Projet

TME 1-3

Ines HARRAOUI, Yanis SAADI DIT SAADA

1 Introduction

Ce projet a pour objectif d'étudier et d'implémenter le problème du mariage stable, un problème d'appariement où dans notre cas deux groupes, étudiants et masters, expriment des préférences mutuelles, et où l'on cherche à former des paires de manière stable, c'est-à-dire sans qu'aucun couple ne puisse s'améliorer en modifiant son appariement. Nous nous intéresserons également à certaines variantes du problème. Pour résoudre ce problème, nous utiliserons l'algorithme de Gale-Shapley, qui garantit une solution stable, ainsi que des méthodes de programmation linéaire pour explorer d'autres approches d'optimisation.

2 Description du code et des structures manipulées

2.1 Structures utilisées

- Les matrices de préférences 'cE', 'cP' contiennent, respectivement, la liste des préférences des étudiants et des parcours.

- La matrice 'tabPref' est utilisée pour retrouver rapidement la position d'un étudiant dans un master (et inversement). Par exemple, tabPref[j][i] donne la position de l'étudiant i dans les préférences du master j.

- La file FIFO (Queue) est utilisée pour gérer les étudiants ou les masters qui doivent encore proposer avec l'ensemble des operations en $O(1)$.

- Le tas est utilisé dans GS etu pour stocker les étudiants affectés à chaque master, permettant de trouver le moins préféré en $O(1)$, et de l'échanger en $O(log(n))$.

- Le tableau 'suivi' indique le nombre de propositions faites par chaque étudiant(ou master).

- Le dictionnaire 'affectation' associe chaque étudiant au master qui lui a été attribué.

2.2 Organisation du code

Le code est divisé en plusieurs fichiers :

- 'Projet.py' contient les fonctions de Gale-Shapley.

- 'PL.py' contient les fonctions des PLNE.
- 'func_tas.py' contient les fonctions liées au tas.

- 'main.py' contient un jeu de tests exécutant les algorithmes de Gale-Shapley ainsi que l'ensemble des exercices du projet.

2.3 Fonctions principales

- studentpref(file), masterpref(file), capacity_master(file) sont des fonctions de lecture de fichiers

- generate_cE(n, nb_parcours), generate_cP(n, nb_parcours) génèrent des tableaux de préférences aléatoires pour n étudiants et $nb_parcours$ parcours.

- time_gs() génère des préférences aléatoires, exécute GSetu et GSmaster pour différentes valeurs de n et mesure les temps d'exécution sous forme de graphique.

- paires_instables(prefetu, prefspe, affectes) recherche des paires qui préféreraient être ensemble plutôt qu'avec leur affectation actuelle.

- GS_etu : Chaque étudiant i postule à ses masters préférés ($cE[i]$). Un master j accepte l'étudiant i s'il a de la place, sinon il le compare avec le moins bien classé parmi ceux qu'il a déjà acceptés. Si l'étudiant i est mieux classé, il remplace l'autre, qui doit alors postuler ailleurs. L'algorithme continue jusqu'à ce que tous les étudiants soient affectés.

- GS_master : Chaque master i propose aux étudiants selon leur ordre de préférence ($cP[i]$). Si un étudiant accepte un meilleur master que celui qu'il possède, il quitte son master actuel, qui doit alors proposer à un autre étudiant. L'algorithme se termine lorsqu'aucune proposition ne peut être faite.

- PL_resolve_1 : Maximise le nombre d'affectation en faisant en sorte que chaque étudiant soit dans l'un de ses k premiers choix.

- PL_resolve_2 : Maximise l'utilité minimale des étudiants en faisant en sorte que chaque étudiant soit dans l'un de ses k premiers choix.

- PL_resolve_3 : Maximise la somme des utilités

- PL_resolve_4 : Maximise la somme des utilités avec la contraintes des $k*$ premiers choix

3 Description de l'algorithme paires instables

Etant donnés une liste des affectations et la liste des préférences de chacun, il s'agit dans cet algorithme, de s'assurer pour chaque étudiant, qu'il a bien son premier choix valide et s'il ne l'a pas eu, vérifier que c'est dû à sa position dans les préférences de son choix en question. Pour ce dernier, on va récupérer la position de l'étudiant le moins préféré affecté au master en question et la comparer avec l'étudiant qui n'a pas pu y être affecté.

4 Réponses aux questions

Question 2

1. Structure : file
   Complexité : $O(1)$

2. Structure : tableau de mise à jour du suivi des demandes
   Complexité : $O(1)$

3. Structure : matrice des positions des étudiants j pour le parcours i
   Complexité : $O(1)$

4. Structure : tas, la racine est l'étudiant le moins préféré
   Complexité : $O(1)$

5. Structure : mise à jour du tas en $O(log(n))$
   Complexité : $O(log(n))$

Question 3

Complexité sans initialisation : $O(n * log(n))$ car
- le parcours des étudiants est en $O(n)$,
- chaque étudiant fait, au plus, m propositions. $m = 9$ qui est négligeable,
- les opérations sur le tas de chaque parcours se font en $O(log(n))$.

Complexite avec initialisation : $O((n * m) + 2 * n + m + n * log(n)$ - La complexite ici contient toute les initialisation réaliser on remarque qu'on peut simplifier cette formule en $O(4 * n * log(n))$,4 étant une constante cela reviens a notre complexite sans initialisation !

Question 4

Chaque parcours i propose à chaque étudiant, dans son ordre de préférences. Si la capacite du parcours i est remplie et si le nouvel étudiant préfère le parcours i à son ancien parcours, alors, l'étudiant le moins préféré déjà affecté au parcours i est remplacé par le nouvel étudiant. Sinon, on ajoute simplement l'étudiant au parcours i.

Complexité sans initialisation : $O(n * m) = O(n * 9)$ soit $O(n)$
Complexite avec initialisation : $O((n * m) + 2 * n + m + n * m$ qu'on simplifie aussi.

Question 5

Étudiant	**Master**
7	7
9	2
5	0
3	0
10	4
1	5
0	6
4	1
6	8
8	3
2	8

Table 1: Affectation optimisée pour les masters

Étudiant	**Master**
0	5
1	6
4	1
5	0
7	7
9	2
10	4
3	0
6	8
2	8
8	3

Table 2: Affectation optimisée pour les étudiants

Question 9

En théorie, notre algorithme de Gale-Shapley est en $O(n * log(n))$ pour le côté etu et $O(n)$ pour le côté master.
En pratique, on analyse dans la section Anayse des performance que notre algorithme côté master est bien en $O(n)$. Cette complexité est cohérente car elle n'utilise pas de tas.
L'algorithme côté étudiant est bien en $O(n * log(n))$ et est moins efficace que le côté master.

Question 10

Côté étudiant :

Il y a n étudiants et chaque étudiant fait au maximum $m = 9$ propositions. À chaque proposition, l'étudiant est ajouté à un tas de taille n, ce qui fait $O(log(n))$. L'algorithme fait donc au plus $n * 9$ itérations.

Côté master :

Il y a $m = 9$ masters et chaque master peut proposer au plus à n étudiants.

L'algorithme fait donc au plus $n * 9$ itérations.

On s'attend à ce que le nombre d'itérations soit proportionnel à n, comme le montre le graphique ci-dessous, puisque chaque étudiant est limité en nombre de propositions. Le nombre d'itérations croît linéairement, ce qui est cohérent avec l'analyse théorique.

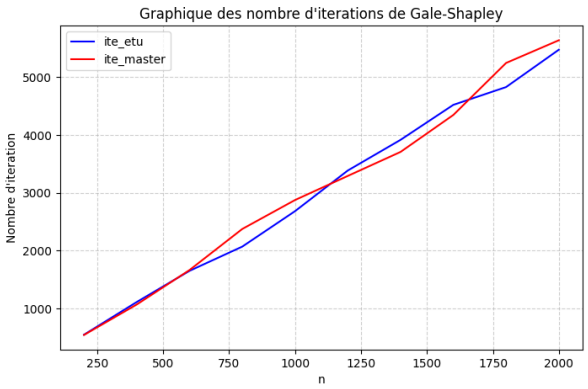

Figure 1: Nombre d'itérations moyen des algorithmes de Gale-Shapley en fonction du nombre d'étudiants n

Question 11

Variable : $x_{ij} = 1$ si l'étudiant i est affecté au parcours j

Fonction objective : $\max \sum_{i=0}^{n} \sum_{j=0}^{9} x_{i,j}$

Contraintes :

Soit E l'ensemble des paires (i, j) telles que l'étudiant i est dans les k premiers choix du parcours j et le parcours j est dans les k premiers choix de l'étudiant i.

- $\sum_{(i,j)\in E} x_{ij} \leq 1, \forall i \in \{1, ...n\}$: chaque étudiant i est affecté au plus à un parcours j.

- $\sum_{j=0}^{9} \sum_{i=0}^{n} x_{ij} \leq c_j$: tout parcours j est limité en nombre d'étudiants par sa capacité c_j.

- $\sum_{i=0}^{n} \sum_{j\in P_i[0:k]} x_{ij} = 1$: l'étudiant i ne peut êtré affecté qu'à un parcours figurant dans l'ensemble de ses k premiers choix que l'on note $P_i[0 : k]$.

- $x_{ij} \in \{0,1\}, \forall (i,j) \in E$: domaine des variables

Question 12

Il n'existe pas de solution pour $k = 3$.

Question 13

Il existe une solution pour $k = 5$.

Programme linéaire maximisant l'utilité minimale des étudiants (équité):

- Variables :

$x_{ij} = 1$ si l'étudiant i est affecté au parcours j

uti_{min} représente le minimum des utilités.

- Fonction objective : $\max uti_{min}$

- Contraintes :

- $\sum_{(i,j)\in E} x_{ij} \leq 1, \forall i \in \{1, ...n\}$: chaque étudiant i est affecté au plus à un parcours j.

- $\sum_{j=0}^{9} \sum_{i=0}^{n} x_{ij} \leq c_j$: tout parcours j est limité en nombre d'étudiants par sa capacité c_j.

- $\sum_{i=0}^{n} \sum_{j\in P_i[0:k]} x_{ij} = 1$: l'étudiant i ne peut êtré affecté qu'à un parcours figurant dans l'ensemble de ses k premiers choix que l'on note $P_i[0:k]$.

- $\sum_{i=0}^{n} \sum_{j=0}^{9} x_{ij} * u_i(j) \geq uti_{min}$: chaque étudiant a une utilité d'au moins uti_{min}.

- $x_{ij} \in \{0,1\}, \forall (i,j) \in E$: domaine des variables

Question 14

Programme linéaire maximisant la somme des utilités (efficacité):

Variable : $x_{ij} = 1$ si l'étudiant i est affecté au parcours j
Fonction objective : $max \sum_{i=0}^{n} \sum_{j=0}^{9} x_{ij} * (u_i(j) + u_j(i))$
avec $u_i(j)$ l'utilité de l'étudiant i pour le parcours j et $u_j(i)$ l'utilité du parcours j pour l'étudiant i

Contraintes :

- $\sum_{(i,j)\in E} x_{ij} = 1, \forall i \in \{1, ...n\}$: chaque étudiant i est affecté à exactement un parcours j.

- $\sum_{j=0}^{9} \sum_{i=0}^{n} x_{ij} \leq c_j$: tout parcours j est limité en nombre d'étudiants par sa capacité c_j.

- $\sum_{i=0}^{n} \sum_{j\in P_i[0:k]} x_{ij} = 1$: l'étudiant i ne peut être affecté qu'à un parcours figurant dans l'ensemble de ses k premiers choix que l'on note $P_i[0:k]$.

- $x_{ij} \in \{0,1\}, \forall (i,j) \in E$: domaine des variables

L'utilité moyenne obtenue est 14.18.

L'utilité minimale des étudiants obtenue est 3.

Question 15

Variable : $x_{ij} = 1$ si l'étudiant i est affecté au parcours j

Fonction objective : $max \sum_{i=0}^{n} \sum_{j=0}^{9} x_{ij} * (u_i(j) + u_j(i))$
avec $u_i(j)$ l'utilité de l'étudiant i pour le parcours j et $u_j(i)$ l'utilité du parcours j pour l'étudiant i

Contraintes :

- $\sum_{(i,j)\in E} x_{ij} = 1, \forall i \in \{1, ...n\}$: chaque étudiant i est affecté à exactement un parcours j.

- $\sum_{j=0}^{9} \sum_{i=0}^{n} x_{ij} \leq c_j$: tout parcours j est limité en nombre d'étudiants par sa capacité c_j.

- $\sum_{i=0}^{n} \sum_{j\in P_i(k*)} x_{ij} = 1$: l'étudiant i ne peut être affecté qu'à un parcours figurant dans représente l'ensemble des $k*$ premiers choix de l'étudiant i que l'on note $P_i(k*)$.

- $x_{ij} \in \{0, 1\}, \forall (i, j) \in E$: domaine des variables

Question 16

Nous avons comparé six méthodes d'affectation des étudiants aux masters selon trois critères : stabilité (nombre de paires instables), utilité moyenne des étudiants et utilité minimale des étudiants.

Stabilité des affectations

En ce qui concerne la stabilité des affectations, nos PLNE nous renvoient des affectations instables.

Les algorithmes de Gale-Shapley renvoient uniquement des paires stables. L'instabilité des PLNE est dû à l'absence d'une contrainte de stabilité.

Utilité moyenne des étudiants

Pl_resolve_3 offre la meilleure utilité moyenne (14.18) suivi de Pl_resolve_4 (14.0).

Les solutions Gale-Shapley offrent des utilités moyennes légèrement inférieures (13.45 côté étudiant, 13.36 côté master).

PL_resolve_1 offre l'utilité la plus basse (10.63) car il se concentre sur les affectations pour les k premiers choix et ne maximise pas l'utilité.

Utilité minimale des étudiants

PL_resolve_2 (maximisation de l'utilité minimale dans les k premiers choix) est, par définition, la meilleure solution pour ce critère, assurant une utilité minimale de 4.

PL_resolve_1 et PL_resolve_4 ont également une utilité minimale de 4 grâce à la contrainte qui limite les affectations aux k premiers choix.

Les solutions PL_resolve_3 et Gale-Shapley affichent une utilité minimale légèrement inférieure qui est justement dûe à l'absence de la contrainte des k premiers choix.

5 Description des jeux de tests

5.1 Algorithmes de Gale-Shapley + paires_instables

Pour les fonctions GS_etu et GS_master, nous les avons d'abord testées avec les fichiers PrefEtu.txt et PrefSpe.txt. Nous avons également crée deux petites matrices de préférences ainsi que les capacités des

masters exécutant ces mêmes fonctions puis en inversant deux paires stables afin de tester la fonction paires_instables.

5.2 Exercices

Dans le fichier main.py, on pourra retrouver les tests mesurant les temps d'exécution des algorithmes de Gale-Shapley (Exercice 1), calculant le nombre d'itérations des algorithmes de Gale-Shapley (Exercice 2) et comparant les paires instables, les utilités moyennes et les utilités minimales des étudiants pour toutes les solutions (Exercice 3). Les résultats sont générés dans le fichier Exercice.txt.

6 Analyse des performances

Les trois graphes montrent une augmentation du temps d'exécution avec n.
L'échelle logarithmique $log(n)$ montre que le temps d'execution n'est pas exponentielle.

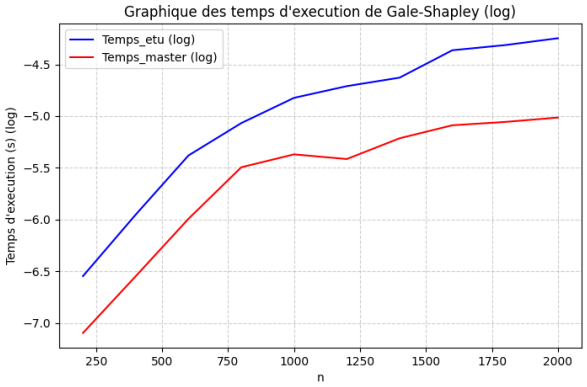

Figure 2: Temps d'exécution de l'algortihme de Gale-Shapley en fonction du nombre d'étudiants(logarithme népérien)

L'échelle racine montre une allure plus linéaire ce qui montre que la complexité de nos algorithmes se rapprochent plus d'une complexité quadratique qu'exponentielle.

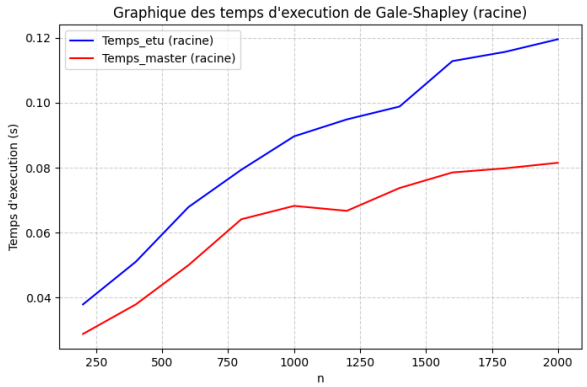

Figure 3: Temps d'exécution de l'algortihme de Gale-Shapley en fonction du nombre d'étudiants(racine)

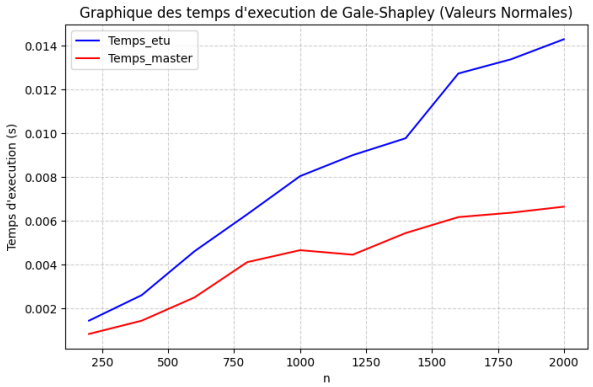

Figure 4: Temps d'exécution de l'algortihme de Gale-Shapley en fonction du nombre d'étudiants

Dans les trois graphes, la courbe 'Temps_etu' est toujours plus lente que la courbe 'Temps_master'. L'écart entre les deux courbes semble se creuser, à mesure que n augmente, montrant que GS_master est plus efficace.

7 Conclusion

Dans ce projet, nous avons exploré et mis en œuvre l'algorithme de Gale-Shapley, et avons comparé son efficacité avec celle d'approches basées sur la programmation linéaire (PLNE). Nos résultats ont montré que l'algorithme de Gale-Shapley garantit une stabilité des appariements, mais avec une utilité moyenne légèrement inférieure par rapport aux solutions optimisées par programmation linéaire. Cependant, ces approches PLNE entraînent souvent des affectations instables, ce qui souligne l'importance de la présence de stabilité dans la méthode de Gale-Shapley. En termes d'efficacité, l'algorithme de Gale-Shapley côté master est plus rapide, tandis que du côté étudiant, la complexité reste plus élevée, bien qu'elle ne dépasse pas un comportement quadratique. Enfin, notre analyse comparative des différentes méthodes a permis d'identifier les compromis entre équité, utilité et stabilité dans l'affectation des étudiants aux masters.