sol fichierpl.lp
```

La solution trouvée est alors décrite dans le fichier `affectation.sol`.

Voici une solution alternative (ceci évite d'avoir a retaper les commandes ci-dessus à chaque session) :

- Allez à la racine de votre répertoire, et regardez si vous avez un fichier .bashrc (>ls -a car fichier caché).
- Si c'est le cas, ajoutez à la fin de votre fichier les 3 commandes :
  `export GUROBI_HOME="/opt/gurobi801/linux64"`
  `export PATH="${PATH}:${GUROBI_HOME}/bin"`
  `export LD_LIBRARY_PATH="${LD_LIBRARY_PATH}:${GUROBI_HOME}/lib"`
- Sinon, créer un fichier nommé .bashrc contenant ces trois commandes à la racine de votre répertoire.
- Déloguez-vous, reloguez-vous.