Projet : Algorithme du mariage stable

Master 2 IASD - Aide à la Décision

Étudiant:
MARIF Amin et BOULAC Nizar

Encadrant:
Mme KACI Souhila

26 novembre 2025

Résumé

Ce projet explore les défis algorithmiques de l'appariement bilatéral (matching), un mécanisme central dans les systèmes d'affectation modernes comme Parcoursup. L'objectif est d'analyser le comportement de l'algorithme de Gale-Shapley (Acceptation Différée) au-delà de sa garantie théorique de stabilité. À travers une implémentation en Python, nous avons confronté l'algorithme à des scénarios réalistes en introduisant des préférences corrélées (stratification des vœux) et des contraintes de capacité. Nous avons volontairement écarté l'algorithme de Boston, jugé non pertinent pour cette étude car instable (inéquitable) et manipulable comme présenté dans le cours. Nos simulations révèlent une

sensibilité critique aux conditions du marché : en situation de forte tension ou de corrélation élevée, l'avantage théorique du côté "proposant" disparaît provoquant une convergence vers une solution stable unique. Au-delà de l'analyse de la satisfaction (regret, rang moyen), ce travail aborde la problématique du passage à l'échelle. Nous proposons une extension intégrant des représentations compactes (CP-Nets et Logique de Pénalité), permettant de gérer l'explosion combinatoire des préférences dans des instances de grande taille sans sacrifier la pertinence des affectations.

Table des matières

Chapitre 1

Introduction

1.1 Contexte : Les enjeux de l'appariement bilatéral

Aujourd'hui, les mécanismes d'affectation sont devenus incontournables dans la gestion des politiques publiques. Que ce soit pour l'entrée au lycée, pour les études supérieures avec Parcoursup, ou pour l'affectation des internes en médecine, on demande à ces systèmes de relever un défi énorme : réussir à transformer des milliers de vœux individuels, souvent contradictoires, en une solution collective cohérente.

La grosse différence avec un marché économique classique, c'est qu'ici, ce n'est pas le prix qui régule les échanges, mais les préférences. On ne cherche pas à équilibrer l'offre et la demande avec de l'argent, mais à faire correspondre du mieux possible les envies des candidats avec les critères de sélection des établissements. Si le système est mal conçu, les conséquences peuvent être graves : cela crée de l'instabilité, un fort sentiment d'injustice, et ça pousse même les utilisateurs à essayer de contourner les règles pour s'en sortir.

1.2 Problématique

Quand on conçoit ce type de système, on se retrouve vite coincé entre plusieurs critères qui sont souvent difficiles à concilier :

— **La Stabilité :** C'est la base. On veut s'assurer qu'aucun étudiant ne se fasse passer devant par quelqu'un de moins bien classé que lui pour une même formation. C'est ce qu'on appelle éviter l'envie justifiée.

— **L'Efficacité :** Est-ce que le système permet de contenter le plus de monde possible sans désavantager personne ?

— **La Sincérité ou non-manipulabilité :** Est-ce que le système encourage les candidats à dire la vérité ? Ils ne devraient pas avoir besoin de mentir ou de faire des calculs stratégiques pour obtenir ce qu'ils veulent.

Du coup, tout l'enjeu de ce projet, c'est de voir comment l'algorithme de Gale-Shapley réagit face à ces défis. On sait que c'est la référence théorique pour la stabilité, mais on veut voir ce qu'il donne vraiment quand on le met sous pression, notamment quand il y a un gros déséquilibre entre l'offre et la demande ou quand on doit traiter énormément de données.

1.3 Objectifs et Démarche

Pour ce travail, nous avons choisi une approche qui mélange théorie et pratique. Voici nos trois principaux objectifs :

1. **Préférences et Implémentation de l'algorithme :** Nous analyserons en détail la mécanique de l'algorithme de Gale-Shapley et ses propriétés de stabilité. Cette étape comprend l'implémentation en Python des deux variantes (Étudiant-proposant vs Établissement-proposant) ainsi que la conception de modèles de génération de préférences (aléatoires et corrélées) pour simuler des comportements réalistes en amont des tests.
2. **Tester en conditions réelles :** La théorie ne couvre pas entièrement la satisfaction des individus. Nous procéderons donc à des simulations dans des scénarios complexes (comme lorsque les places sont épuisées ou que tout le monde sollicite la même chose) afin de déterminer les véritables limites du système.
3. **Gérer le passage à l'échelle :** Quand il y a trop de choix, c'est impossible pour un utilisateur de tout classer un par un. On propose donc une solution technique (les CP-Nets) pour simplifier l'expression des préférences sans noyer l'utilisateur.

Chapitre 2

Modélisation et Génération des Données

2.1 Formalisation du problème

Pour modéliser ce problème, on reprend le modèle classique du "mariage stable", qu'on adapte au contexte des admissions universitaires. Concrètement, on a deux groupes distincts :

— Les étudiants : l'ensemble $I = \{i_1, i_2, ..., i_n\}$.

— Les établissements (ou écoles) : l'ensemble $S = \{s_1, s_2, ..., s_m\}$.

Chaque établissement s_j a un nombre de places limité, c'est sa capacité $q_j \geq 1$. Le fonctionnement est assez intuitif : chaque étudiant fait sa liste de vœux en classant les écoles qu'il préfère. De l'autre côté, les écoles classent aussi les candidats, généralement selon leurs notes ou leur dossier.

L'objectif, c'est d'obtenir une **affectation stable**. En gros, on veut éviter une situation où un étudiant et un établissement auraient tous les deux intérêt à "tromper" le système pour se mettre ensemble. C'est ce qu'on appelle une paire bloquante (i, s). Pour que ce soit stable, il ne faut pas qu'on puisse trouver un étudiant i et une école s tels que :

1. L'étudiant i préfère l'école s à celle qu'il a obtenue.
2. L'école s est aussi d'accord : elle préfère recruter i plutôt que l'un des étudiants qu'elle a déjà acceptés (ou alors, elle a encore des chaises vides).

Si une telle paire n'existe pas, alors l'affectation est solide.

2.2 Structures de données

Concernant l'aspect technique, on a opté pour le langage Python en utilisant la programmation orientée objet. C'est l'option la plus pratique pour gérer proprement les différents états des agents (savoir s'ils sont libres, placés, ou si l'établissement est rempli à capacité).Voici la manière dont on a organisé nos classes :

Classe Etudiant : Elle stocke l'ID de l'étudiant, son nom, sa liste de vœux et son statut actuel (pour suivre la progression de son affectation).

Classe Etablissement : En plus de ses propres critères de sélection, elle gère sa capacité d'accueil (q_j) et la liste des étudiants qu'elle a acceptés temporairement. Surtout, elle dispose de fonctions pour vérifier si elle est pleine et pour identifier

l'étudiant le "moins bon" parmi ceux qu'elle a déjà pris. C'est indispensable pour gérer l'étape de rejet qui est le mécanisme central de Gale-Shapley.

2.3 Protocoles de génération des préférences

Pour tester la solidité et les performances de l'algorithme, on a développé deux méthodes pour générer les préférences. L'idée est de se rapprocher de la réalité, car on sait bien que les choix des étudiants ne sont jamais faits totalement au hasard.

2.3.1 Mode Aléatoire (Uniforme)

Pour ce premier scénario, on a fait simple : les préférences sont générées de manière totalement aléatoire (cf permutation uniforme). En clair, chaque agent classe les autres au hasard et toutes les combinaisons ont la même probabilité d'arriver.

C'est utile pour s'assurer que l'algorithme fonctionne correctement et pour confirmer la théorie, mais on sait que cela ne représente pas fidèlement la réalité. En pratique, les décisions ne sont pas prises de manière aléatoire : certains établissements ont une réputation nettement supérieure à d'autres et attirent la majorité des étudiants et certains étudiants sont préférés par les établissements.

2.3.2 Mode Corrélé (Stratification)

Pour simuler un contexte plus réaliste, nous avons implémenté un générateur de préférences corrélées.. Nous associons à chaque établissement s un score de qualité intrinsèque $Q_s \in [0, 1]$ (ex : classement académique comme le classement Shangai) et à chaque étudiant i un niveau académique $N_i \in [0, 1]$ (ex : moyenne générale).

Construction des vœux des étudiants Lorsqu'un étudiant i construit sa liste de vœux, il classe les écoles selon une valeur perçue, notée $V_{i,s}$, qui est une combinaison linéaire de la réputation de l'école et d'une préférence personnelle :

$$V_{i,s} = \underbrace{\alpha \cdot Q_s}_{\text{Composante Commune}} + \underbrace{(1-\alpha) \cdot \epsilon_{i,s}}_{\text{Composante Personnelle}} \tag{2.1}$$

Où :
- Q_s est la qualité objective de l'école (identique pour tous).
- $\epsilon_{i,s} \sim \mathcal{U}(0, 1)$ est un bruit aléatoire spécifique à l'étudiant i pour l'école s (facteur subjectif : localisation, campus, etc.).
- $\alpha \in [0, 1]$ est le facteur de corrélation.

Analyse du facteur α Ce paramètre agit comme un curseur entre l'objectivité et la subjectivité :
- **Si $\alpha \to 1$ (Forte Corrélation) :** La composante commune domine. Tous les étudiants ont quasiment le même classement des écoles. Cela engendre une forte compétition pour les établissements ayant un Q_s élevé.
- **Si $\alpha \to 0$ (Indépendance) :** La composante personnelle domine. On retrouve donc un comportement proche du mode aléatoire.

Note : De manière symétrique, les établissements classent les étudiants en utilisant une formule analogue basée sur le niveau académique N_i de l'étudiant et un bruit spécifique à l'établissement.

Chapitre 3

Algorithme d'Acceptation Différée

3.1 Principe général (Gale-Shapley)

Le cœur de notre système d'affectation repose sur l'algorithme de Gale et Shapley (1962), référence de la théorie du mariage stable. Contrairement aux algorithmes dits "immédiats" (comme l'algorithme de Boston) qui assignent définitivement les places dès le premier tour, l'algorithme de Gale-Shapley utilise un principe d'**acceptation différée**.

Le mécanisme garantit qu'aucune affectation n'est définitive tant que l'algorithme n'est pas terminé. À chaque étape, un agent "proposant" fait une offre à son meilleur choix restant. L'agent "recevant" accepte l'offre temporairement, mais conserve le droit de la rompre plus tard s'il reçoit une proposition plus intéressante.

L'algorithme s'arrête lorsqu'il n'y a plus de rejet possible. La propriété fondamentale de cet algorithme est la **stabilité** : il ne peut pas y avoir de couple (étudiant, université) qui se préfèrent mutuellement à leur situation finale.

3.2 Implémentation : Étudiants proposants

Dans cette variante, ce sont les étudiants qui effectuent les démarches. Cette version est théoriquement optimale pour les étudiants (proposants). Nous utilisons une file d'attente (`etudiants_non_affectes`) et un index de progression pour chaque étudiant afin de ne jamais proposer deux fois au même établissement.

Algorithme 1 Gale-Shapley : Étudiants Proposants

Entrée : E (Ensemble des étudiants), S (Ensemble des établissements)
Sortie : μ (Affectation stable)

Initialisation :
Réinitialiser les affectations courantes
$File \leftarrow$ Copie de la liste E
$\forall e \in E, index[e] \leftarrow 0$
while $File$ n'est pas vide **do**
    $e \leftarrow$ défiler($File$)
    if $index[e] \geq$ taille($liste\_voeux[e]$) **then**
        Continuer ▷ L'étudiant a épuisé tous ses vœux
    end if
    $s \leftarrow liste\_voeux[e][index[e]]$
    $index[e] \leftarrow index[e] + 1$
    if s n'est pas plein **then**
        Affecter e à s temporairement
    else
        $pire \leftarrow$ étudiant le moins prioritaire actuellement dans s
        $rang\_nouveau \leftarrow$ rang de e dans $liste\_voeux[s]$
        $rang\_pire \leftarrow$ rang de $pire$ dans $liste\_voeux[s]$
        if $rang\_nouveau < rang\_pire$ **then** ▷ Le candidat est meilleur
            Retirer $pire$ de s
            Affecter e à s
            Enfiler($File$, $pire$) ▷ L'étudiant rejeté redevient libre
        else
            Enfiler($File$, e) ▷ L'étudiant proposant est rejeté
        end if
    end if
end while

3.3 Implémentation : Universités proposantes

Nous avons également implémenté la variante symétrique où les établissements proposent. Cette version tend à favoriser les établissements. La logique diffère légèrement : un établissement peut être remis dans la file d'attente s'il se fait "plaquer" par un étudiant (c'est-à-dire si un étudiant rompt son engagement pour une meilleure offre), libérant ainsi une place.

Algorithme 2 Gale-Shapley : Établissements Proposants

Entrée : E (Ensemble des étudiants), S (Ensemble des établissements)
Sortie : μ (Affectation stable)

Initialisation :
Réinitialiser les affectations courantes
$File \leftarrow$ Copie de la liste S
$\forall s \in S, index[s] \leftarrow 0$
while $File$ n'est pas vide **do**
    $s \leftarrow$ défiler($File$)
    if s est plein **ou** $index[s] \geq$ taille($liste\_voeux[s]$) **then**
        Continuer
    end if
    $e \leftarrow liste\_voeux[s][index[s]]$
    $index[s] \leftarrow index[s] + 1$
    Enfiler($File$, s) ▷ On remet s pour compléter ses places restantes
    if e est libre **then**
        Affecter e à s
    else
        $s_{actuel} \leftarrow$ établissement actuel de e
        $rang\_actuel \leftarrow$ rang de s_{actuel} dans $liste\_voeux[e]$
        $rang\_nouveau \leftarrow$ rang de s dans $liste\_voeux[e]$
        if $rang\_nouveau < rang\_actuel$ **then** ▷ L'étudiant préfère la nouvelle offre
            Retirer e de s_{actuel}
            if s_{actuel} n'est pas dans $File$ **then**
                Enfiler($File$, s_{actuel}) ▷ L'ancien établissement redevient actif
            end if
            Affecter e à s
        else
            Passer ▷ L'étudiant rejette l'offre
        end if
    end if
end while

Chapitre 4

Méthodologie d'évaluation et Métriques

4.1 Mesure de la satisfaction (Efficacité et Équité)

Pour analyser les résultats de nos simulations, nous ne pouvons pas nous contenter de regarder si l'algorithme termine. Il faut quantifier la qualité de l'affectation pour les étudiants et les établissements. Nous avons donc implémenté plusieurs indicateurs dans le fichier `metriques.py`, permettant de juger la performance sous plusieurs angles.

Le Rang Moyen : C'est la moyenne des rangs obtenus par les étudiants affectés. Soit E_{aff} l'ensemble des étudiants ayant obtenu une affectation $\mu(e)$. Le rang moyen est défini par :

$$\bar{R} = \frac{1}{|E_{aff}|} \sum_{e \in E_{aff}} \text{rang}(e, \mu(e))$$

où $\text{rang}(e, s)$ est la position de l'établissement s dans la liste de préférences de l'étudiant e (1 étant le premier vœu). C'est l'indicateur de base pour mesurer la satisfaction globale : plus il est proche de 1, meilleure est l'affectation collectivement.

Le Regret Maximal : Il correspond au pire classement obtenu par un étudiant dans l'affectation finale ($\max_{e \in E_{aff}} \text{rang}(e, \mu(e))$). C'est une mesure de sécurité importante : un rang moyen de 2.0 peut cacher le fait qu'un étudiant a obtenu son $50^{\text{ème}}$ vœu, ce qui pourrait s'avérer problématique en pratique.

L'Écart-Type : Nous calculons la dispersion des rangs autour de la moyenne. Cela nous sert d'indicateur d'inégalité : un écart-type élevé signifie qu'il y a de grandes disparités de traitement entre les étudiants (des "gagnants" et des "perdants"), alors qu'un écart-type faible indique une satisfaction homogène.

Le Taux d'Affectation : Dans les cas où il y a plus d'étudiants que de places (tension), le premier critère de satisfaction est simplement d'obtenir une place. Nous comptons donc systématiquement le nombre d'étudiants non affectés.

Enfin, nous traçons la distribution complète des rangs (histogrammes) pour visualiser si les affectations sont concentrées sur les premiers vœux.

4.2 Vérification de la Stabilité

La propriété théorique principale de l'algorithme de Gale-Shapley est la stabilité. Pour nous assurer que notre implémentation est correcte (et pour vérifier qu'il n'y a pas de bugs

dans les cas limites), nous avons codé une fonction pour vérifier la stabilité des solutions.

Le principe est de rechercher s'il existe des "paires bloquantes" dans le résultat final. Une paire (Étudiant e, Établissement s) est instable si :

1. L'étudiant e préfère s à son affectation actuelle (ou n'est pas affecté).
2. L'établissement s a une place libre OU préfère e à l'un de ses admis actuels.

Voici l'algorithme de vérification implémenté :

Algorithme 3 Vérification de la Stabilité (Détection des Paires Instables)

function CompterPairesInstables(E, μ)
    $Nb_{Instables} \leftarrow 0$
    for chaque étudiant $e \in E$ **do**
        if e n'est pas affecté **then**
            Continuer
        end if
        $s_{actuel} \leftarrow$ établissement affecté à e
        $rang_{actuel} \leftarrow$ rang de s_{actuel} dans les vœux de e
                ▷ On teste uniquement les écoles que e préfère à son affectation
        $Ecoles\_Preferes \leftarrow liste\_voeux[e][0 : rang_{actuel}]$
        for chaque s_{vise} dans $Ecoles\_Preferes$ **do**
            if s_{vise} n'est pas plein **then**
                $Nb_{Instables} \leftarrow Nb_{Instables} + 1$
            else
                $pire \leftarrow$ étudiant le moins prioritaire dans s_{vise}
                if s_{vise} préfère e à $pire$ **then**
                    $Nb_{Instables} \leftarrow Nb_{Instables} + 1$
                end if
            end if
        end for
    end for
    return $Nb_{Instables}$
end function

Si cet algorithme retourne une valeur supérieure à 0 sur une exécution de Gale-Shapley, cela signifierait que la stabilité n'est pas atteinte, ce qui n'est evidemment pas la réalité théorique...

Chapitre 5

Expérimentation et Analyse des Résultats

Dans cette partie, on confronte notre implémentation de Gale-Shapley à la réalité des statistiques. L'idée, ce n'est pas juste de vérifier que le code marche (ça, on le sait), mais surtout de voir comment l'algorithme réagit quand on fait varier la structure du "marché" : quand il y a de la tension, de la sélectivité ou que les vœux se ressemblent tous.

On a découpé l'analyse en deux temps : d'abord une comparaison sur cinq scénarios bien précis, puis une analyse plus globale sur la structure et le passage à l'échelle.

5.1 Analyse Comparative des Scénarios ($N = 50$)

Pour bien isoler les variables qui jouent un rôle, on a fait tourner l'algorithme sur cinq configurations différentes avec une base de 50 étudiants.

À chaque fois, on compare systématiquement les résultats pour voir ce que ça change selon qui a l'initiative (le rôle de "proposant") : est-ce que ce sont les étudiants ou les universités ?

5.1.1 Synthèse des Métriques

Le Tableau 5.1 rassemble les indicateurs de performance moyens obtenus.

Scénario	Proposant	Rang Moy. Étudiant	Regret Max	Écart-type	Non-Affectés	Rang Moy. Univ
Cas 1 : Bijection	Étudiant	**3.52**	11	2.53	0	12.70
($N = M = 50$)	Univ	9.88	40	8.77	0	**5.06**
Cas 2 : Tension	Étudiant	10.80	20	6.64	30	1.10
($N = 50, M = 20$)	Univ	10.80	20	6.64	30	1.10
Cas 3 : Sélectifs	Étudiant	4.92	10	2.90	24	2.58
($N = 50, M \approx 24$)	Univ	4.92	10	2.90	24	2.58
Cas 4 : Abondance	Étudiant	1.06	2	0.24	0	21.08
($N = 50, M \approx 70$)	Univ	1.06	2	0.24	0	21.08
Cas 5 : Corrélation	Étudiant	8.80	20	6.16	30	5.05
($N = 50, \rho = 0.9$)	Univ	8.80	20	6.16	30	5.05

TABLE 5.1 – Tableau récapitulatif des métriques de satisfaction et de stabilité

5.1.2 Le Cas Théorique : L'Equilibre (Cas 1)

Le Cas 1 représente le modèle standard théorique : un marché équilibré ($N = M$) avec des préférences aléatoires uniformes.

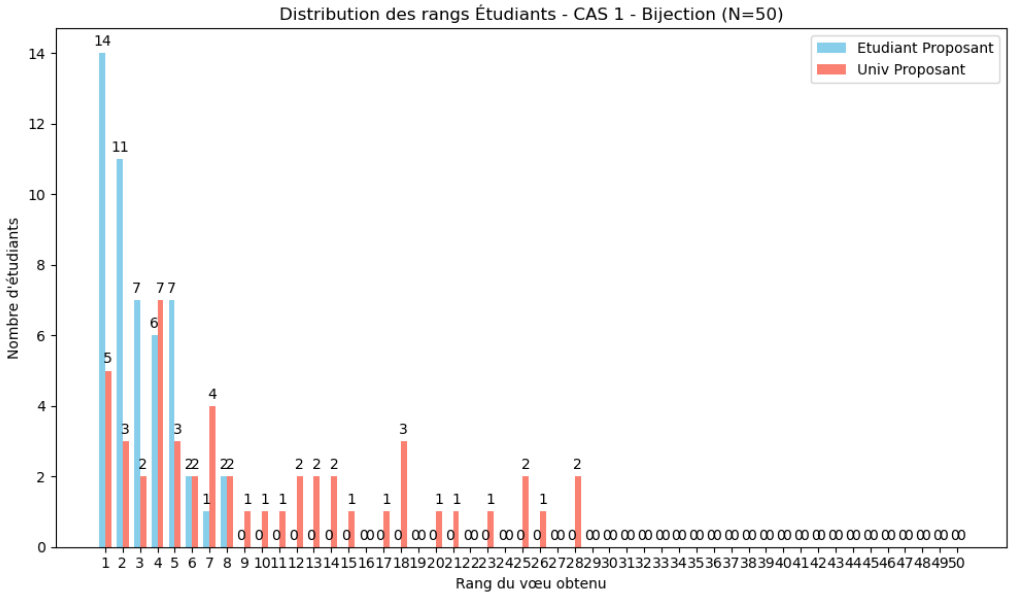


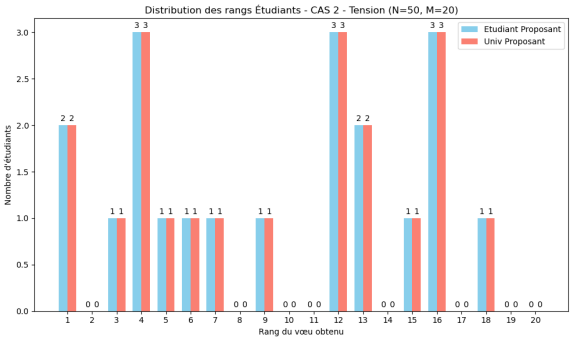
FIGURE 5.1 – Distribution des rangs dans le Cas 1 : L'asymétrie Proposant/Disposant

Analyse : On voit tout de suite un gros écart entre les deux exécutions. Quand les étudiants ont l'initiative, ils s'en sortent très bien avec un rang moyen de 3.52 et un regret max limité à 11. Par contre, dès que c'est l'université qui propose, la satisfaction des étudiants s'effondre : le rang moyen tombe à 9.88 et le regret grimpe jusqu'à 40.

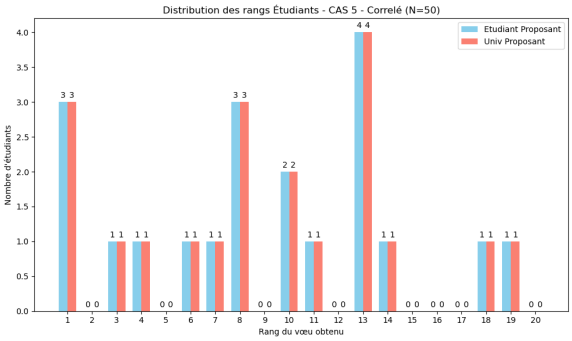
Interprétation : Ces chiffres valident concrètement le théorème de Gale-Shapley sur l'optimalité du proposant. Dans un marché dense comme celui-ci, il existe plein de solutions stables possibles (ce qu'on appelle le "treillis" des solutions). Le fait de choisir qui propose permet de se placer soit tout en haut, soit tout en bas de ce treillis.

5.1.3 Phénomène d'Effondrement du Noyau (Cas 2, 3 et 5)

Contrairement au Cas 1, les Cas 2, 3 et 5 présentent une particularité remarquable : les résultats sont strictement identiques quel que soit le proposant ($\mu_{Etu} = \mu_{Univ}$).



(a) Cas 2 : Forte Tension



(b) Cas 5 : Forte Corrélation

FIGURE 5.2 – Convergence des solutions sous contraintes

Après des recherches sur le sujet, on interprète cette convergence comme un effondrement du noyau. Concrètement, l'ensemble des solutions stables se réduit à une seule possibilité (un singleton).

— **Impact de la Pénurie (Cas 2 et 3) :** Quand 30 étudiants sur 50 se retrouvent sur le carreau (Cas 2), le rapport de force s'inverse complètement en faveur des établissements. Les étudiants n'ont plus aucune marge de manœuvre : refuser une proposition, c'est prendre le risque de finir sans affectation. Du coup, l'algorithme file tout droit vers l'unique solution imposée par les places disponibles.

— **Impact de la Corrélation (Cas 5) :** Même s'il y a des places, la forte corrélation des vœux ($\rho = 0.9$) bloque les possibilités d'échanges. Si tout le monde suit le même "palmarès" pour classer les écoles, la solution stable devient unique et inévitable. On voit d'ailleurs que la satisfaction en prend un coup (rang moyen 8.80 contre 3.52 dans le cas équilibré), tout simplement parce que la compétition se focalise sur les mêmes cibles.

5.1.4 Le Cas de l'Abondance (Cas 4)

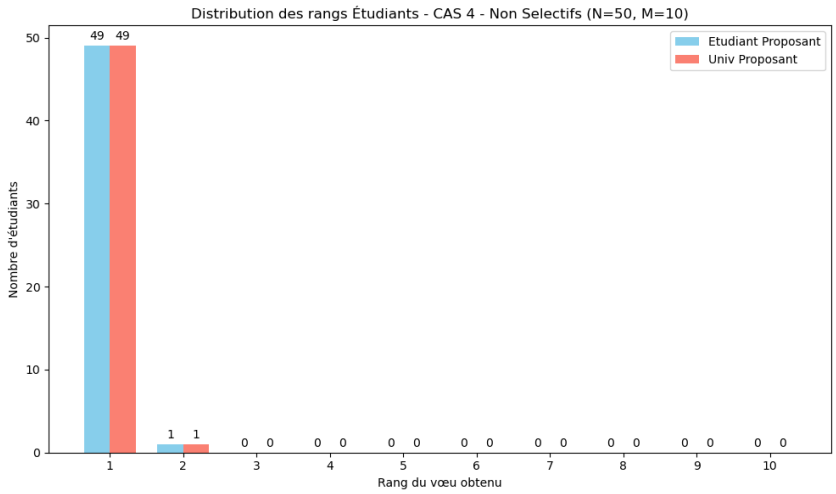


FIGURE 5.3 – Cas 4 : Satisfaction quasi-totale

Dans ce scénario où l'offre dépasse largement la demande (environ 70 places pour 50 étudiants), on atteint un rang moyen record de 1.06.

Concrètement, sur 50 étudiants, 47 obtiennent directement leur premier vœu. C'est un cas qu'on peut qualifier de trivial : comme il n'y a quasiment aucun conflit, l'algorithme est quasi-immédiat (très peu d'itérations) et la solution est unique.

5.1.5 Synthèse Croisée

La Figure 5.4 montre bien que l'avantage théorique d'être "celui qui propose" ne marche vraiment que dans des cas très précis (comme le Cas 1).

Dès que le marché se déséquilibre (tension sur les places) ou que les vœux se ressemblent trop (corrélation), le choix de l'algorithme (qui a l'initiative ?) n'a plus vraiment d'importance en pratique.

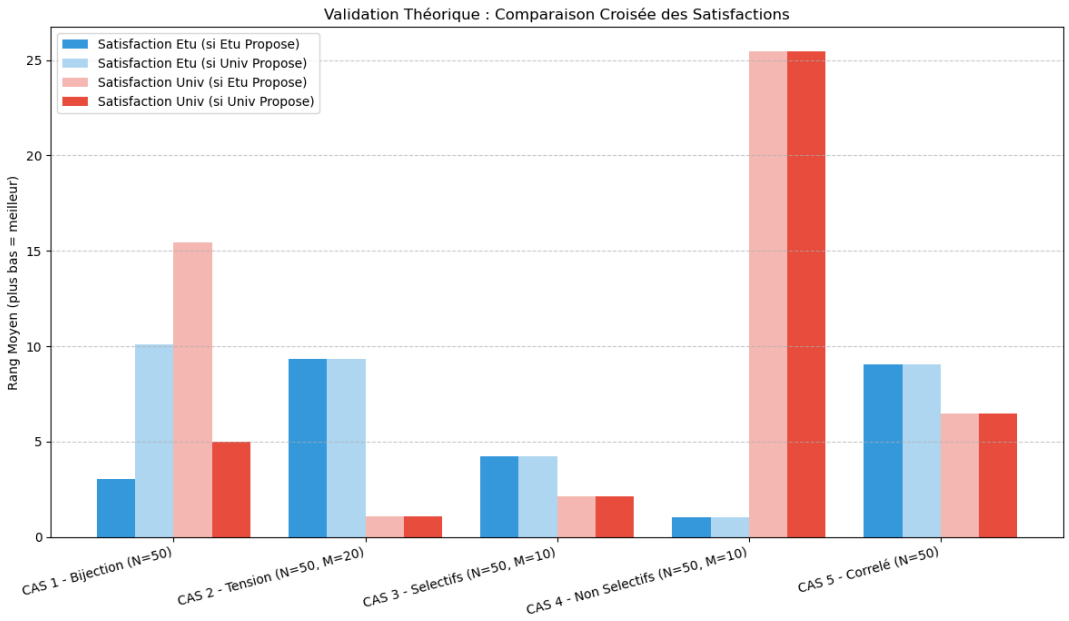


FIGURE 5.4 – Comparaison des satisfactions : L'avantage du proposant disparaît hors du Cas 1

5.2 Analyse Asymptotique et Structurelle

Au-delà des cas statiques, on a étudié le comportement dynamique de l'algorithme en faisant varier N (le nombre d'étudiants) et la structure de l'offre.

5.2.1 Complexité Temporelle et Scalabilité

Nous avons mesuré le nombre d'itérations nécessaires à la convergence pour N allant de 10 à 430 étudiants. La Figure 5.5 superpose nos résultats empiriques (lignes pleines) aux bornes théoriques (lignes pointillées).

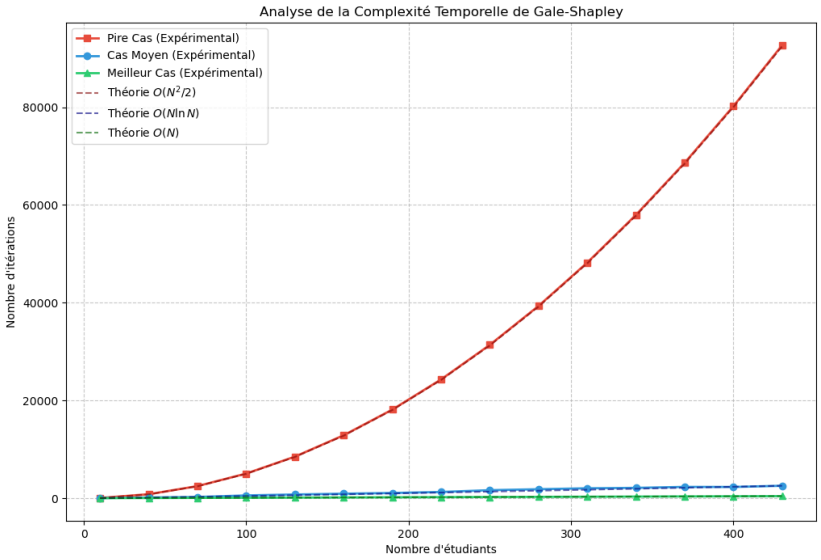


FIGURE 5.5 – Analyse de la complexité temporelle (Nombre d'itérations)

Les résultats s'expliquent par la structure des préférences générées dans nos tests :

Meilleur Cas ($O(N)$) :
En vert, ce scénario correspond à un alignement parfait des préférences (chaque étudiant i est le favori de l'école i). Chaque proposition est acceptée immédiatement et définitivement. L'algorithme termine donc en exactement N itérations.

Pire Cas ($O(N^2)$) :
En rouge, nous observons le scénario pathologique où les préférences sont strictement inversées (les étudiants classent les écoles de 0 à N, tandis que les écoles préfèrent les étudiants N à 0). Cette configuration force une cascade de refus systématique : l'école 0 rejette $N-1$ candidats, l'école 1 en rejette $N-2$, etc. Le nombre total de propositions suit alors une suite arithmétique :

$$C_{pire}(N) = \sum_{k=1}^{N} k = \frac{N(N+1)}{2} \approx \frac{N^2}{2} \tag{5.1}$$

Nos mesures confirment exactement cette prédiction : pour $N = 400$, nous obtenons 80 200 itérations (soit $\frac{400\times401}{2}$),ce qui valide la complexité quadratique.

Cas Moyen ($O(N \ln N)$) :

En bleu, les préférences sont aléatoires uniformes. Les conflits étant dispersés, le nombre de rejets diminue drastiquement. Le comportement asymptotique rejoint la complexité théorique moyenne :

$$C_{moyen} \sim N \ln N$$

Pour $N = 400$, l'algorithme converge en seulement 2 306 itérations (contre 80 200 pour le pire cas).

Conclusion : Bien que la complexité théorique soit quadratique, l'algorithme de Gale-Shapley se révèle extrêmement efficace en moyenne. Le rapport entre le pire cas et le cas moyen (facteur 35 pour $N = 400$) confirme que les scénarios pathologiques sont statistiquement improbables, justifiant l'utilisation de cet algorithme sur des volumes de données massifs type Parcoursup.

5.2.2 Transition de Phase et Ratio de Tension

On a analysé l'évolution de la satisfaction en fonction du ratio de tension $R = \frac{N}{M}$ (Demande / Offre).

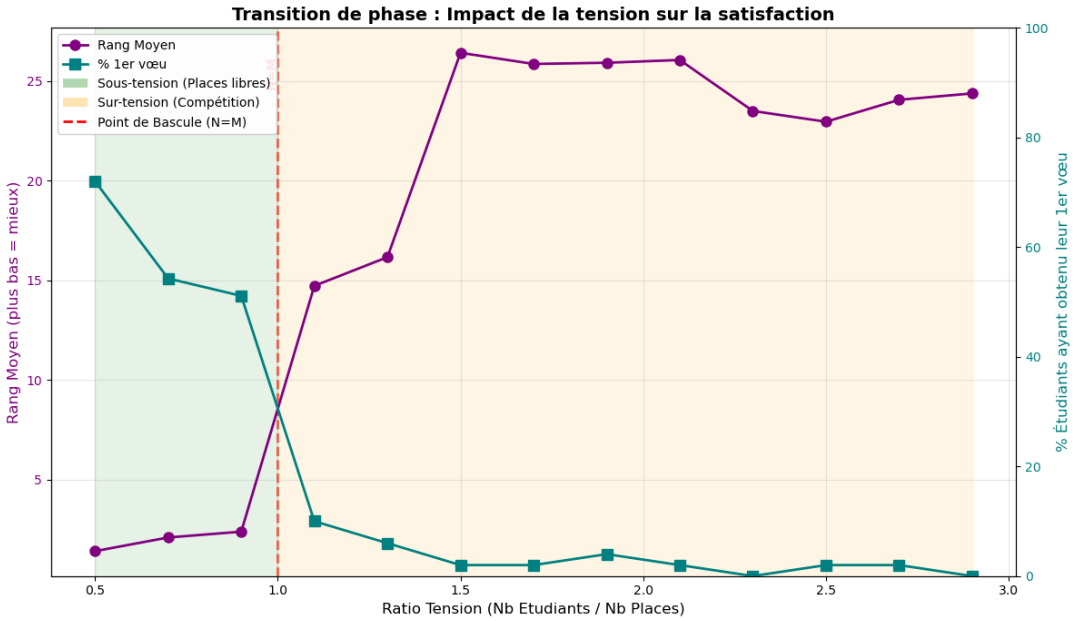


FIGURE 5.6 – Impact du ratio de tension sur la satisfaction étudiante

Sur ce graphique, on repère tout de suite une transition de phase assez brutale autour du point critique $R = 1$:

— **Zone de Sous-tension ($R < 1$) :** C'est le scénario idéal. Le rang moyen reste excellent (< 2.5) et plus de la moitié des étudiants décrochent leur premier vœu.

— **Zone de Sur-tension ($R > 1$) :** Dès qu'il y a plus d'étudiants que de places, la satisfaction dégringole de façon exponentielle. Avec un ratio de seulement 1.1 (55 étudiants pour 50 places), le taux de 1er vœu s'effondre, passant de 51% à 10%.

— **Palier de Saturation :** Passé $R = 1.5$, le rang moyen stagne autour de 26 (pour 50 places). À ce stade, l'affectation ressemble quasiment à un tirage aléatoire pour les places restantes.

5.2.3 Impact de la Granularité de l'Offre

Pour finir, en gardant la capacité constante (500 places pour 500 étudiants), on a fait varier la structure de l'offre. On est passé d'un marché totalement fragmenté (500 écoles d'une place) à un marché ultra-centralisé (une seule école de 500 places).

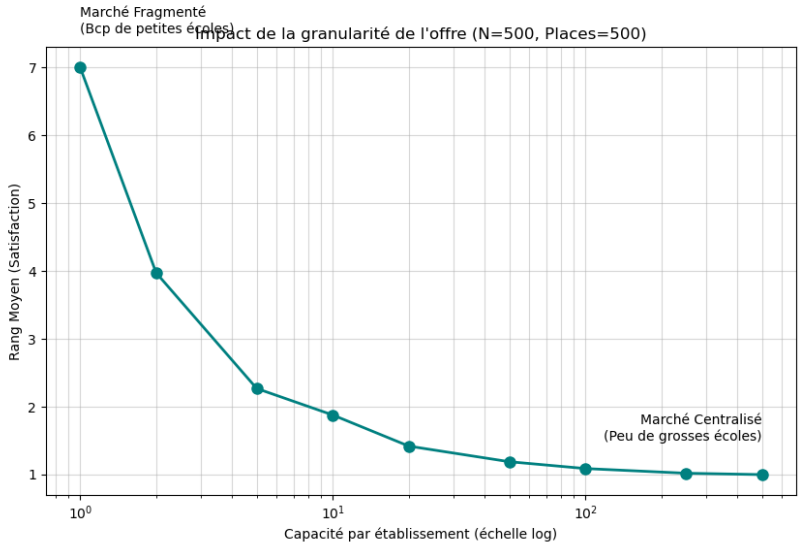


FIGURE 5.7 – Impact de la fragmentation de l'offre sur la satisfaction

Observation : On observe une amélioration logarithmique de la satisfaction à mesure que la capacité des établissements augmente.

— **Marché fragmenté (Capacité 1) :** Le rang moyen est de 7.00.

— **Marché concentré (Capacité 100) :** On descend à un excellent rang moyen de 1.09.

Interprétation : Dans un marché centralisé, les "grosses" écoles absorbent de larges parties de la demande sans déclencher les chaînes de refus en cascade qu'on subit avec des petites capacités. Ça indique qu'à quantité équivalente, une proposition concentrée est plus performante sur le plan algorithmique pour la satisfaction globale qu'une offre éparpillée. Attention, il est bon de prendre des pincettes à ce sujet et mettre cela en perspective avec notre moyen de mesurer la satisfaction...

Chapitre 6

Extension aux représentations compactes

Jusqu'ici, on a utilisé une méthode assez basique où chaque agent fournit sa liste complète et ordonnée de tous les candidats. Ça marche très bien pour $N = 30$, mais sur des cas réels comme Parcoursup (15 000 formations), c'est impossible. L'espace des possibilités est gigantesque et ça ferait exploser la mémoire.

Dans cette section, on propose donc d'abandonner les listes explicites pour passer à des modèles plus malins basés sur les attributs, en utilisant deux concepts vus en cours : la logique de pénalité (avec des scores) et les CP-nets (avec des ordres de préférence).

6.1 Changement d'approche : des ID aux attributs

Pour que ça tienne moins de place en mémoire, on ne peut plus voir les étudiants ou les écoles juste comme des numéros (ID). Il faut les décrire avec des variables.

Par exemple, un établissement s ne sera plus défini par son identifiant, mais par un vecteur d'attributs $Att(s) = (Loc, Type, Selectivite)$ où :

— $Dom(Loc) = \{Nord, Sud\}$

— $Dom(Type) = \{Ingenieur, Commerce\}$

— $Dom(Selectivite) = \{Haute, Basse\}$

L'ensemble des résultats possibles Ω devient le produit cartésien de tout ça. C'est sur ces critères précis qu'on va définir les préférences.

6.2 Approche Quantitative : La Logique de Pénalité

Première méthode pour générer des classements sans tout stocker : la **Penalty Logic**. L'idée est simple : on associe un "coût" à chaque fois qu'un critère ne nous plaît pas.

6.2.1 Le principe

Chaque étudiant i a une base de règles $\Sigma_i = \{(\phi_k, a_k)\}$. Si une école vérifie la formule ϕ_k (ce qu'on ne veut pas), elle prend a_k points de pénalité. Par exemple, un étudiant qui veut absolument le Sud et qui aimerait bien une école d'ingénieur (mais c'est moins grave) aura :

$$\Sigma_i = \{(\neg Sud, 100), (\neg Ingenieur, 20)\}$$

6.2.2 Comment on classe ?

Pour chaque école s, on calcule son score de pénalité $\rho(s)$ en additionnant les malus :

$$\rho(s) = \sum \{a_k \mid (\phi_k, a_k) \in \Sigma_i \text{ et } s \not\models \phi_k\}$$

Ensuite, Gale-Shapley utilise ce score : moins on a de points, mieux l'école est classée. S'il y a égalité, on tranche avec l'ordre alphabétique ou l'ID pour garder un ordre strict (sinon l'algorithme ne tourne pas).

6.3 Approche Qualitative : Les CP-nets

Le problème de la méthode précédente, c'est qu'il faut inventer des poids un peu au pif. Les **CP-nets** (Conditional Preference Networks) offrent une alternative graphique plus intuitive, basée sur le principe "toutes choses égales par ailleurs".

6.3.1 Structure

On modélise les préférences avec un graphe orienté. Chaque nœud est un attribut. Une flèche $Loc \rightarrow Type$ veut dire que ma préférence pour le type d'école dépend de sa localisation. Le graphe de dépendance (Figure 6.1) indique que la préférence sur le Type d'établissement est conditionnée par la Localisation géographique, et que la préférence de Sélectivité dépend du Type choisi.

Chaque nœud a une petite table de préférences (CPT). Par exemple :

— Pour Loc : Je préfère $Sud > Nord$.

— Pour $Type$ (si je suis au Sud) : $Ingenieur > Commerce$.

— Pour $Type$ (si je suis au $Nord$) : $Commerce > Ingenieur$.

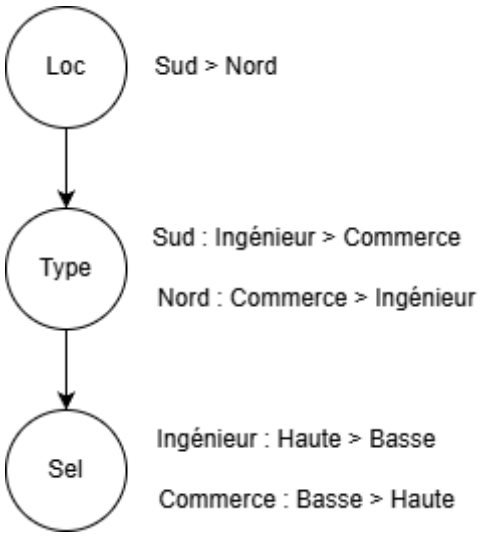


FIGURE 6.1 – Graphe du CP-net associé

6.3.2 Intégration dans l'algo

Le souci, c'est qu'un CP-net donne souvent un ordre partiel. Pour l'utiliser dans Gale-Shapley, il faut linéariser tout ça. On utilise la technique du "worsening flip" (changement

détériorant). On part de l'école idéale (ex : $Sud, Ingenieur, Haute$) et on dégrade petit à petit les critères selon le graphe pour trouver la suivante. C'est super visuel pour l'utilisateur, mais ça peut être lourd en calcul si le graphe est complexe.

6.4 Impact sur l'implémentation

Intégrer ça change radicalement notre classe Etudiant. Au lieu de stocker une liste géante (mémoire $O(M)$), l'objet stocke juste la formule logique (mémoire $O(Attributs)$).

On doit donc adapter Gale-Shapley pour qu'il fonctionne dans ce contexte :

1. L'étudiant ne génère pas sa liste au début.
2. Quand il doit faire une proposition, il demande à son "générateur" (Penalty ou CP-net) : "C'est quoi la meilleure école dispo pour moi maintenant ?".

C'est cette astuce qui permet de gérer des énormes volumes de données sans saturer la RAM.

Chapitre 7

Conclusion et Perspectives

Au terme de ce projet, on a pu mesurer à quel point le problème de l'affectation stable est complexe. Ce n'est pas juste une question de code ou de mathématiques, c'est un véritable enjeu de société qui touche à l'équité et à l'efficacité des politiques publiques.

7.1 Bilan des travaux

D'un point de vue technique, l'implémentation de l'algorithme de Gale-Shapley nous a permis de confirmer sa robustesse. Contrairement à l'algorithme de Boston, qu'on a écarté à cause de sa manipulabilité et de son manque d'équité, l'Acceptation Différée reste le seul mécanisme capable de garantir une stabilité parfaite. C'est un socle indispensable pour éviter que les étudiants ne se sentent lésés et ne cherchent à contourner le système.

Cependant, nos simulations ont montré que la théorie a ses limites. Le fameux « avantage du proposant » ne fonctionne réellement que dans un monde idéal (marché équilibré et préférences aléatoires). Dès qu'on introduit de la tension (plus d'étudiants que de places) ou de la corrélation (tout le monde veut les mêmes écoles), cet avantage disparaît. On assiste à un « effondrement du noyau » : il n'y a plus qu'une seule solution stable possible, et l'algorithme converge vers elle, peu importe qui a l'initiative.

7.2 Limites et Ouvertures

On s'est aussi rendu compte que le modèle classique, où l'on classe tout le monde un par un, ne tient pas la route face au Big Data. Sur des systèmes comme Parcoursup, l'explosion combinatoire est inévitable. C'est pour ça qu'on a exploré les représentations compactes (CP-Nets et logique de pénalité). Même si notre implémentation de cette partie reste exploratoire, elle montre clairement que l'avenir de l'aide à la décision passe par une gestion plus intelligente des préférences, basée sur des attributs plutôt que sur des listes interminables.

7.3 Le mot de la fin

Pour finir, ce projet nous a appris une leçon importante : un algorithme, aussi performant soit-il, ne peut pas faire de miracles. Gale-Shapley garantit que l'affectation sera stable et juste, mais il ne peut pas créer des places qui n'existent pas. Quand la tension

est trop forte (comme dans notre Cas 2), la frustration des usagers est mathématiquement inévitable. Le rôle de l'informaticien est donc de fournir l'outil le plus transparent et équitable possible pour gérer cette pénurie.