MINI-Projet 1

Laurrene LI & Hugo AMADO

Table des matières

1 Énoncé

Ce mini-projet consiste à attribuer les étudiants aux neuf parcours de master en fonction de leurs préférences, en appliquant l'algorithme de Gale-Shapley. L'objectif est de trouver une affectation stable, où aucun étudiant et aucun parcours ne préféreraient être associés autre qu'avec leur affectation assignée. Une analyse sera menée pour évaluer la complexité et le temps d'exécution de l'algorithme sur des instances de tailles croissantes. Enfin, l'affectation obtenue sera optimisée en termes d'équité et d'utilité à l'aide de la programmation linéaire en nombres entiers (PLNE), afin de comparer différentes solutions selon certaines contraintes.

2 Structure de données

2.1 Listes (list)

Utilisation : stocker des séquences de données (préférences, capacités, etc.)
Propriété : faciles à manipuler et à parcourir
Exemple : matrice_etu() et matrice_spe() retourne une liste de listes pour les matrices de préférences et capacite() retourne une liste des capacités d'accueil des parcours.

2.2 Dictionnaire (dict)

Utilisation : stocker des associations clé-valeur (étudiant → parcours, parcours → étudiants)
Propriété : accès rapide aux éléments par clé
Exemple : Dans gale_shapley_ce() et gale_shapley_cp() , affectations et etu_affecte_par sont des dictionnaire {étudiant : parcours}.

2.3 Ensembles (set)

Utilisation : stocker des éléments uniques (étudiants déjà affectés)
Propriété : vérifications d'appartenance en temps réduit.
Exemple : deja_prise dans gale_shapley_cp() est un ensemble pour stocker les étudiants déjà affectés.

2.4 File (deque)

Utilisation : gérer les étudiants ou les parcours libres dans l'algorithme de Gale-Shapley
Propriété : opérations popleft() et append() en temps réduit.
Exemple : libre dans gale_shapley_ce() est une file d'étudiants libres.

2.5 Tableaux numpy (numpy.array)

Utilisation : générer et mélanger des listes de préférences aléatoires.
Propriété : efficaces pour les opérations numériques
Exemple : np.arange() et np.random.shuffle() dans genere_mat_etu() et genere_mat_spe().

2.6 Tuples (tuple)

Utilisation : retourner des paires de valeurs (étudiant, parcours).
Propriété : immutables donc retourne des résultats sans fautes
Exemple : sorted((k, [v]) for k, v in affectations.items()) dans gale_shapley_ce retourne une liste de tuples (étudiant, [parcours]).

3 Description globale du code

3.1 Fichier probleme_affectation.py

Lecture de fichier

Les fonctions permettant de générer une matrice de préférences et la capacité d'accueil des parcours à partir d'un fichier : matrice_etu(fichier), matrice_spe(fichier), capacite_spe(fichier).

Gale Shapley

Les fonctions permettant de trouver une affectation stable selon les préférences des étudiants et des parcours et la capacité d'accueil de parcours :gale_shapley_ce(ce, cp, cap),gale_shapley_cp(ce, cp, cap).

Paires instables

La fonction, avec un mariage donné en comparant les préférences des hommes et des femmes, trouve les paires instables : paires_instables(mariage, cote_h, cote_f).

3.2 Fichier temps_calcul.py

Génération de préférences et de capacité

Les fonctions permettant de générer une matrice de préférences et la capacité d'accueil des parcours aléatoirement selon n étudiants : genere_mat_etu(n, spe=9),genere_mat_spe(n, spe=9), capacite(n).

Mesure des temps d'exécution

La fonction mesure les temps moyens d'exécution des algorithmes GS étudiants et GS parcours pour différentes valeurs de n sur 10 tests :temps(nb_test=10).

Visualisation graphique

La fonction trace des graphiques montrant l'évolution du temps de calcul pour GS étudiants et GS parcours en fonction du nombre d'étudiants : graphe() .

3.3 Fichier plne.py

Il génère un fichier .lp contenant un programme linéaire en nombres entiers (PLNE) qui maximise le nombre d'affectations optimales en équité, efficacité et utilité.

4. Description schématique des algorithmes

Fichiers : problèmes_affectation.html et temps_calul.html

5. Réponses aux questions

Q1.2. Pour pouvoir trouver un étudiant libre à chaque itération, nous pouvons construire une file composée d'étudiants libre permettant ainsi de récupérer avec un temps de complexité O(1).
Ensuite, nous initialiserons une liste stockant le nombre de propositions faites par chaque étudiant pour retrouver le prochain parcours à qui faire la proposition. Cette structure est de complexité O(1).
Puis nous utiliserons une liste de dictionnaire retrouvant les indices de préférences des parcours pour chaque étudiant ([{etudiant : indices}]. L'accès à un élément est dans une dictionnaire est O(1).
Par la suite, pour récupérer l'étudiant le moins bien classé en préférences, on utilisera une liste où nous stockerons les parcours affectées (O(1)).
Enfin, pour remplacer un étudiant dans l'affection, nous utiliserons des fonctions de manipulation de liste de complexité O(1).

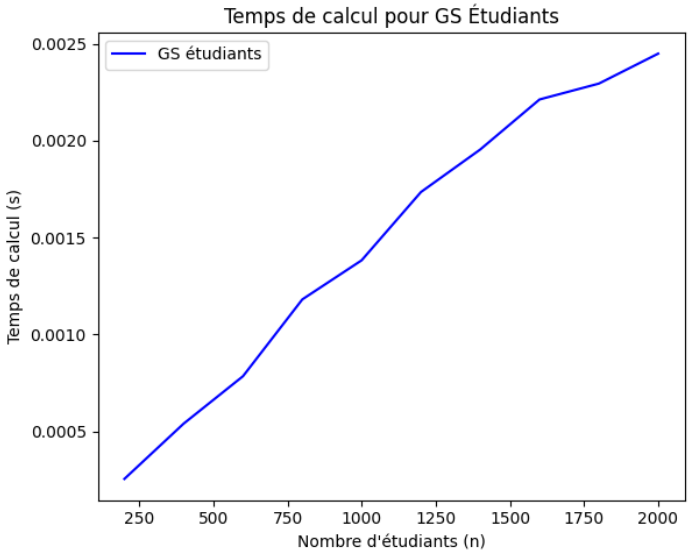
Q1.3. La complexité de l'initialisation vaut $O(n^2)$ en raison de prefEtuIndices, une variable stockant les indices des préférences des parcours sur les étudiants.
La complexité au niveau de la boucle vaut O(nombre de parcours) (ici, on a 9 parcours) car les opérations dans la boucle vaut O(1).
La complexité de la mise en forme du résultat en liste de tuples pour une meilleure lisibilité est de O(nlog n).
Ainsi, la complexité dominante de l'algorithme est située à la partie initialisation donc notre algorithme est $O(n^2)$.

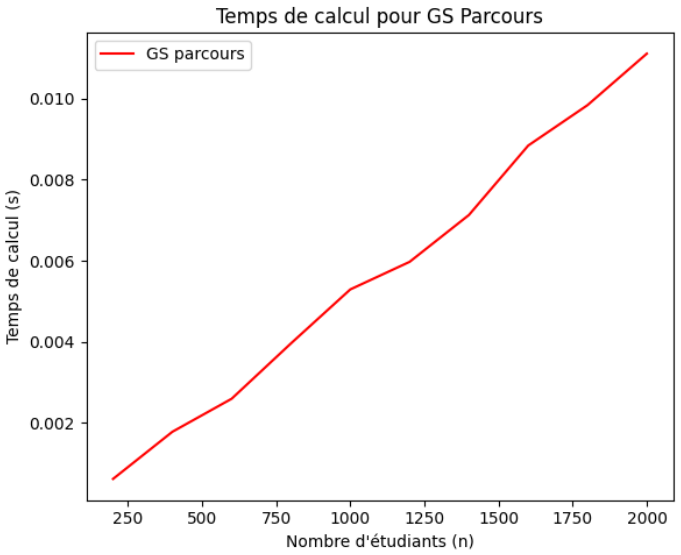
Q1.4. Nous utilisons une file (deque) contenant les parcours ayant encore des places disponibles. L'extraction du prochain parcours libre se fait en temps constant O(1).
Une liste permet de suivre le nombre de propositions envoyées par chaque parcours.
L'accès au prochain étudiant à qui faire une proposition est en temps constant O(1) en consultant l'élément correspondant dans cette liste.
Pour chaque étudiant, nous stockons un dictionnaire qui associe chaque parcours à son indice dans la liste de préférences de l'étudiant. Ainsi, l'accès à la position d'un parcours dans les préférences d'un étudiant se fait en temps constant O(1).
Pour déterminer l'étudiant le moins bien classé parmi ceux déjà affectés à un parcours, nous comparons les indices de préférence dans les listes de préférences des étudiants (O(n)).
Enfin, pour remplacer un étudiant dans l'affection, nous utiliserons des fonctions de manipulation de liste de complexité O(1).

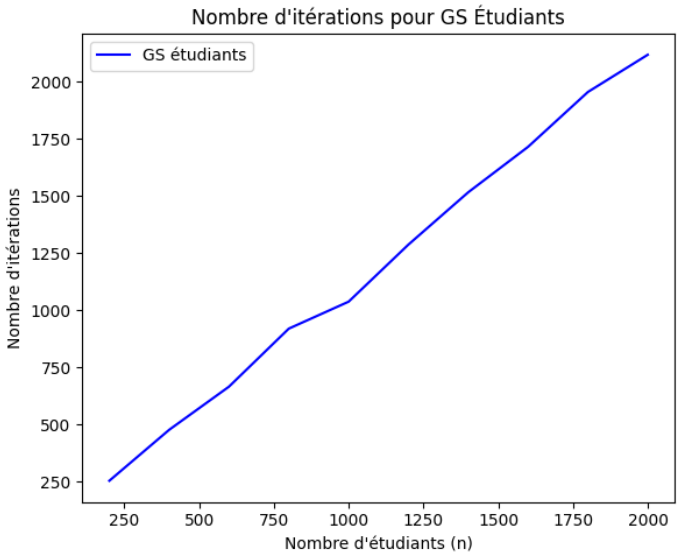
Q1.5. Le résultat d'affectation du coté étudiants est [(0, [5]), (1, [6]), (2, [8]), (3, [0]), (4, [1]), (5, [0]), (6, [8]), (7, [7]), (8, [3]), (9, [2]), (10, [4])] et du coté parcours [(0, [5, 3]), (1, [4]), (2, [9]), (3, [8]), (4, [10]), (5, [1]), (6, [0]), (7, [7]), (8, [6, 2])], ce qui représentent des affectations stables.

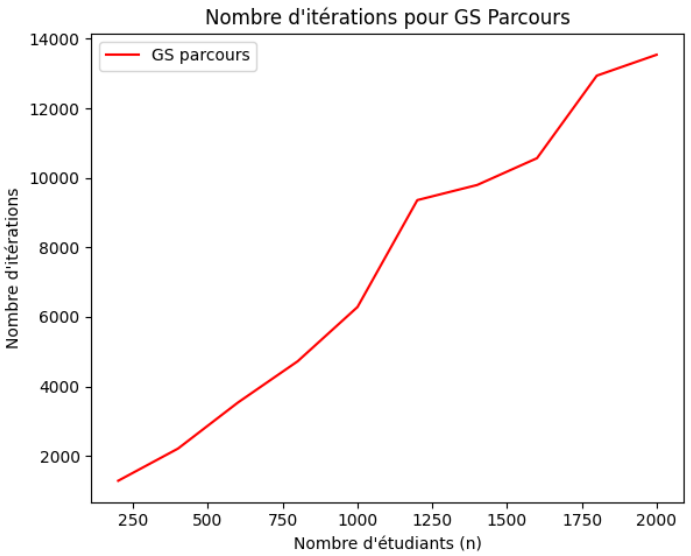
Q2.9/10. Les schémas ci-dessous représentent les graphes du temps d'exécution moyen et le nombre moyen d'itérations de l'algorithme Gale-Shapley du coté étudiant et du côté parcours. Les courbes bleue et rouge montrent une courbe un près linéaire (presque droites). D'après les analyses des questions de la première partie du projet, que l'algorithme de Gale-Shapley possède une complexité de

O(n²) dans le pire cas. Ainsi, les courbes ne correspondent pas à la complexité théorique. La différence est peut-être dû à une optimisation du coût des opérations et que nous ne atteignons pas le cas les plus défavorables .









Q3.11

Nous avons trouvé que le PLNE permettant de savoir s'il existe une affectation où tout étudiant a un de ses k premiers choix était :

Il y a une première contrainte c'est que tous les étudiants soient affectés à un parcours. Pour ce faire nous mettons en place un système qui pour un Xi correspond au lien entre un parcours et un étudiant. Ainsi nous pouvons dire que pour tout Xi correspondant à un parcours la somme de ce dernier doit être égale à la capacité du parcours.

Dans notre exemple pour le parcours 0 nous mettons cette contrainte :

```
c12: x0 + x9 + x18 + x27 + x36 + x45 + x54 + x63 + x72 + x81 + x90 = 2
```

x0 correspond au lien entre l'étudiant 0 et le parcours 0, x9 le lien entre l'étudiant 1 et le parcours 0, … Nous faisons pareil pour tous les parcours.

Ensuite, nous ajoutons 2 autres contraintes qui sont que pour tous les étudiants, leur parcours affecté doit être parmis leur k premier choix.

Ainsi pour k=5 nous avons :

```
c1: x3 + x5 + x6 + x7 + x8 = 1
```

```
c21: x0 + x1 + x2 + x4 = 0
```

Pour c1 nous mettons tous les k préféré parcours de l'étudiant 0.
Pour c21 nous mettons tous les parcours ne faisant pas partie des k préféré.
Ainsi nous assurons que l'étudiant 0 est l'un de c'est parcours préféré et qu'il n'est pas affecté à un parcours ne faisant pas partie de ces k premiers choix.
La fonction objectif correspond à la somme de tous les Xi.

Q3.12

Il n'existe pas de solution pour k=3, mais il existe au moins une solution pour un k plus grand car nous avons déjà réalisé Gale-Shapley et nous avons trouvé au moins un couple parfait.

Q3.13

Nous avons trouvé que pour k>=5 il existe une solution qui satisfait tous les étudiants. Pour k=5 une solution est :
Etu0 -> 3
Etu1 -> 0
Etu2 -> 0
Etu3 -> 5
Etu4 -> 1
Etu5 -> 4
Etu6 -> 8
Etu7 -> 8
Etu8 -> 7
Etu9 -> 2
Etu10 -> 6

Q3.14

Pour la suite, nous gardons la même structure de PLNE que le premier en considérant le k=11 pour ne considérer que l'utilité. Nous devons également modifier la fonction objectif, nous devons pour chaque Xi mettre en coefficient sont utilité qui est 9-(La position du parcours dans la liste de choix de l'étudiant)+11-(La position de l'étudiant dans la liste de choix du parcours). Nous obtenons donc une utilité moyenne de 81%.
Pour l'utilité côté étudiant nous changeons la fonction objectif en mettant comme coefficients 9-(La position du parcours dans la liste de choix de l'étudiant). Nous obtenons 83% d'utilité.

Q3.15

Pour ce PLNE maximisant la somme des utilités, nous avons gardé les mêmes conditions que le PLNE décrit en premier en ajoutant 2 conditions. La première que tous les Xi tel que Xi appartiennent au parcours m, et que Xi soit l'un des k premier choix du parcours m, on dit que leur la somme de ces Xi doit être égal à la capacité du parcours m.

La deuxième condition est la suite de la première, on dit que tous les Xi qui n'appartiennent pas au k premiers choix doivent être égal 0.

Q3.16

Nous pouvons donc conclure, que dans ce cas précis les étudiants était légèrement favorisés puisque leur utilité moyenne est supérieure à celle des parcours. De plus les parcours ayant beaucoup de choix similaires cela crée des problèmes pour minimiser l'utilité minimal puisque certains étudiant se retrouve avec forcé à être mis en relation avec des parcours de dernier choix.

6. Description des jeux d'essais

Test de la matrice de préférence étudiant (test_matrice_etu) :
Ce test vérifie que la fonction matrice_etu lit correctement le fichier PrefEtu.txt et génère la matrice des préférences des étudiants.

Test de la Matrice des Préférences des Parcours (test_matrice_spe) :
Ce test vérifie que la fonction matrice_spe lit correctement le fichier PrefSpe.txt et génère la matrice des préférences des parcours.

Test des Capacités des Parcours (test_capacite) :
Ce test vérifie que la fonction capacite_spe lit correctement le fichier PrefSpe.txt et génère la liste des capacités des parcours.

Test de l'Algorithme de Gale-Shapley (test_gs) :
Ce test vérifie que les fonctions gale_shapley_ce et gale_shapley_cp produisent les affectations correctes en utilisant les matrices de préférences et les capacités.

Test des Paires Instables (test_paires_instables_m_instable) :
Ce test vérifie que la fonction paires_instables identifie correctement les paires instables dans un mariage instable.

Test des Paires Instables avec Gale-Shapley (test_paires_instables_gs) :
Ce test vérifie que les affectations produites par les algorithmes de Gale-Shapley sont stables, c'est-à-dire qu'il n'y a pas de paires instables.

Test de Génération des Matrices de Préférences (test_genere_mat) :
Ce test affiche les matrices de préférences générées aléatoirement pour 5 étudiants et 9 parcours.

Test de Génération des Capacités (test_genere_cap) :
Ce test affiche les capacités générées pour les parcours en fonction du nombre d'étudiants.

Test des Temps de Calcul (test_temps) :
Ce test mesure et affiche les temps moyens d'exécution et les nombres d'itérations des algorithmes de Gale-Shapley pour différentes valeurs de nn.