TME - Semaines 1 à 3

Yuxiang Zhang Antoine Lecomte

Partie 1 : Problème et affectation

Question 2

Nous avons différentes possibilités pour optimiser les performances de l'algorithme de Gale-Shapley en ce qui concerne les structures de données :

Nous utilisons plusieurs structures de données qui ont chacune des avantages pour les optimisations que nous cherchons à effectuer.

1 Algorithme `gale_shapley_etudiants`

1.1 Structures de données utilisées

- `deque` **(double-ended queue) pour** `etudiants_libres`
  - Permet une gestion efficace des étudiants libres avec un accès en $O(1)$ pour les ajouts/suppressions en tête.
  - Utilisé pour traiter rapidement les étudiants non encore affectés.
- `list` **pour** `prochain_parcours`
  - Tableau d'entiers permettant de suivre l'indice du prochain parcours à proposer pour chaque étudiant.
  - Accès en $O(1)$ par étudiant.
- `dict` **(dictionnaire) pour** `classement_parcours`
  - Dictionnaire imbriqué où chaque parcours associe un étudiant à un rang de préférence.
  - Accès direct en $O(1)$ pour comparer les classements.
- `dict` **(dictionnaire) pour** `affectations` **avec** `heapq` **(min-heap)**
  - Stocke les étudiants affectés à chaque parcours sous forme de file de priorité (heap binaire).
  - Permet de récupérer et de comparer rapidement l'étudiant le moins préféré ($O(\log k)$ pour une suppression/insertion).

- `list` **pour** `capacite_restante`
  - Tableau d'entiers indiquant le nombre de places restantes pour chaque parcours.
  - Accès en $O(1)$.

1.2 Pourquoi ces structures ?

- L'utilisation de `deque` optimise la gestion des étudiants libres.
- Les dictionnaires assurent un accès rapide aux classements et affectations.
- L'utilisation de `heapq` permet de gérer efficacement les parcours lorsqu'ils atteignent leur capacité limite.

Question 3

Cet algorithme suit l'approche où les étudiants proposent aux parcours.

1.3 Complexité temporelle

Analysons les différentes étapes :

Initialisations :

- Création de `etudiants_libres` $\rightarrow O(n)$
- Création de `prochain_parcours` $\rightarrow O(n)$
- Création de `classement_parcours` $\rightarrow O(nm)$
- Création de `affectations` $\rightarrow O(m)$
- Copie de `capacites` $\rightarrow O(m)$

Total pour l'initialisation : $O(nm)$

Boucle principale : Tant qu'il y a des étudiants libres :

- Chaque étudiant propose à un parcours en fonction de sa liste de préférences.
- Un étudiant peut être rejeté et re-proposer $O(n)$ fois dans le pire cas.
- Accéder à `classement_parcours[j][i]` est en $O(1)$.
- Ajout et suppression dans un tas : $O(\log k)$ où k est la capacité du parcours.

Conversion des affectations en liste triée :

- Chaque parcours trie ses k étudiants en $O(k \log k)$.

Total pour les propositions : $O(nm \log k)$ dans le pire cas.

1.4 Complexité spatiale

- `etudiants_libres` $\rightarrow O(n)$
- `prochain_parcours` $\rightarrow O(n)$
- `classement_parcours` $\rightarrow O(nm)$
- `affectations` (min-heaps) $\rightarrow O(mk)$
- `capacite_restante` $\rightarrow O(m)$

Total : $O(nm + mk)$

Question 4

2 Algorithme `gale_shapley_parcours`

2.1 Structures de données utilisées

- `heapq` **(min-heap) pour** `parcours_libres`
  - Stocke les parcours non encore remplis avec une extraction en $O(\log m)$.
  - Permet un traitement ordonné des parcours encore disponibles.
- `list` **pour** `capacite_restante`
  - Identique au premier algorithme : permet un accès rapide en $O(1)$ au nombre de places restantes.
- `list` **pour** `etudiant_affecte`
  - Tableau indiquant à quel parcours chaque étudiant est actuellement affecté (-1 s'il est encore libre).
  - Permet un accès en $O(1)$ pour vérifier si un étudiant est déjà affecté.
- `dict` **pour** `classement_etudiants`
  - Stocke les préférences de chaque étudiant sous forme de dictionnaire {parcours: rang}.
  - Accès en $O(1)$.

- **`dict` avec `deque` pour `CP` (préférences des parcours)**
  - Remplace une liste classique pour éviter un `pop(0)` coûteux ($O(n)$ en liste classique, $O(1)$ avec `deque`).
  - Utilisé pour suivre quels étudiants sont encore dans la liste de préférences des parcours.
- **`dict` avec `set` pour `affectations`**
  - Stocke les étudiants affectés à chaque parcours sous forme d'ensembles (`set`).
  - Permet une suppression en $O(1)$ contrairement à une liste ($O(n)$ en recherche et suppression).

2.2 Pourquoi ces structures ?

- L'utilisation de `heapq` optimise la gestion des parcours libres, permettant une sélection efficace.
- `deque` réduit le coût des suppressions en début de liste.
- `set` améliore l'efficacité des mises à jour des affectations par rapport à une liste classique.

3 Résumé des différences entre les deux algorithmes

Structure	`gale_shapley_etudiants`	`gale_shapley_parcours`
File d'attente	`deque` pour étudiants	`heapq` pour parcours
Classements	`dict` des parcours	`dict` des étudiants
Suivi des affectations	`heapq` (min-heap)	`set` pour accès rapide
Gestion des préférences	Liste indexée des choix	`deque` pour éviter `pop(0)`

Table 1: Comparaison des structures de données des algorithmes

Les deux algorithmes sont optimisés en fonction du point de vue (étudiants ou parcours) et utilisent des structures adaptées pour garantir un bon équilibre entre accessibilité rapide et efficacité en mémoire.

Cet algorithme suit l'approche où les parcours font les propositions aux étudiants.

3.1 Complexité temporelle

Initialisations :

- Création de `parcours_libres` $\rightarrow O(m)$

- Construction du tas (heapify) $\rightarrow O(m)$
- Copie de `capacite_restante` $\rightarrow O(m)$
- Création de `etudiant_affecte` $\rightarrow O(n)$
- Création de `classement_etudiants` $\rightarrow O(nm)$
- Conversion de `CP` en `deque` $\rightarrow O(m)$
- Création de `affectations` $\rightarrow O(m)$

Total pour l'initialisation : $O(nm)$

Boucle principale : Tant qu'il y a des parcours libres :

- Chaque parcours propose aux étudiants dans l'ordre de préférence.
- Un parcours peut proposer jusqu'à $O(n)$ étudiants avant de se stabiliser.
- Opérations sur les structures :
  - `heapq.heappop(parcours_libres)` $\rightarrow O(\log m)$
  - `popleft()` sur `deque` $\rightarrow O(1)$
  - Mise à jour de `affectations` et `capacite_restante` $\rightarrow O(1)$
  - `heapq.heappush(parcours_libres, ancien_parcours)` $\rightarrow O(\log m)$

Conversion des affectations en liste : $O(mk)$

Total pour les propositions : $O(nm \log m)$ dans le pire cas.

3.2 Complexité spatiale

- `parcours_libres` $\rightarrow O(m)$
- `capacite_restante` $\rightarrow O(m)$
- `etudiant_affecte` $\rightarrow O(n)$
- `classement_etudiants` $\rightarrow O(nm)$
- `CP` (deque) $\rightarrow O(m)$
- `affectations` (set) $\rightarrow O(mk)$

Total : $O(nm + mk)$

4 Comparaison des Algorithmes

Algorithme	Temps (pire cas)	Espace
Gale-Shapley Étudiants	$O(nm \log k)$	$O(nm + mk)$
Gale-Shapley Parcours	$O(nm \log m)$	$O(nm + mk)$

Temps :

- Gale-Shapley Étudiants est plus rapide si k est petit ($\log k \leq \log m$).
- Gale-Shapley Parcours peut être plus efficace si $m \ll n$.

Espace : Les deux algorithmes utilisent $O(nm+mk)$, donc leur empreinte mémoire est similaire.

5 Conclusion

- Si $m \ll n$, Gale-Shapley Parcours est préférable.
- Si k est petit, Gale-Shapley Étudiants est plus rapide.
- Les deux versions utilisent des structures efficaces pour minimiser les suppressions coûteuses.

À noter que côté parcours : chaque parcours a une capacité k et doit donc gérer plus de conflits que les étudiants. Un étudiant peut remplacer son affectation plusieurs fois, ce qui multiplie les rejets et réaffectations. De plus, en moyenne, un parcours effectue plus d'itérations avant que les affectations ne soient stables. Il y a donc moins de réaffectations côté étudiant en moyenne.

Question 5

Affectations finales (Côté étudiants) :

- Étudiants 3, 5 : 0
- Étudiant 4 : 1
- Étudiant 9 : 2
- Étudiant 8 : 3
- Étudiant 10 : 4
- Étudiant 0 : 5
- Étudiant 1 : 6

- Étudiant 7 : 7
- Étudiants 2, 6 : 8

Affectations finales (Côté parcours) :

- Parcours 0 : 5, 3
- Parcours 1 : 4
- Parcours 2 : 9
- Parcours 3 : 8
- Parcours 4 : 10
- Parcours 5 : 1
- Parcours 6 : 0
- Parcours 7 : 7
- Parcours 8 : 6, 2

Question 6

À l'exécution :

Vérification de la stabilité de l'affectation (étudiants) :

- Aucune paire instable trouvée. L'affectation est stable.

Vérification de la stabilité de l'affectation (parcours) :

- Aucune paire instable trouvée. L'affectation est stable.

Le test de vérification pour rechercher d'éventuelles paires instables échoue,... (pas de paires instables détectées)

Partie 2 : Evolution du temps de calcul

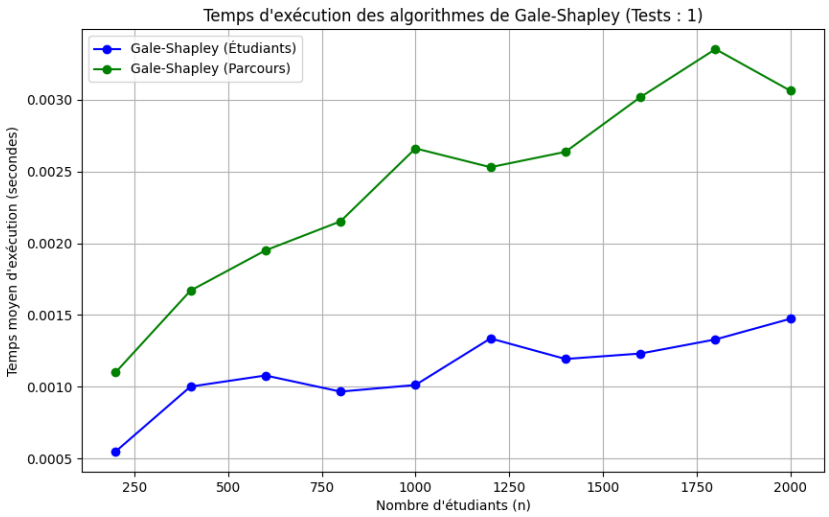
Question 9

La complexité temporelle observée semble cohérente avec l'analyse théorique des algorithmes de Gale-Shapley. L'algorithme centré sur les étudiants est plus efficace, lorsque l'on fait des moyennes de temps d'exécution (voir le graphique ci-après avec une moyenne sur 100 itérations pour 200, 400, 600, ..., 2000 étudiants).

La courbe bleue concernant Gale-Shapley côté étudiant montre une croissance linéaire modérée du temps d'exécution avec le nombre d'étudiants. Le temps moyen d'exécution reste relativement bas, même pour 2000 étudiants

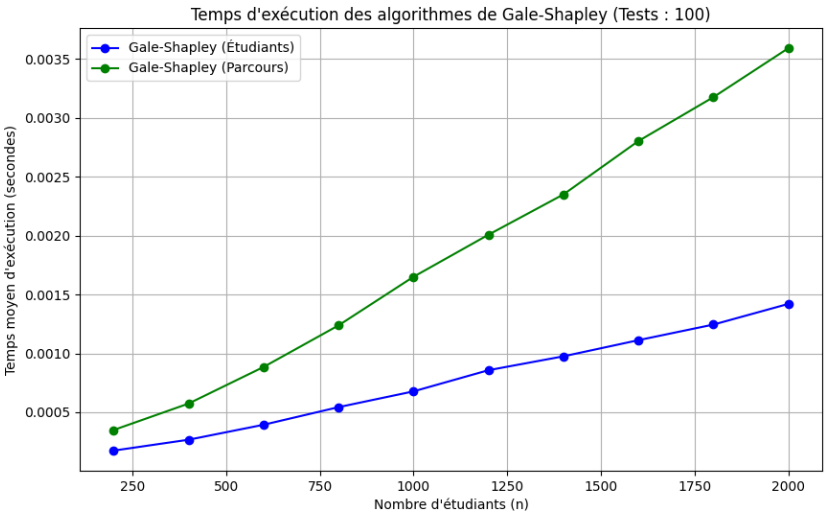
il est inférieur à 15 ms. La croissance linéaire est cohérente avec la complexité $O(nm \log k)$ et en moyenne il y a peu de réaffectations des étudiants dans cette version de l'algorithme en comparaison avec l'autre. Comme m (nombre de parcours) et k (capacité des parcours) restent constants (et les valeurs sont petites), le facteur dominant est O(n).

La courbe verte concernant Gale-Shapley côté parcours croît plus rapidement et à partir de n=1000 étudiants, la différence commence à devenir plus significative et la courbe semble également indiquer une croissance linéaire du temps d'exécution avec le nombre d'étudiants. On voit, malgré de faibles différences, que l'algorithme côté parcours devient moins performant à mesure que le nombre d'étudiants augmente car la courbe est plus inclinée, cela montre une croissance plus rapide : cohérent avec $O(nm \log m)$.



Question 10

Jusqu'à 100 tests effectués, nous obtenons des courbes ressemblant de près à des courbes linéaires, et cela en fonction du nombre d'étudiants impliqués. Les courbes obtenues sont conformes à nos attentes. Comme on peut le remarquer, lorsqu'on effectue un seul test, on voit que les temps d'exécution sont proches des temps moyens obtenus avec une centaine d'itérations, bien que cette fois-ci, on ne peut distinguer l'apparence linéaire des courbes (ce qui n'est pas surprenant, car faire une moyenne permet d'approcher un comportement auquel se rapproche chaque exécution, en terme de temps utilisé) et il suffit de relancer plusieurs fois l'exécution n°2 du menu de notre code pour s'en convaincre, car l'allure de la courbe ne sera jamais exactement



la même. Au contraire, lorsque l'on relance avec 100 tests, puisque c'est une moyenne, l'allure de la courbe est toujours similaire et cela nous aide à comprendre les complexités de nos algorithmes.

Nous pouvons noter que les temps d'exécutions pour Gale-Shapley côté étudiant vont de 0.2 ms approximativement pour 200 étudiants à 1.4 ms en moyenne pour 2000 étudiants. Pour l'algorithme côté parcours, nous obtenons environ 0.3 ms comme temps d'exécution avec 200 étudiants et jusqu'à 3.6 ms en moyenne avec 2000 étudiants. Donc, nous voyons clairement que l'algorithme de Gale-Shapley côté étudiant est plus efficace que celui côté parcours, au fur et à mesure que l'on augmente le nombre d'étudiants. De plus, le nombre m de parcours est seulement de 9, donc les deux algorithmes restent très performants, mais avec davantage de parcours considérés, nous obtiendrions des différences bien plus significatives entre les temps d'exécution des deux versions de l'algorithme, car il est susceptible d'y avoir beaucoup plus de réaffectations des étudiants côté parcours et ce qui en découle est que cela augmenterait énormément les temps d'exécution des deux algorithmes et en particulier pour l'algorithme côté parcours.

Partie 3 : Equité et PL(NE)

Question 11

Dans cette étude, nous avons utilisé les fichiers `PrefEtu.txt` et `PrefSpe.txt` pour $n = 11$ et $k = 3$. Notre objectif est de générer un fichier `.lp` correspondant au Programme Linéaire en Nombres Entiers (PLNE) et de le résoudre

à l'aide du solveur Gurobi.

6 Problème et Modélisation en PLNE

6.1 Définitions et Notations

- S : ensemble des étudiants.
- P : ensemble des parcours.
- $x_{i,j}$: variable binaire indiquant si l'étudiant i est affecté au parcours j :

$$x_{i,j} = \begin{cases} 1, & \text{si l'étudiant } i \text{ est affecté au parcours } j \\ 0, & \text{sinon} \end{cases}$$

- $U_{i,j}$: score de Borda attribué par l'étudiant i au parcours j.
- m : nombre total de parcours et k le nombre maximal de choix considérés.

6.2 Formulation PLNE

Contraintes :

1. **Chaque étudiant est affecté à un seul parcours** :

$$\sum_{j \in P} x_{i,j} = 1, \quad \forall i \in S$$

2. **Respect des capacités des parcours** :

$$\sum_{i \in S} x_{i,j} \leq C_j, \quad \forall j \in P$$

3. **Affectation dans les k premiers choix** :

$$\sum_{j \in P : U_{i,j} \geq m-k} x_{i,j} = 1, \quad \forall i \in S$$

4. **Variables binaires** :

$$x_{i,j} \in \{0, 1\}, \quad \forall i \in S, \forall j \in P$$

6.3 Génération du fichier LP avec Python

Le modèle PLNE est généré à l'aide de la fonction `generate_lp_file_k`, qui prend en entrée les préférences des étudiants et des parcours, ainsi que les capacités des parcours et le nombre maximal de choix k pour chaque étudiant.

Ce fichier LP est ensuite résolu à l'aide de Gurobi pour trouver une solution qui respecte les contraintes d'affectation, de capacité et de choix parmi les k premiers choix des étudiants.

6.4 Résolution avec Gurobi

Une fois le fichier LP généré, il peut être résolu avec Gurobi pour déterminer l'affectation optimale des étudiants aux parcours, en respectant les contraintes mentionnées.

6.5 Analyse des Résultats

Une fois la solution obtenue, nous analyserons l'affectation des étudiants et la capacité de chaque parcours. Nous examinerons également l'utilité moyenne et minimale des étudiants pour évaluer la qualité de la solution.

7 Génération du fichier `.lp`

Nous avons écrit une fonction permettant de générer un fichier `.lp` représentant notre problème d'affectation. La méthode suit les étapes suivantes :

1. Définition des variables $x_{i,j}$, représentant l'affectation de l'étudiant i au parcours j.
2. Ajout des contraintes :
   - Chaque étudiant est affecté à un seul parcours.
   - Respect des capacités des parcours.
   - Le choix des étudiants respecte les k premiers choix.
3. Maximisation de la fonction objective : maximiser les scores de Borda.

Question 12

Nous avons utilisé la fonction `generate_lp_file_k`. Après l'exécution de cette fonction, nous avons trouvé les contraintes dans le fichier `affectation_k.lp`.

Ensuite, nous avons exécuté la commande suivante :

```
gurobi_cl ResultFile=solution.sol affectation_k.lp
```

Les résultats obtenus sont les suivants :

Model is infeasible
Best objective -, best bound -, gap -
Unable to retrieve attribute 'X'

Cela signifie qu'il n'y a pas de solution pour $k = 3$.

Question 13

À l'aide de la fonction `find_min_k`, nous avons trouvé la valeur minimale de k permettant à tous les étudiants d'obtenir un de leurs k premiers choix dans les affectations pour les Masters. Echec à k=4, et affectation stable à partir de k=5, donc il est possible d'obtenir au moins une affectation stable à partir de ce kmin.

Maximisation de l'utilité minimale des étudiants

Nous cherchons à maximiser l'utilité minimale des étudiants pour un k donné. Le problème est formulé sous forme d'un Programme Linéaire en Nombres Entiers (PLNE) avec les contraintes suivantes :

1. **Affectation unique de chaque étudiant** : Chaque étudiant i doit être affecté à un seul parcours parmi ses k premiers choix. Cela se traduit par la contrainte suivante :

$$\sum_{j \in P_i^k} x_{i,j} = 1, \quad \forall i \in S$$

   où P_i^k est l'ensemble des k premiers choix de l'étudiant i, et $x_{i,j}$ est une variable binaire indiquant si l'étudiant i est affecté au parcours j.

2. **Respect des capacités des parcours** : La capacité de chaque parcours j doit être respectée, c'est-à-dire que le nombre d'étudiants affectés à ce parcours ne doit pas dépasser sa capacité. La contrainte est formulée comme suit :

$$\sum_{i \in S} x_{i,j} \leq C_j, \quad \forall j \in P$$

   où C_j est la capacité du parcours j.

3. **Maximisation de l'utilité minimale** : L'objectif est de maximiser l'utilité minimale des étudiants. La contrainte relative à l'utilité minimale est la suivante :

$$\sum_{j \in P_i^k} U_{i,j} \cdot x_{i,j} \geq U_{\min}, \quad \forall i \in S$$

où $U_{i,j}$ est le score de Borda combiné pour l'étudiant i et le parcours j, et $U_{\min}$ est la variable représentant l'utilité minimale que nous cherchons à maximiser.

4. **Objectif** : L'objectif du problème est de maximiser l'utilité minimale $U_{\min}$.

$$\text{Maximiser } U_{\min}$$

Nous avons utilisé `PULP` en utilisant la méthode `PULP_CBC_CMD`. Le résultat renvoie l'affectation optimale des étudiants aux parcours ainsi que l'utilité minimale atteinte.

Remarque : Dans la solution obtenue, chaque affectation de l'étudiant à un parcours est représentée par $x_{i,j} = 1$, indiquant que l'étudiant i est affecté au parcours j, et l'utilité minimale obtenue correspond à la valeur de $U_{\min}$.

Les résultats renvoyés sont les suivants :

Votre choix : 5
Aucune solution trouvée pour $k = 1$, tentative avec $k = 2$
Aucune solution trouvée pour $k = 2$, tentative avec $k = 3$
Aucune solution trouvée pour $k = 3$, tentative avec $k = 4$
Aucune solution trouvée pour $k = 4$, tentative avec $k = 5$
Solution trouvée pour $k = 5$
Le plus petit k pour lequel la solution est faisable est : 5

Afin de maximiser l'utilité minimale des étudiants, nous avons utilisé la fonction `maximize_min_utility`, qui, à l'aide de la fonction `score_borda_combined` (fonction renvoyant une matrice contenant l'ensemble des sommes des scores étudiants et parcours), nous a permis de déterminer l'utilité minimale la plus haute, atteinte.

L'affichage obtenu est le suivant :

Plus petit k trouvé : 5
Affectation optimale : {(0, 8): 1.0, (1, 4): 1.0, (2, 0): 1.0, (3, 0): 1.0, (4, 1): 1.0, (5, 7): 1.0, (6, 5): 1.0, (7, 2): 1.0, (8, 6): 1.0, (9, 8): 1.0, (10, 3): 1.0}
Utilité minimale atteinte : 8.0

Les valeurs `1.0` indiquent que les affectations des étudiants aux parcours ont bien été réalisées. Sinon, elles seraient à `0.0` (mais l'algorithme renvoie directement la solution trouvée, donc il n'y a pas de `0.0`). Les couples sont sous la forme (étudiant, parcours), et l'utilité minimale trouvée est de 8 pour $k = 5$ ($k_{\min}$). Cela signifie que l'étudiant le moins satisfait a une utilité minimale de 8, et tous les autres étudiants ont au moins 8 comme utilité minimale également.

Modèle de Programmation Linéaire pour Maximiser l'Efficacité Totale avec Équité

Question 14

7.1 PLNE pour maximiser la somme des utilités

Nous devons écrire un **Programme Linéaire en Nombres Entiers (PLNE)** permettant de maximiser la somme des utilités des étudiants et des parcours.

7.1.1 Modèle PLNE

Variables de décision :

$$x_{i,j} \in \{0, 1\}$$

où $x_{i,j} = 1$ si l'étudiant i est affecté au cours j, sinon 0.

Objectif : Maximiser la somme des utilités (score Borda combiné) :

$$\max \sum_{i=1}^{n} \sum_{j \in P_i} s_{i,j} \cdot x_{i,j}$$

où $s_{i,j}$ est le score Borda combiné du couple (i, j) et P_i est l'ensemble des k premiers choix de l'étudiant i.

Contraintes :

- **Chaque étudiant est affecté à un seul cours**

$$\sum_{j \in P_i} x_{i,j} = 1, \quad \forall i \in \{1, ..., n\}$$

- **Capacité des parcours respectée**

$$\sum_{i : j \in P_i} x_{i,j} \leq c_j, \quad \forall j \in \{1, ..., m\}$$

- **Les variables sont binaires**

$$x_{i,j} \in \{0, 1\}, \quad \forall i, j$$

7.1.2 Résolution avec Gurobi

Nous avons résolu le PLNE pour différentes valeurs de k :

Interprétation :

- L'utilité totale **augmente avec** k, mais **se stabilise à** $k = 7$.
- À partir de $k = 7$, **l'utilité minimale atteint 10**, garantissant une meilleure équité.

k	Utilité totale atteinte	Utilité moyenne atteinte	Utilité minimale	Valeur objective
5	131	6.55	2	129
6	144	7.2	7	144
7	153	7.65	10	153
8	153	7.65	10	153
9	153	7.65	10	153

Table 2: Résultats de l'optimisation en fonction de k

- $k = 7$ **est donc un bon choix**, car il maximise l'utilité et assure une équité entre étudiants.

Réponse finale :

- **Utilité moyenne obtenue :** 7.65
- **Utilité minimale obtenue :** 10
- **Valeur optimale de k choisie :** 7

Resultat obtenu:

Parcours	Etudiant(s)
0	[4, 5]
1	[10]
2	[7]
3	[3]
4	[9]
5	[0]
6	[8]
7	[6]
8	[1, 2]

Question 15

Nous souhaitons maximiser la somme des utilités combinées des étudiants et des parcours tout en garantissant une utilité minimale pour chaque étudiant. Le modèle de programmation linéaire est défini comme suit :

Variables de décision

- $x_{ij} = \begin{cases} 1 & \text{si l'étudiant } i \text{ est affecté au parcours } j \\ 0 & \text{sinon} \end{cases}$
- $U_{\min} \geq 0$: Utilité minimale garantie pour chaque étudiant.

Paramètres

- n : Nombre d'étudiants.
- m : Nombre de parcours.
- k : Nombre de premiers choix considérés pour chaque étudiant.
- S_{ij} : Score de Borda combiné de l'étudiant i pour le parcours j.
- C_j : Capacité maximale du parcours j.

Fonction Objectif

Nous maximisons la somme des utilités combinées des étudiants et des parcours, en ajoutant un poids pour garantir l'équité :

$$\max \left(\sum_{i=1}^{n} \sum_{j \in P_i^k} S_{ij} \cdot x_{ij} + U_{\min} \right)$$

où P_i^k est l'ensemble des k premiers choix de l'étudiant i.

Contraintes

1. **Affectation unique des étudiants :**

$$\sum_{j \in P_i^k} x_{ij} = 1, \quad \forall i \in \{1, \ldots, n\}$$

2. **Respect des capacités des parcours :**

$$\sum_{\substack{i=1 \\ j \in P_i^k}}^{n} x_{ij} \leq C_j, \quad \forall j \in \{1, \ldots, m\}$$

3. **Utilité minimale garantie pour chaque étudiant :**

$$\sum_{j \in P_i^k} S_{ij} \cdot x_{ij} \geq U_{\min}, \quad \forall i \in \{1, \ldots, n\}$$

4. **Variables binaires :**

$$x_{ij} \in \{0, 1\}, \quad \forall i \in \{1, \ldots, n\}, \quad \forall j \in P_i^k$$

En appliquant la fonction `maximize_utility_and_fairness` avec les paramètres suivants :

- **Valeur de** k : 5

Nous obtenons les résultats suivants :

- **Utilité totale atteinte** : 131.0
- **Utilité moyenne atteinte** : 6.55
- **Utilité minimale des étudiants** : 8.0

Conclusion

L'application de ce modèle d'optimisation a permis de maximiser l'efficacité totale tout en garantissant une certaine équité entre les étudiants. L'utilité moyenne atteinte est de 6.55, ce qui reflète une distribution globale satisfaisante des affectations. Cependant, l'utilité minimale des étudiants reste relativement basse à 2.0, ce qui indique qu'au moins un étudiant bénéficie d'une affectation loin de ses préférences optimales. Cela souligne la difficulté d'équilibrer parfaitement efficacité et équité dans ce type de problème d'affectation.

Comparaison des différentes solutions obtenues

Question 16

Ensemble des affectations trouvées pour chaque question :

Question	Affectations
Q3	0: [3, 5], 1: [4], 2: [9], 3: [8], 4: [10], 5: [0], 6: [1], 7: [7], 8: [2, 6]
Q4	0: [5, 3], 1: [4], 2: [9], 3: [8], 4: [10], 5: [1], 6: [0], 7: [7], 8: [6, 2]
Q13	0: [2, 3], 1: [4], 2: [7], 3: [10], 4: [1], 5: [6], 6: [8], 7: [5], 8: [0, 9]
Q14	0: [4, 5], 1: [10], 2: [7], 3: [3], 4: [9], 5: [0], 6: [8], 7: [6], 8: [1, 2]
Q15	0: [7, 10], 1: [4], 2: [6], 3: [9], 4: [5], 5: [0], 6: [1], 7: [3], 8: [2, 8]

Nous avons plusieurs solutions :

- **Gale-Shapley côté étudiants (Résultats Q3)** : C'est la solution obtenue par l'algorithme Gale-Shapley basé sur les préférences des étudiants.
- **Gale-Shapley côté parcours (Résultats Q4)** : C'est la solution obtenue par l'algorithme Gale-Shapley basé sur les préférences des parcours.
- **Solutions des questions 13, 14 et 15** : Ces solutions proviennent de l'optimisation visant à maximiser l'utilité totale tout en respectant certaines contraintes supplémentaires.

Nous allons comparer ces solutions selon les critères suivants :

- **Stabilité** (existe-t-il des paires instables ?)
- **Utilité moyenne** (la moyenne des utilités des étudiants)
- **Utilité minimale** (l'utilité minimale parmi les étudiants)

1. Stabilité

Selon la nature de l'algorithme **Gale-Shapley**, il garantit que les paires retournées sont stables. Par conséquent, les solutions obtenues dans la **Q3** et la **Q4** doivent être stables.

- **Résultats Q3 (GS côté étudiants)** : La solution est stable par définition.
- **Résultats Q4 (GS côté parcours)** : La solution est stable par définition.
- **Résultats Q13** : Instable d'après le résultat obtenu.
- **Résultats Q14** : Instable d'après le résultat obtenu.
- **Résultats Q15** : Instable d'après le résultat obtenu.

Si nous souhaitons trouver des affectations stables (contrainte), il faut donc utiliser l'algorithme de Gale-Shapley, mais néanmoins, cet algorithme (côté étudiant comme côté parcours) ne garantit pas l'équité et l'efficacité.

2. Utilité moyenne

L'utilité moyenne représente la satisfaction globale des étudiants. Une utilité moyenne plus élevée indique une plus grande satisfaction des étudiants.

Méthode

Pour chaque étudiant, on regarde où se trouve le parcours qui lui a été attribué dans sa liste de préférences.

Gale-Shapley côté étudiants

Étudiant	Parcours attribué	Liste de préférences(coté Etu)	Classement (8 = meilleur choix)
3	0	[6, 5, 7, 0, 8, 4, 3, 1, 2]	5
5	0	[0, 7, 4, 2, 8, 3, 1, 6, 5]	8
4	1	[1, 6, 7, 5, 0, 2, 4, 8, 3]	8
9	2	[2, 6, 5, 8, 3, 1, 4, 7, 0]	8
8	3	[5, 7, 6, 2, 8, 3, 0, 1, 4]	3
10	4	[6, 4, 0, 8, 3, 1, 5, 2, 7]	7
0	5	[5, 7, 6, 8, 3, 2, 0, 1, 4]	8
1	6	[6, 5, 0, 4, 7, 2, 8, 3, 1]	8
7	7	[7, 0, 4, 2, 8, 3, 1, 6, 5]	8
2	8	[4, 0, 7, 2, 8, 3, 1, 6, 5]	4
6	8	[5, 7, 6, 2, 8, 3, 0, 1, 4]	4

Étudiant	Parcours attribué	Liste de préférences(coté Spe)	Classement (10 = meilleur choi
3	0	[7, 9, 5, 4, 3, 1, 0, 10, 6, 8, 2]	6
5	0	[7, 9, 5, 4, 3, 1, 0, 10, 6, 8, 2]	8
4	1	[7, 5, 9, 4, 3, 1, 0, 10, 8, 6, 2]	7
9	2	[3, 9, 5, 4, 7, 6, 1, 0, 10, 8, 2]	9
8	3	[7, 9, 5, 4, 3, 1, 0, 6, 10, 8, 2]	1
10	4	[10, 3, 0, 4, 5, 6, 7, 8, 9, 1, 2]	10
0	5	[1, 0, 3, 4, 5, 6, 7, 2, 9, 10, 8]	9
1	6	[0, 1, 3, 4, 5, 6, 7, 2, 8, 10, 9]	9
7	7	[7, 6, 9, 5, 4, 3, 1, 0, 10, 8, 2]	10
2	8	[1, 0, 3, 4, 5, 6, 7, 2, 9, 10, 8]	3
6	8	[1, 0, 3, 4, 5, 6, 7, 2, 9, 10, 8]	5

Satisfaction moyenne des étudiants (côté étudiants) en fusionnant les classement de deux tableau au-dessus

$$\frac{11+16+15+17+4+17+17+17+18+7+9}{20} = 7.4$$

Gale-Shapley côté parcours

Étudiant	Parcours attribué	Liste de préférences(coté Spe)	Classement (10 = meilleur choix)
5	0	[7, 9, 5, 4, 3, 1, 0, 10, 6, 8, 2]	8
3	0	[7, 9, 5, 4, 3, 1, 0, 10, 6, 8, 2]	6
4	1	[7, 5, 9, 4, 3, 1, 0, 10, 8, 6, 2]	7
9	2	[3, 9, 5, 4, 7, 6, 1, 0, 10, 8, 2]	9
8	3	[7, 9, 5, 4, 3, 1, 0, 6, 10, 8, 2]	1
10	4	[10, 3, 0, 4, 5, 6, 7, 8, 9, 1, 2]	10
1	5	[1, 0, 3, 4, 5, 6, 7, 2, 9, 10, 8]	10
0	6	[0, 1, 3, 4, 5, 6, 7, 2, 8, 10, 9]	10
7	7	[7, 6, 9, 5, 4, 3, 1, 0, 10, 8, 2]	10
6	8	[1, 0, 3, 4, 5, 6, 7, 2, 9, 10, 8]	5
2	8	[1, 0, 3, 4, 5, 6, 7, 2, 9, 10, 8]	3

Étudiant	Parcours attribué	Liste de préférences(coté Etu)	Classement (8 = meilleur choix
5	0	[0, 7, 4, 2, 8, 3, 1, 6, 5]	8
3	0	[6, 5, 7, 0, 8, 4, 3, 1, 2]	5
4	1	[1, 6, 7, 5, 0, 2, 4, 8, 3]	8
9	2	[2, 6, 5, 8, 3, 1, 4, 7, 0]	8
8	3	[5, 7, 6, 2, 8, 3, 0, 1, 4]	3
10	4	[6, 4, 0, 8, 3, 1, 5, 2, 7]	7
1	5	[6, 5, 0, 4, 7, 2, 8, 3, 1]	7
0	6	[5, 7, 6, 8, 3, 2, 0, 1, 4]	6
7	7	[7, 0, 4, 2, 8, 3, 1, 6, 5]	8
6	8	[5, 7, 6, 2, 8, 3, 0, 1, 4]	4
2	8	[4, 0, 7, 2, 8, 3, 1, 6, 5]	4

Satisfaction moyenne des étudiants (côté parcours) en fusionnant les classement de deux tableau au-dessus

$$\frac{16 + 11 + 15 + 17 + 4 + 17 + 17 + 16 + 18 + 9 + 7}{20} = 7.35$$

- **Résultats Q3** : Utilité moyenne 7.4
- **Résultats Q4** : Utilité moyenne 7.35
- **Résultats Q13** : Utilité moyenne 6.55
- **Résultats Q14** : Utilité moyenne 7.65
- **Résultats Q15** : Utilité moyenne 6.45

La solution **Q14** a la plus haute utilité moyenne, indiquant que les étudiants sont plus satisfaits, comparé aux autres solutions. Donc la meilleure solution est k=7 pour maximiser l'utilité moyenne.

3. Utilité minimale

L'utilité minimale est la valeur de l'utilité la plus faible parmi tous les étudiants. Idéalement, on souhaite que cette valeur soit aussi élevée que possible afin de garantir l'équité.

- **Résultats Q3** : Utilité minimale 4.0
- **Résultats Q4** : Utilité minimale 4.0
- **Résultats Q13** : Utilité minimale 2.0
- **Résultats Q14** : Utilité minimale 10.0
- **Résultats Q15** : Utilité minimale 8.0

La solution, **Q14** a une utilité minimale de 10, ce qui est bien plus équitable que les solutions des autres questions, qui ont des utilités minimales inférieures ou égales à 8.

Conclusion

- **Stabilité** : Les solutions aux **Q3** et **Q4** sont stables (garanties par Gale-Shapley), tandis que les solutions **Q13**, **Q14**, et **Q15** sont plus optimisées, mais peuvent contenir des paires instables.
- **Utilité moyenne** : La solution **Q14** a la plus grande utilité moyenne, ce qui signifie que les étudiants sont globalement plus satisfaits.
- **Utilité minimale** : La solution **Q14** a la plus grande utilité minimale, ce qui est plus équitable que les résultats trouvés aux autres questions, qui ont des utilités minimales inférieures ou égales à 8, signifiant que l'évaluation de l'affectation pour l'étudiant ayant le plus petit score pour celle-ci, selon la position dans ses préférences au parcours auquel il est associé et de la position de l'étudiant dans les préférences de ce parcours, est meilleure que l'évalutation du plus petit score obtenu pour chacune des autres questions. Autrement dit, la pire affectation de toutes obtient un score plus élevé avec les affectations trouvées à la **Q14** qu'avec les autres questions.

En résumé, la solution obtenue à la **Q14** est la plus intéressante, bien qu'il puisse y avoir quelques paires instables, elle améliore l'utilité des étudiants et est plus équitable que les affectations données par les algorithmes de Gale-Shapley. Donc, hormis la contrainte de stabilité, la solution trouvée à cette question est meilleure en tout point.