Université de Montpellier

Rapport du projet
Conception et implantation d'un système d'aide à la décision

El Houiti Chakib
Kezzoul Massili

30 novembre 2021

Introduction

Objectif du projet

L'objectif principal du projet est de réaliser une analyse critique de l'algorithme du mariage stable. Dans un premier temps l'objectif est d'implémenter l'algorithme de Gale et Shapley (mariage stable), pour l'affectation des étudiants aux instituts, ensuite, de proposer une méthode de satisfaction pour les deux côtés et de tester cet algorithme sur plusieurs jeux de données. Finalement, c'est de proposer une représentation compacte des préférences.

Environnement de développement

Le projet a été développé sur notre propre environnement de travail. On a utilisé le langage Python, pour l'implémentation des différentes fonctionnalités, en utilisant plusieurs bibliothèques propres à Python.

Structure du projet

Pour une meilleure compréhension de l'environnement du projet, voici ci-dessous différentes informations sur les différents fichiers et répertoires du projet :

src/ Répertoire contenant les fichiers sources du projet.

- **generate.py** Fichier pour la génération automatique de jeux de données.
- **algorithme.py** Fichier implémentant les différents algorithmes, notamment celui de Gale et Shapley.
- **graphviz.py** Fichier de visualisation.
- **main.py** Fichier contenant le programme principale du projet.

data/ Répertoire contenant les différents jeux de données de différentes tailles.

output/ Répertoire contenant toutes les sorties du programme.

README.md Fichier expliquant la manière d'utiliser le programme (initialisation, exécution). Referez-vous à la section *Utilisation* de ce dernier pour plus d'informations.

1 Modélisation et implémentation

1.1 Programme de génération de préférences aléatoires

La première partie du travail est la génération de préférences aléatoires pour les étudiants ainsi que les instituts. L'objectif ici est de générer un fichier contenant ces préférences qui pourra ensuite être interprété par un programme. Nous avons choisi de représenter les préférences par un fichier *JSON*[1]. En effet, ce format est très expressif et facile à manipuler.

1. JavaScript Object Notation est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple.

```
{
  "students": {
      "E1": ["I3", "I1", "I2"],
      "E2": ["I1", "I2", "I3"],
      "E3": ["I3", "I2", "I1"]
  },
  "institutions": {
      "I1": {
          "capacities": 1,
          "preferences": ["E1", "E2", "E3"]
      },
      "I2": {
          "capacities": 1,
          "preferences": ["E1", "E3", "E2"]
      },
      "I3": {
          "capacities": 1,
          "preferences": ["E3", "E2", "E1"]
      }
  }
}
```

Listing 1 – "Exemple d'un fichier de préférences"

Pour cela, nous avons défini une fonction qui, en lui donnant en paramètre : N le nombre d'étudiants et K le nombre d'instituts, génère le fichier *JSON* ci-dessus. Ce fichier représente les préférences des étudiants et celles des instituts. On attribut à chaque étudiant une liste de K instituts générer aléatoirement et classées par ordre de préférence. La même chose est faite pour chaque institut. Mais cette fois, pour chaque instituts, on lui attribut aléatoirement en plus une capacité d'accueil. La capacité d'accueil est généré de sorte que la somme de toutes les capacités soit égale à N le nombre d'étudiants.

Ceci a été implémenté dans le fichier *generate.py* indépendamment du reste des programmes. Les jeux de données ainsi générer sont ensuite mis dans le répertoire *data/*.

1.2 Implémentation de l'algorithme du mariage stable

La seconde partie du projet est l'implémentation d'un algorithme de mariage stable. Pour cela nous avons adapté deux implémentations de l'algorithme de *Gale & Shapley* à nos jeux de données. L'une donnant la priorité aux étudiants et la seconde aux instituts [2]. Dans ces implémentation, nous supposant qu'il y assez de place dans les instituts pour tout les étudiants. Nous supposant aussi que la taille de la liste des préférences des étudiants (Resp. les instituts) est égale à K le nombre d'instituts (Resp. N le nombre d'étudiants). Cette condition est nécessaire afin d'assurer qu'un mariage stable existe (Voir le Théorème de Hall).

Ensuite, nous obtenons un programme principal (*main.py*), utilisable en ligne de commande, qui affiche (voir Figure 1) le temps d'exécution des deux algorithmes ainsi que la satisfaction des étudiants et des instituts, point qu'on vas aborder un peu plus loin dans ce rapport.

De plus, ce programme exporte dans deux fichiers sous le format *JSON*[3] les résultats des affectations (priorité aux étudiants et priorité aux instituts).

2. Les deux fonctions ont été définis dans le fichier *algorithme.py*
3. Le format *CSV* est aussi possible mais pour la visualisation en graphe c'est le format *JSON* qui est utilisé

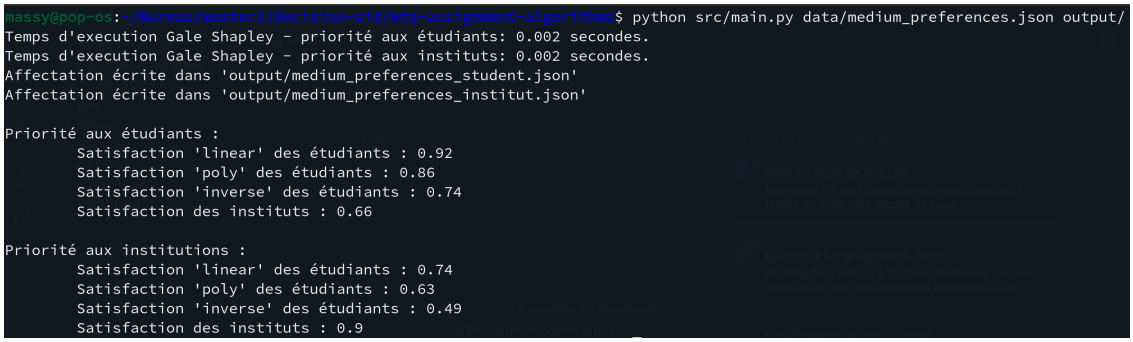

FIGURE 1 – Affichage produit par le programme principal

```
0  {
1     "I1": ["E1", "E2", "E11"],
2     "I2": ["E5", "E12", "E17", "E18", "E19"],
3     "I3": ["E6", "E9", "E16", "E10"],
4     "I4": ["E4", "E7", "E8", "E13", "E20"],
5     "I5": ["E3", "E14", "E15"]
6  }
```

Listing 2 – "Fichier des affectations priorité au étudiants"

1.3 Interface de visualisation

Pour pouvoir mieux visualiser les affectations des étudiants aux instituts, on a pensé à une structure de graphes. Chaque institut et chaque étudiant sont représentés par un nœud étiqueté par leurs noms. Chaque affectation est représentée par une arête qui lie un étudiant à son institut.

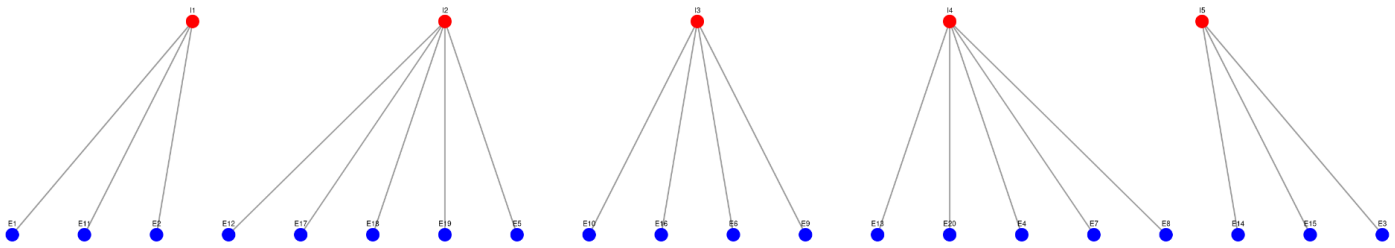

FIGURE 2 – Visualisation en graphe

On obtient cette visualisation [4] en utilisant le programme *graphviz.py* à qui en passe en argument un fichier *JSON* produits précedement. Ce programme utilise la bibliothèque *Dash*, qui permet de projeter des graphes interactifs via un petit serveur web en local.

4. Les points rouges représentent les instituts et les points bleus représentent les étudiants

1.4 Méthodes de satisfaction

La satisfaction de chaque côté est nécessaire, pour évaluer nos algorithmes et les classés. Les problèmes de satisfaction apparaissent toujours dans les systèmes d'aid à la décision ou plus précisément dans les systèmes d'affectations. Dans notre projet, on s'est concentré sur la satisfaction des étudiants, qui est un problème très fréquent dans la vie réelle.

Satisfaction des étudiants

Il existe plusieurs manières pour calculer la satisfaction des étudiants, on a choisi des méthodes qui sont significatives. Ces méthodes ont de même des points forts et des points faibles. Toutes les méthodes sont faites, d'une façon de donner une note à chaque étudiant, selon son affectation par rapport à sa liste de préférences. Une moyenne de ces notes permet de mesurer la satisfaction globale de tous les étudiants. La note de chaque étudiant est comprise entre 0 et 1, c.à.d. si un étudiant a eu son premier choix, il aura une note de 1 et s'il a eu son dernier choix, il aura une satisfaction égale à 0. Les notes des autres choix sont calculées avec des fonctions mathématiques du genre $y = f(i)$, tel que, y est la note de satisfaction et i est la position de l'affectation de l'étudiant dans sa liste de préférences.

Linéaire en premier, une fonction linéaire, une façon très simple de calculer les satisfactions. Son principe est de donner une note à un choix i d'un étudiant qui est inférieure à la note du choix *i-1*. Cette note diminue de 1 vers 0 d'une façon constante (comme la montre le graphe ci-dessous) et cela selon le nombre de choix **k** des étudiants. Donc, par exemple si **k** = 10, on aura une liste de valeurs = (1, 0.9, 0.8 ,0.7, ..., 0). Cette méthode de calculer est efficace pour un petit jeu de données, mais si on a un grand jeu de données, par exemple 100000 étudiant et 100 instituts, on a constaté que la satisfaction globale approche de 1, car les cinq premiers choix auront des notes élevées égales à (1, 0.99, 0.98, 0.97, 0.96), alors que dans un jeu de données pareil, 90% des étudiants ont eu un de leurs trois premiers choix et que le pire étudiant a eu son 11ème choix avec une note de 0.9, tout cela affecte la moyenne globale des satisfactions et du fait quelle soit proche de 1. Aussi, vu que notre fonction diminue d'une façon linéaire, la différence entre la note du 1er et du 2ème choix est égale à la différence entre la note du 66ème et du 67ème choix. Ce qui nous a mener à penser à d'autres fonctions.

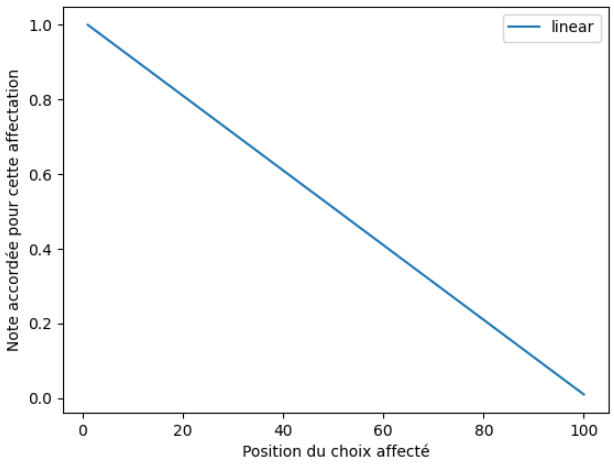

FIGURE 3 – Fonction linéaire

Polynomiale une fonction qui diminue d'une façon polynomiale, c.à.d. la différence entre chaque deux choix décroît en allant du premier au dernier choix. Ce qui limite le problème de la fonction linéaire. Par exemple, la différence entre la note du 1er et du 2ème choix est supérieure à la différence entre la note du 66ème et du 67ème choix. Ce qui donne une forme d'importance aux premiers choix et une importance moindre aux derniers. Dans un cas pratique, un étudiant qui aura son 50ème choix ou son 60ème choix reste déçu dans les deux cas. Par contre, un étudiant qui aura son 1er choix ou son 10ème, voit sa satisfaction très affecter. Le graphe ci-dessous montre la décroissance non-linéaire de la note d'une affectation.

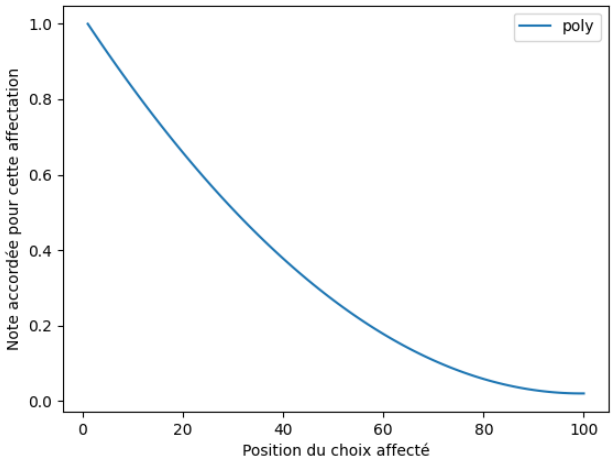

FIGURE 4 – Fonction polynomiale

Inverse une fonction inverse du genre $f(i) = \frac{1}{i}$. Cette fonction permet de donner une plus grande importance aux premiers choix, dans ce cas-là le 2ème choix aura une note de 0.5, qui est une différence énorme entre le 1er et le 2ème choix. On a pensé à cette fonction pour donner une satisfaction globale qui accorde plus d'importance au 1er choix, Donc plus la moyenne est élevée, plus on saura qu'un grand nombre d'étudiants ont eu leurs 1er choix. Le graphe ci-dessous montre la décroissance de la fonction inverse.

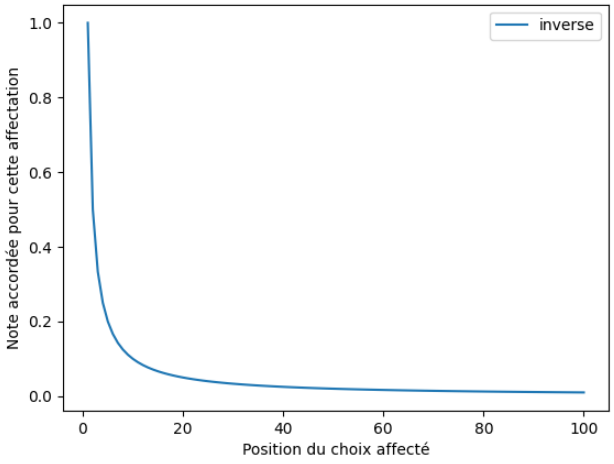

FIGURE 5 – Fonction inverse

Nous pouvons remarquer, sur la figure ci-dessous, la différence entre ces différentes fonctions.

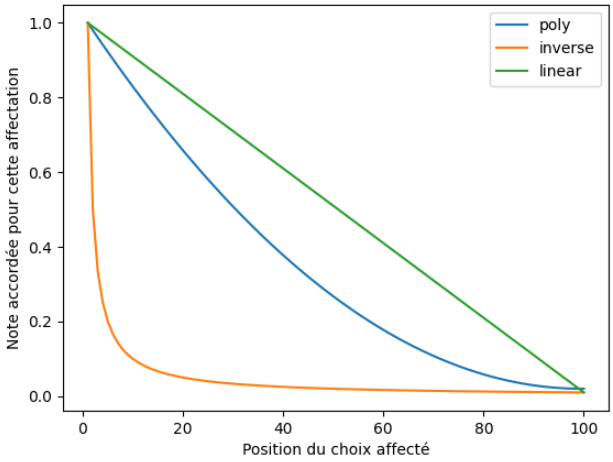

FIGURE 6 – Comparaison des fonctions

Satisfaction des instituts

Contrairement à la satisfaction des étudiants, celle des instituts doit respecter une contrainte supplémentaire, qui est due au fait qu'on affecte à chaque institut plusieurs étudiants selon sa capacité Q. Pour palier à cette contrainte et avoir une note de satisfaction significative, Nous attribuons une note maximale de 1 pour un institut donné, si et seulement si les étudiants affectés à cet institut sont les Q **premiers** choix de ce dernier. Inversement, une note de 0 est attribuée si et seulement si les étudiants affectés à cet institut sont les Q **derniers** choix de ses préférences. Toutes les affectations qui sont au milieu suivent une fonction qui diminue linéairement. Une moyenne est calculée pour avoir une satisfaction globale de tous les instituts.

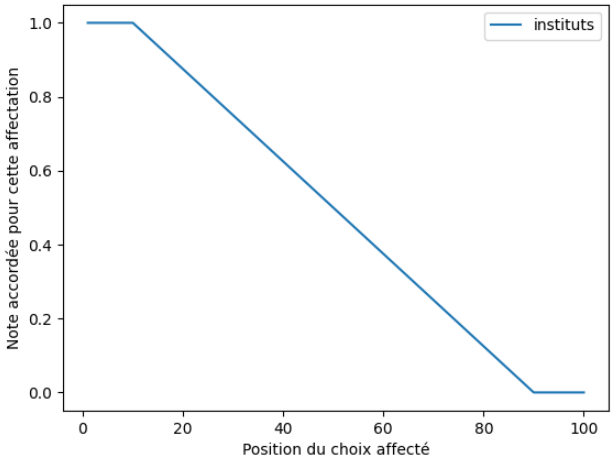

FIGURE 7 – Fonction priorité aux instituts pour $Q = 10$

2 Extension du système

Dans cette dernière partie du projet, l'objectif est de proposer une représentation compacte des préférences. On a choisi de se référer à la vie réelle, où les étudiants ont un nombre précis de choix et non pas le nombre d'instituts. Cela pose un problème pour l'algorithme du mariage stable, donc si on exécute bêtement l'algorithme de *Gale & Shapley*, on n'aura pas une affectation stable entre les étudiants et les instituts, on sortira avec un résultat où un certain nombre d'étudiants se retrouveront sans instituts. Pour remédier à ce problème on a choisi deux méthodes, pour affecter les étudiants aux instituts, ces deux dernières modifient l'algorithme de *Gale & Shapley* pour avoir un résultat.

2.1 Random assignement

Random assignement ou affectation aléatoire, cette méthode consiste à affecter temporairement et aléatoirement un étudiant à un institut *Ii* qui a toujours de la place, si cet étudiant *Ei* n'a plus de choix possible dans sa liste de préférences (il a été refusé pour tout ces choix). De cette manière, on s'assure que chaque étudiant soit affecté. Si un autre étudiant *Ej* a dans ces choix l'institut *Ii*, on préfère *Ej* à *Ei*, même si ce dernier est mieux classé que *Ej*.

2.2 Les sans instituts

Contrairement à la première méthode, on traite les étudiants qui n'ont plus de choix en dernier. c.à.d, on réalise l'affectation pour les étudiants ayant toujours un choix. Ensuite, tous les étudiants qui n'ont plus de choix seront affectés en dernier à l'institut auquel ils sont le mieux classés parmi les instituts restants qui ont toujours de la place.

3 Conclusion

3.1 Utilisation du programme

Génération des jeux de données La première étape est de générer les jeux de données. Pour cela, le script *generate.py* est utilisé. Il prend en paramètre N le nombre d'étudiants, K le nombre d'instituts et le nom du fichier à créer.

```
0 python3 src/generate.py <N> <K> <path/to/file.json>
```

Listing 3 – "Commande qui génére un jeu de données"

Exécution de l'algorithme du mariage stable Ensuite, on passe un jeu de données au programme principal *main.py*. Ce dernier prend en paramètre le fichier de préférence ainsi qu'un répertoire où écrire les affectations.

```
0 python3 src/main.py <preferences.json> <output/dir/>
```

Listing 4 – "Commande qui exécute le programme principal"

Cette commande génére deux fichiers en résultat.
— *preference_students.json* qui représente les affectations avec priorité aux étudiants
— *preference_institut.json* qui représente les affectations avec priorité aux instituts

Visualisation du graphe Enfin, ce dernier résultat est passé en argument au programme *graphviz.py* qui lance un serveur en local. Un lien est fourni pour accéder à l'interface de visualisation.

```
0 python3 src/graphviz.py <filename.json>
```

Listing 5 – "Commande qui lance la visualisation"

3.2 Perspectives

Dans l'objectif d'étendre notre système, nous avons fait des recherches sur l'état de l'art actuel concernant les représentations compactes. Nous avons notamment trouvé une étude très intéressante de *Guillaume Haeringer & Vincent Iehlé* qui aborde dans leur article une solution à ce problème. Ils proposent notamment de représenter les préférences des étudiants par une liste des instituts acceptables. Ils utilisent le théorème de *Hall* afin de vérifier l'existence d'un mariage stable.

Cette étude est accessible via ce lien : Two-sided matching with one-sided preferences.