HAI902I
Aide à la Décision

Projet : Algorithmes du mariage stable

AUTEUR : Bilal BELLI

Université de Montpellier
Faculté des sciences

2024 - 2025

Table des matières

1 Introduction

L'aide à la décision est un domaine multidisciplinaire qui vise à assister les individus ou les organisations dans le choix d'options optimales parmi un ensemble de solutions possibles. En combinant les apports des mathématiques, de l'informatique et des sciences humaines, elle repose sur la modélisation des préférences, l'évaluation des alternatives et l'optimisation des choix. Cette discipline est au cœur de nombreuses applications pratiques, telles que la gestion des ressources, l'allocation des tâches, ou encore la planification stratégique.

Au fil des années, plusieurs classes d'algorithmes ont été développées pour répondre à des problématiques variées en aide à la décision. Les algorithmes d'optimisation mathématique, comme la programmation linéaire ou quadratique, sont utilisés lorsque les objectifs et contraintes sont formalisés sous forme mathématique. Les approches heuristiques, telles que les algorithmes génétiques ou les colonies de fourmis, se montrent efficaces pour résoudre des problèmes complexes où les solutions exactes sont difficiles à obtenir. De plus, les algorithmes d'apprentissage machine sont de plus en plus intégrés pour anticiper les préférences ou améliorer la prise de décision en temps réel.

Dans ce cadre, les algorithmes d'appariement, tels que l'algorithme du mariage stable, jouent un rôle crucial dans la résolution de problèmes où il s'agit de former des paires entre deux ensembles, en respectant des critères de stabilité et d'équité. Introduit par Gale et Shapley en 1962, l'algorithme du mariage stable est une solution élégante à cette problématique, garantissant qu'aucun couple ne préférerait se former en dehors de l'appariement proposé. Ce type d'algorithme trouve des applications pratiques dans des contextes tels que le recrutement, la répartition des ressources ou encore les systèmes de recommandation.

Ce projet s'inscrit dans cette perspective en utilisant l'algorithme du mariage stable pour analyser et résoudre un problème spécifique d'aide à la décision. L'objectif est d'explorer les mécanismes de cet algorithme, de proposer des métriques de satisfaction, d'étudier les impacts des priorités, et d'adapter l'approche à des contextes où les préférences sont représentées de manière compacte ou incomplète. Par cette démarche, il s'agit de démontrer la pertinence et la flexibilité de cet algorithme dans des scénarios complexes et réalistes.

2 Contexte

Le problème du mariage stable, introduit par Gale et Shapley, consiste à établir des appariements stables entre deux ensembles distincts, comme des candidats et des établissements, sur la base de leurs préférences respectives. Un appariement est dit stable lorsqu'il n'existe aucun couple formé en dehors de l'appariement proposé qui préfère être ensemble plutôt que de rester dans leurs appariements actuels.

Ce problème trouve de nombreuses applications pratiques, notamment dans les systèmes de sélection, tels que les admissions scolaires ou les recrutements. Toutefois, les variations contextuelles, comme la manière de représenter les préférences, leur incomplétude ou la présence d'égalités, posent des défis supplémentaires. Ces scénarios nécessitent souvent des ajustements ou des extensions à l'algorithme classique pour garantir des solutions adaptées à des cas plus complexes tout en respectant la notion fondamentale de stabilité.

3 Problématique

Comment adapter et enrichir l'algorithme du mariage stable pour répondre aux exigences d'un contexte où :

- Les préférences des candidats et des établissements peuvent être représentées de manière complexe ou incomplète ?
- Il est nécessaire de garantir un équilibre entre la satisfaction des deux parties tout en respectant les priorités ?
- Des extensions doivent être apportées pour traiter des cas pratiques spécifiques ou des variantes du problème initial ?

4 Objectifs

Les objectifs à atteindre dans ce travail sont les suivants :

1. Définir une méthode de mesure de satisfaction pour les candidats et les établissements, afin d'évaluer objectivement la qualité des appariements générés.
2. Étudier l'impact des priorités accordées aux candidats ou aux établissements sur les appariements et analyser la sensibilité de la mesure de satisfaction à ce choix.
3. Proposer une extension de l'algorithme pour intégrer des préférences exprimées de manière compacte, adaptées à des cas complexes ou partiellement définis.
4. Explorer une variante avancée de l'algorithme qui prend en compte des préférences incomplètes ou avec égalité, en détaillant son fonctionnement et en illustrant son application sur un exemple concret.

5 Problème du mariage stable

5.1 Aperçu

Le problème du mariage stable se pose dans un cadre où deux ensembles distincts, généralement appelés ***hommes*** et ***femmes***, doivent être appariés selon leurs préférences individuelles. Chaque individu a une liste ordonnée des membres de l'autre ensemble, reflétant ses préférences. L'objectif est de trouver un appariement stable, c'est-à-dire une configuration où il n'existe aucune paire d'individus préférant être appariés entre eux plutôt qu'avec leurs partenaires attribués.

Dans sa version classique, plusieurs hypothèses simplificatrices sont posées :

1. Listes de préférences complètes : Chaque individu a une préférence pour tous les membres de l'autre ensemble.
2. Préférences strictes : Les préférences sont ordonnées sans ex æquo (pas de ***ties***).
3. Équilibre entre les groupes : Les ensembles sont de taille égale.

Cependant, ces hypothèses ne reflètent pas toujours les situations réelles, où :

1. Les préférences peuvent être incomplètes, certains membres n'étant pas considérés comme des partenaires potentiels.
2. Des préférences ex æquo (*égalités*) sont courantes, notamment lorsqu'un individu trouve plusieurs options également acceptables.
3. Les tailles des ensembles peuvent varier, introduisant des contraintes supplémentaires.

5.2 Sommaire de l'article « Stable Marriage Problems with Ties and Incomplete Preferences »

L'article explore une variante du problème classique des mariages stables, appelée ***Stable Marriage with Ties and Incomplete Preferences*** (SMTI). Dans ce modèle, les listes de préférences des hommes et des femmes (Ou établissements et candidats) peuvent inclure des ex æquo et être incomplètes, ce qui complique la recherche de solutions stables.

Les chercheurs examinent trois objectifs d'optimisation pour SMTI :

1. **Maximisation de la cardinalité :** Maximiser le nombre de couples stables.
2. **Égalité des sexes :** Équilibrer les préférences entre hommes et femmes.
3. **Optimisation égalitaire :** Minimiser la somme des rangs des préférences pour maximiser la satisfaction globale.

Ils ont comparé plusieurs approches algorithmiques :

1. **Answer Set Programming (ASP) :** Formulation déclarative basée sur des contraintes logiques.
2. **Integer Linear Programming (ILP) :** Modélisation par équations linéaires.
3. **Constraint Programming (CP) :** Exploitation des contraintes pour réduire l'espace de recherche.

4. **Recherche locale :** Utilisant des algorithmes génétiques et stochastiques pour des solutions approximatives.

Les expérimentations montrent que :

— **ASP et ILP** sont efficaces pour les variantes d'optimisation égalitaire et de cardinalité, mais rencontrent des limites pour l'égalité des sexes.

— Les algorithmes locaux offrent des performances compétitives pour des instances moins complexes, mais perdent en efficacité avec des préférences très incomplètes ou avec de nombreux ex æquo.

— ASP excelle dans la gestion de contraintes complexes grâce à sa flexibilité, tandis qu'ILP est adapté aux grandes instances nécessitant des solutions optimisées.

Cette étude met en lumière la complémentarité entre approches déclaratives et heuristiques pour résoudre des variantes difficiles du problème des mariages stables. Elle ouvre également des perspectives pour des approches hybrides combinant plusieurs paradigmes.

5.3 Algorithme de Gale-Shapley

Parmi les algorithmes dédiés au problème des mariages stables, celui de Gale-Shapley est particulièrement reconnu pour son élégance. La figure 5.1 en présente une visualisation schématique.

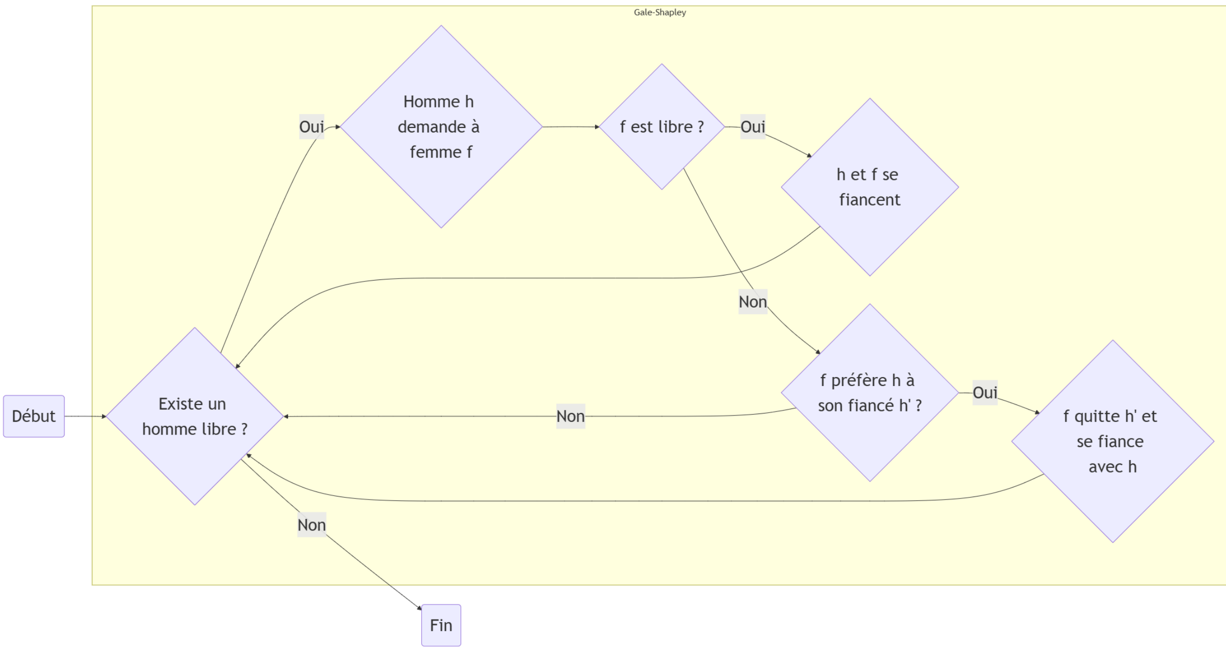


Figure 5.1 – Diagramme montrant le flux de l'algorithme de Gale-Shapley.

L'algorithme peut alors s'exprimer de la manière suivante :

Algorithm 1: Algorithme de Gale-Shapley pour le mariage stable

Data: M : ensemble d'hommes, W : ensemble de femmes, P_m, P_w : listes préférences
Result: S : ensemble de couples fiancés stables

$S \leftarrow \emptyset$
$F \leftarrow M$ `// Ensemble des hommes libres`
while $F \neq \emptyset$ **do**
    Choisir $m \in F$
    $w \leftarrow$ première femme dans P_m que m n'a pas encore demandée
    if *w est libre* **then**
        $S \leftarrow S \cup (m, w)$
        $F \leftarrow F \setminus m$
    else
        $m' \leftarrow$ fiancé actuel de w
        if *w préfère m à m' dans P_w* **then**
            $S \leftarrow (S \setminus (m', w)) \cup (m, w)$
            $F \leftarrow (F \setminus m) \cup m'$
        end
    end
end

Pour illustrer le déroulement de l'algorithme, considérons un exemple simple. Les préférences exprimées par les candidats sont présentées dans la figure 5.2, et celles des établissements dans la figure 5.3.

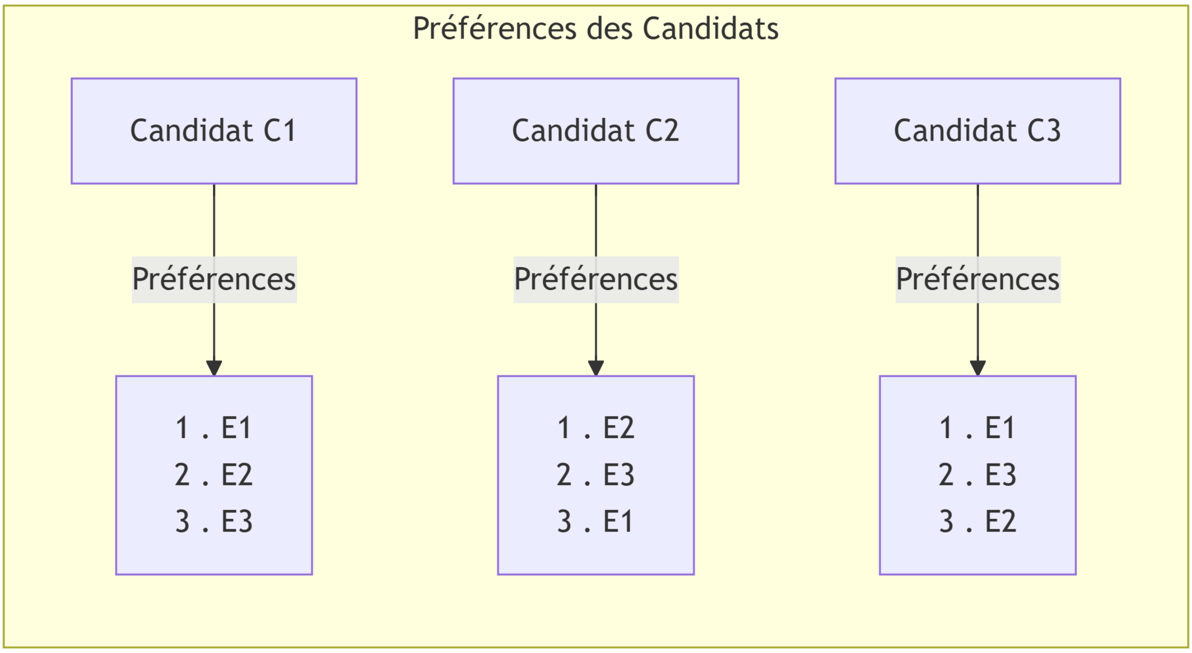


Figure 5.2 – Préférences des candidats (exemple 1).

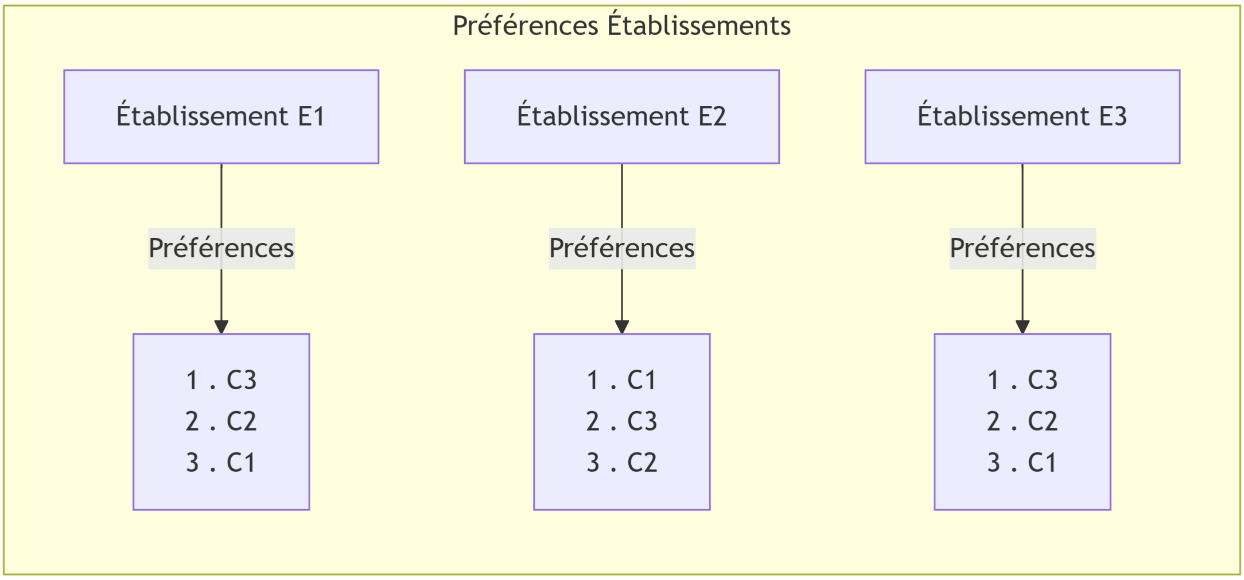


Figure 5.3 – Préférences d'établissements (exemple 1).

Le déroulement est comme montré dans la figure 6.4 :

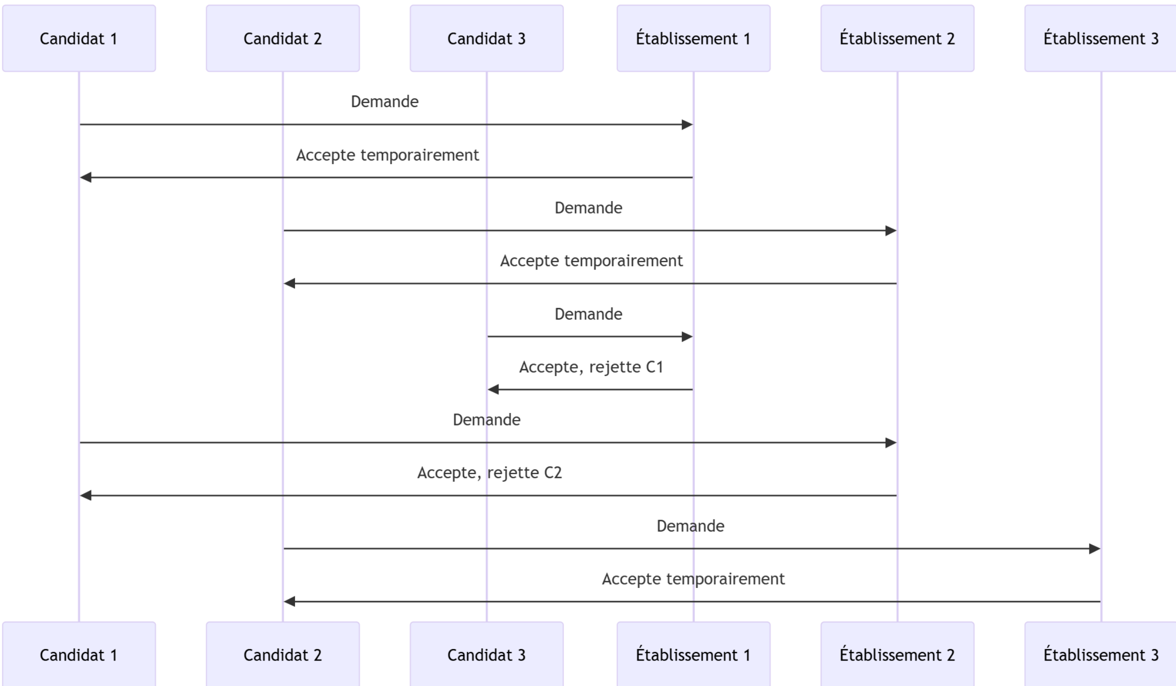


Figure 5.4 – Déroulement de l'algorithme de Gale-Shapley (exemple 1).

Après l'exécution nous allons avoir les résultats comme montrées dans la figure :

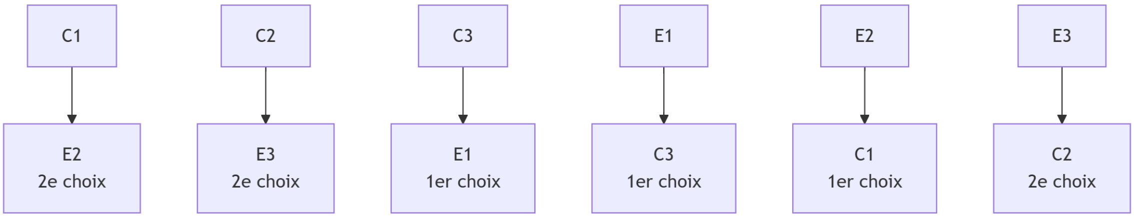
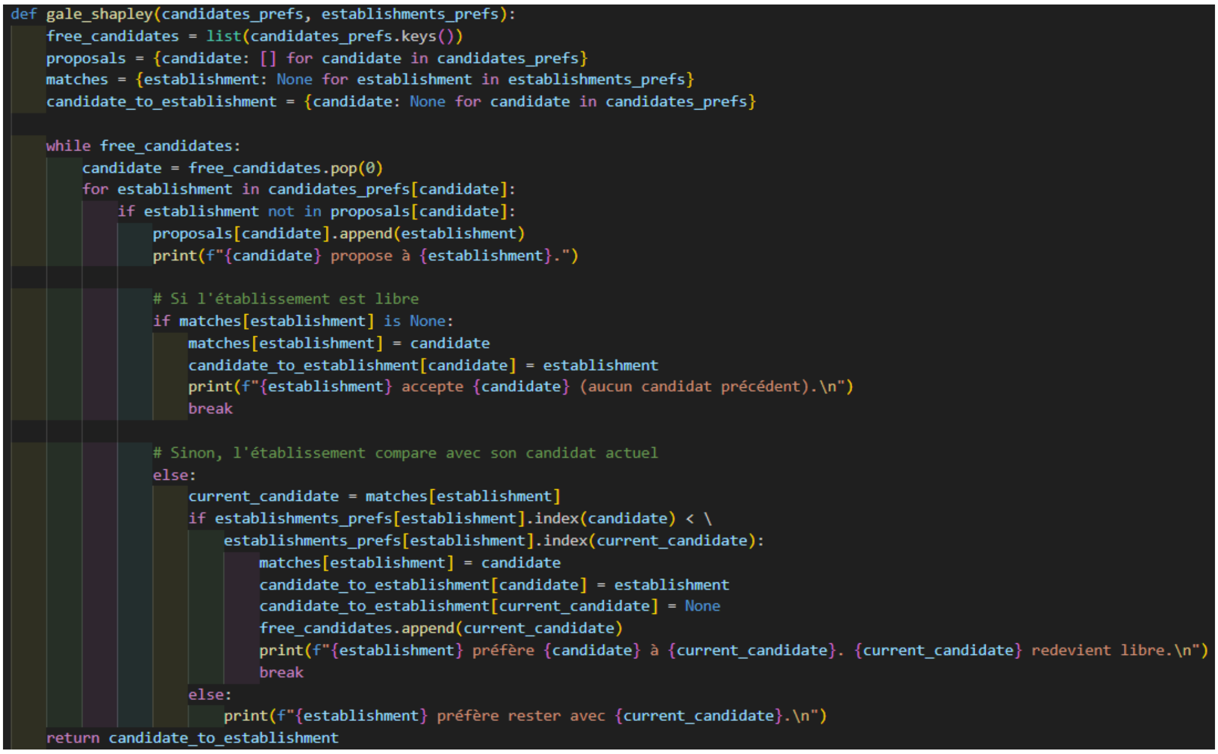


Figure 5.5 – Résultats d'exécution de l'algorithme de Gale-Shapley (exemple 1).

5.4 Implémentation de l'algorithme de Gale-Shapley en Python

Pour l'implémentation de l'algorithme de Gale-Shapley, nous avons choisi d'utiliser le langage de programmation Python en raison de sa simplicité, tant au niveau de la syntaxe que de la transformation des algorithmes en code. Cette implémentation sera utilisée ultérieurement pour tester les résultats des solutions proposées dans le cadre des questions de ce projet.



```python
def gale_shapley(candidates_prefs, establishments_prefs):
    free_candidates = list(candidates_prefs.keys())
    proposals = {candidate: [] for candidate in candidates_prefs}
    matches = {establishment: None for establishment in establishments_prefs}
    candidate_to_establishment = {candidate: None for candidate in candidates_prefs}

    while free_candidates:
        candidate = free_candidates.pop(0)
        for establishment in candidates_prefs[candidate]:
            if establishment not in proposals[candidate]:
                proposals[candidate].append(establishment)
                print(f"{candidate} propose à {establishment}.")

                # Si l'établissement est libre
                if matches[establishment] is None:
                    matches[establishment] = candidate
                    candidate_to_establishment[candidate] = establishment
                    print(f"{establishment} accepte {candidate} (aucun candidat précédent).\n")
                    break

                # Sinon, l'établissement compare avec son candidat actuel
                else:
                    current_candidate = matches[establishment]
                    if establishments_prefs[establishment].index(candidate) < \
                        establishments_prefs[establishment].index(current_candidate):
                            matches[establishment] = candidate
                            candidate_to_establishment[candidate] = establishment
                            candidate_to_establishment[current_candidate] = None
                            free_candidates.append(current_candidate)
                            print(f"{establishment} préfère {candidate} à {current_candidate}. {current_candidate} redevient libre.\n")
                            break
                    else:
                        print(f"{establishment} préfère rester avec {current_candidate}.\n")
    return candidate_to_establishment
```

Figure 5.6 – Capture d'écran de l'implémentation de l'algorithme de Gale-Shapley.

6 Explication des éléments de la solution

6.1 Question 1

6.1.1 Méthode proposée pour la mesure de la satisfaction

Notre approche pour la mesure de la satisfaction repose sur une information supplémentaire fournie par les candidats et les établissements lors de l'expression de leurs préférences classées.

Concrètement, chaque candidat ou établissement, en soumettant sa liste de préférences, doit également spécifier un pourcentage minimal de satisfaction dans sa liste. Si le choix respecte ce pourcentage de satisfaction, le candidat ou l'établissement est considéré comme satisfait ; sinon, il ne l'est pas.

Le diagramme affiché dans la figure 6.1 montre le déroulement de la méthode de mesure de satisfaction des candidats et établissements.

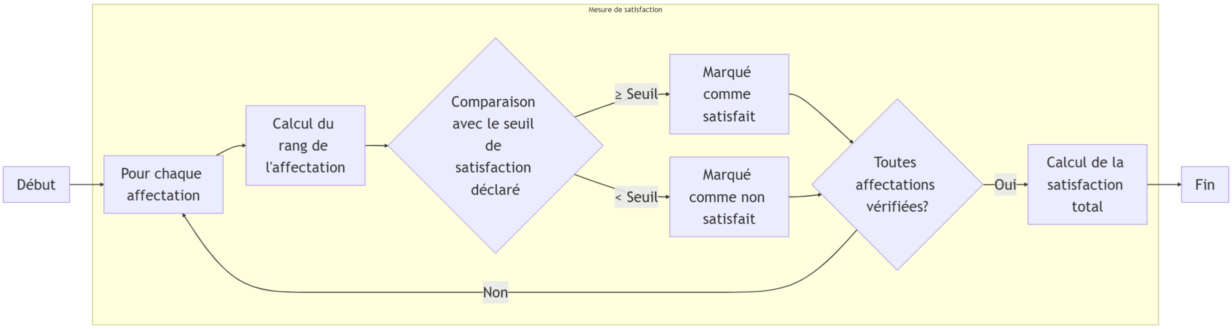


Figure 6.1 – Diagramme montrant la méthode proposée pour la mesure de la satisfaction.

L'algorithme de la méthode proposée est le suivant :

Algorithm 2: Algorithme pour la mesure de satisfaction après Gale-Shapley

Data: S : ensemble des couples formés par Gale-Shapley
P_c : préférences des candidats ; P_e : préférences des établissements
T_c : seuils de satisfaction des candidats ; T_e : seuils de satisfaction des établissements
Result: R_c : satisfaction des candidats ; R_e : satisfaction des établissements

$R_c \leftarrow \emptyset$ `// Ensemble des résultats pour les candidats`
$R_e \leftarrow \emptyset$ `// Ensemble des résultats pour les établissements`
foreach $(c, e) \in S$ **do**
    `// Calculer la satisfaction du candidat` c
    $rank_c \leftarrow$ position de e dans la liste $P_c[c]$
    $percent_c \leftarrow 100 \times (1 - rank_c/|P_c[c]|)$ `// Pourcentage de satisfaction`
    if $percent_c \geq T_c[c]$ **then**
        $R_c[c] \leftarrow Satisfait$
    else
        $R_c[c] \leftarrow Non\ satisfait$
    end
    `// Calculer la satisfaction de l'établissement` e
    $rank_e \leftarrow$ position de c dans la liste $P_e[e]$
    $percent_e \leftarrow 100 \times (1 - rank_e/|P_e[e]|)$
    if $percent_e \geq T_e[e]$ **then**
        $R_e[e] \leftarrow Satisfait$
    else
        $R_e[e] \leftarrow Non\ satisfait$
    end
end
return R_c, R_e

Pour illustrer la méthode proposée, considérons un exemple simple. Les préférences exprimées par les candidats sont présentées dans la figure 6.2, et celles des établissements dans la figure 6.3. Les seuils de satisfaction (pourcentages) y sont également indiqués.

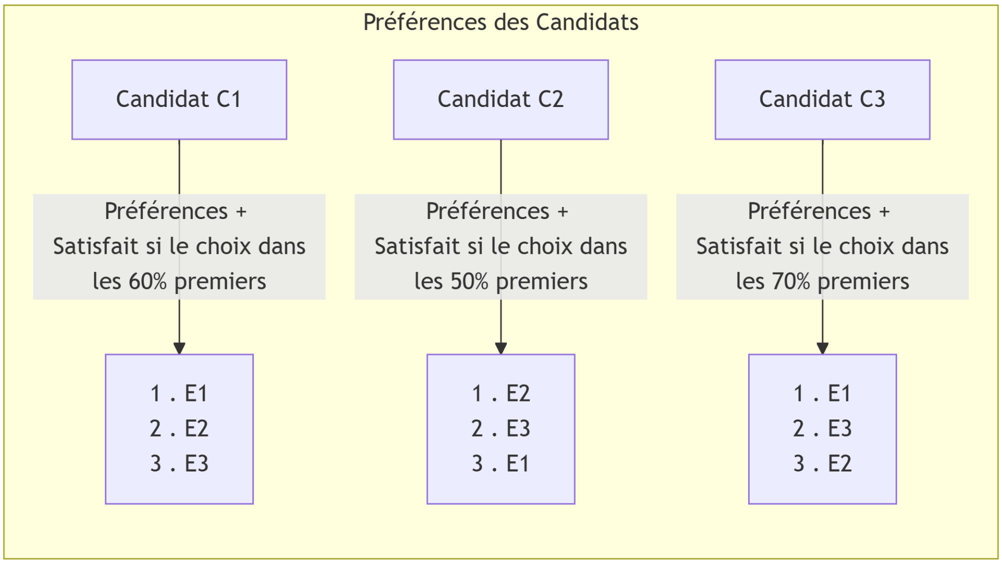


Figure 6.2 – Préférences des candidats (exemple 2).

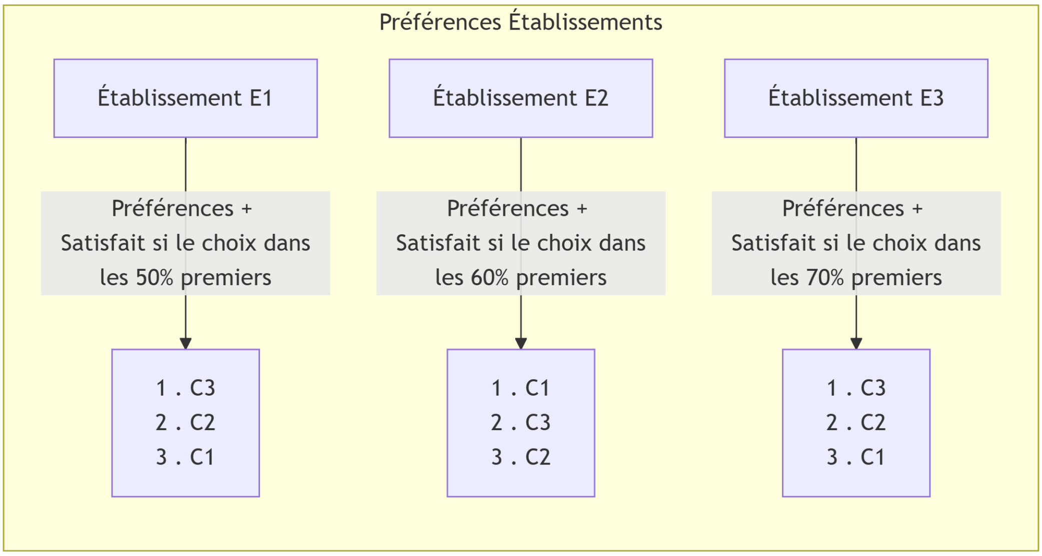


Figure 6.3 – Préférences d'établissements (exemple 2).

Les résultats obtenues sont montrés dans la figure 6.4.

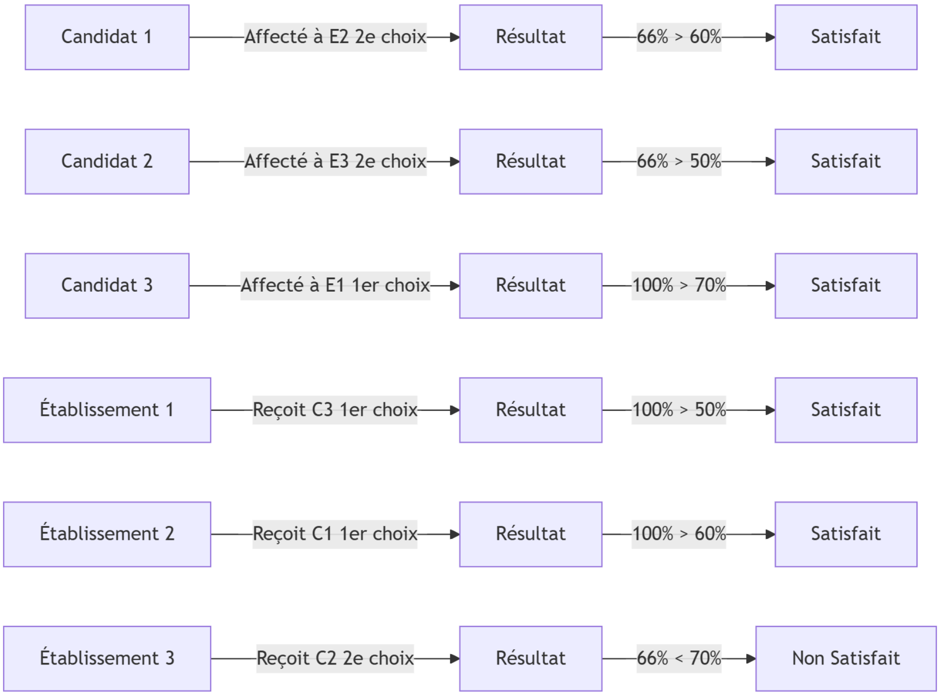
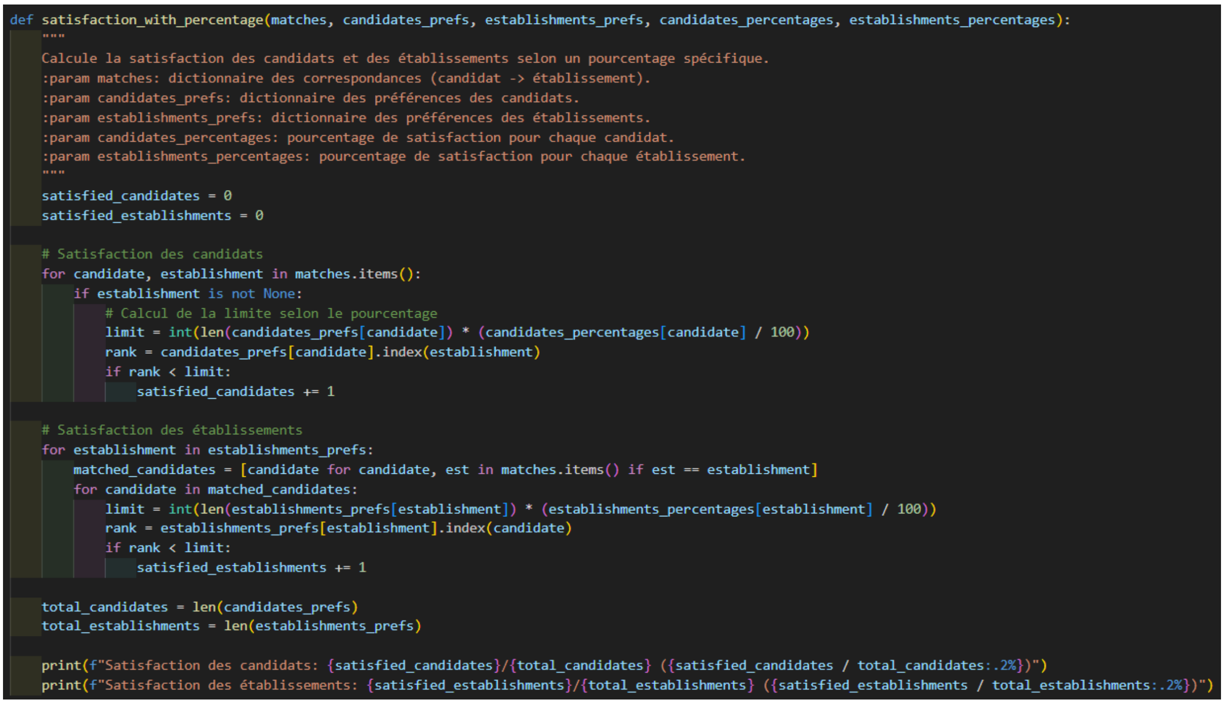


Figure 6.4 – Résultats de la méthode de mesure de satisfaction (exemple 2).

Le rapport final de satisfaction des candidats est de 3/3, soit 100%. En revanche, celui des établissements est de 2/3, soit 66%.

6.1.2 Implémentation en python



```python
def satisfaction_with_percentage(matches, candidates_prefs, establishments_prefs, candidates_percentages, establishments_percentages):
    """
    Calcule la satisfaction des candidats et des établissements selon un pourcentage spécifique.
    :param matches: dictionnaire des correspondances (candidat -> établissement).
    :param candidates_prefs: dictionnaire des préférences des candidats.
    :param establishments_prefs: dictionnaire des préférences des établissements.
    :param candidates_percentages: pourcentage de satisfaction pour chaque candidat.
    :param establishments_percentages: pourcentage de satisfaction pour chaque établissement.
    """
    satisfied_candidates = 0
    satisfied_establishments = 0

    # Satisfaction des candidats
    for candidate, establishment in matches.items():
        if establishment is not None:
            # Calcul de la limite selon le pourcentage
            limit = int(len(candidates_prefs[candidate]) * (candidates_percentages[candidate] / 100))
            rank = candidates_prefs[candidate].index(establishment)
            if rank < limit:
                satisfied_candidates += 1

    # Satisfaction des établissements
    for establishment in establishments_prefs:
        matched_candidates = [candidate for candidate, est in matches.items() if est == establishment]
        for candidate in matched_candidates:
            limit = int(len(establishments_prefs[establishment]) * (establishments_percentages[establishment] / 100))
            rank = establishments_prefs[establishment].index(candidate)
            if rank < limit:
                satisfied_establishments += 1

    total_candidates = len(candidates_prefs)
    total_establishments = len(establishments_prefs)

    print(f"Satisfaction des candidats: {satisfied_candidates}/{total_candidates} ({satisfied_candidates / total_candidates:.2%})")
    print(f"Satisfaction des établissements: {satisfied_establishments}/{total_establishments} ({satisfied_establishments / total_establishments:.2%})")
```

Figure 6.5 – Capture d'écran de l'implémentation de la méthode de mesure satisfaction.

6.2 Question 2

Durant le déroulement de l'algorithme de mariage stable, illustré dans la figure 6.1, et en utilisant la méthode de mesure de satisfaction discutée dans la question 1, nous avons constaté que les candidats sont beaucoup plus satisfaits que les établissements. Cela s'explique par le fait que la priorité a été donnée aux choix des candidats.

À présent, nous allons tester la satisfaction en donnant la priorité aux préférences des établissements et analyser si cette sensibilité à la priorité a un impact sur les résultats.

La figure 6.6 illustre le déroulement en prenant les préférences des établissements comme priorité.

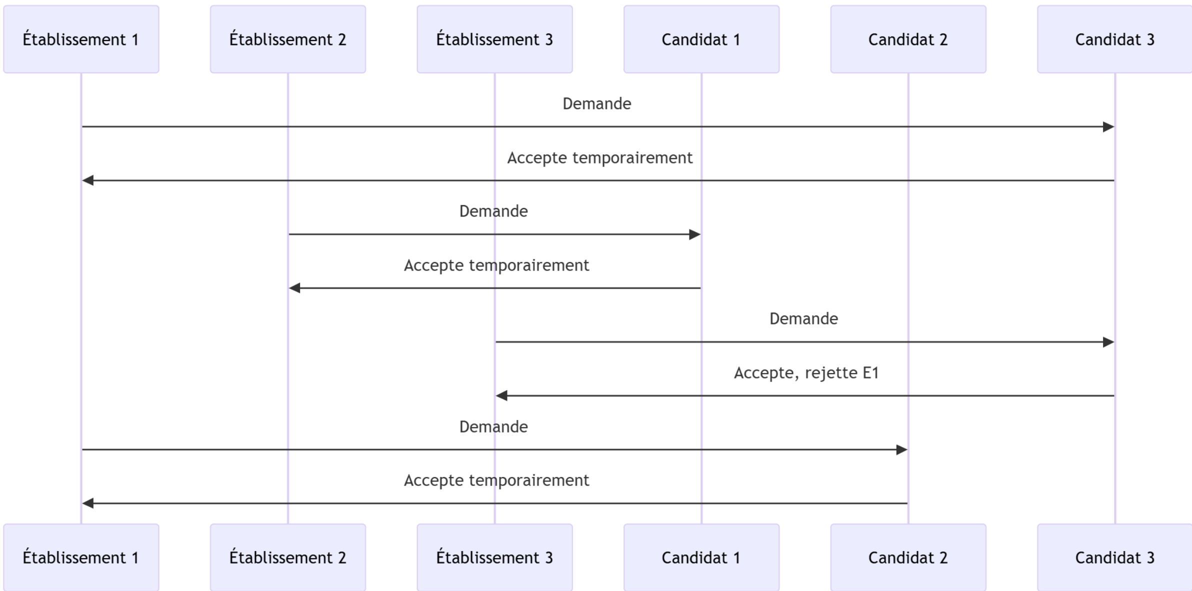


Figure 6.6 – Déroulement en priorisant la préférence des établissement.

Les résultats obtenues sont montrés dans la figure 6.7.

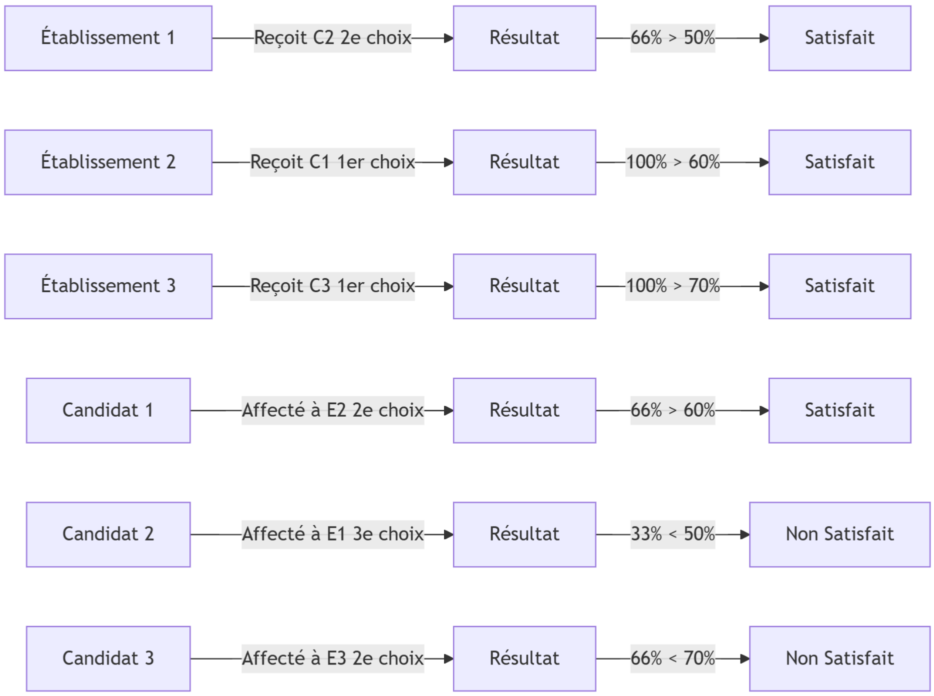


Figure 6.7 – Résultats de satisfaction en priorisant la préférence des établissement.

Le rapport final de satisfaction des établissements est de 3/3, soit 100%. En revanche, celui des candidats est de 1/3, soit 33%.

D'après ces résultats, nous concluons que notre méthode proposée dans la question 1 est sensible à la priorité accordée aux préférences. Si l'on souhaite qu'un établissement soit satisfait (en termes de profits financiers, gains physiques et moraux), il est nécessaire de donner la priorité aux préférences des établissements lors de l'application de l'algorithme. En revanche, si l'objectif est de satisfaire les candidats (en termes de gains moraux, physiques, etc.), il faut alors accorder la priorité aux préférences des candidats.

6.3 Question 3

Le problème du mariage stable consiste à former des correspondances entre deux ensembles (candidats et établissements) en respectant les préférences de chacun, de sorte qu'aucun couple ne puisse s'améliorer en rompant leurs correspondances pour former un nouveau couple. Cependant, dans des scénarios réels, les préférences des individus ou des établissements ne sont pas toujours des listes exhaustives ordonnées.

L'algorithme classique présente des limites par rapport à l'expression des préférences qui peut différer.

— **Taille des ensembles :** Lorsque le nombre de candidats et d'établissements est très grand, une liste exhaustive devient difficile à manipuler.

— **Préférences implicites :** Les préférences peuvent dépendre de critères complexes (par exemple, proximité géographique, compatibilité des compétences).

— **Dynamisme des critères :** Les critères peuvent évoluer ou être influencés par des facteurs externes, rendant les listes statiques peu réalistes.

Pour adresser ces limites, nous proposons d'utiliser des fonctions comme représentation compacte des préférences. Une fonction est une mesure qui attribue un score numérique à chaque correspondance possible. Chaque candidat et établissement évalue les options disponibles en fonction de cette fonction. La correspondance préférée est celle qui maximise cette dernière.

1. **Pour les candidats :** Plus la valeur de la fonction $u_c(c, e)$ est élevée, plus e est préféré par c. Un candidat peut baser ses préférences sur plusieurs critères pondérés comme : Localisation, Salaire, Réputation... La fonction d'utilité globale pour un candidat est donc une combinaison linéaire des différentes utilités pondérées par des coefficients $\alpha_1, \alpha_2, \alpha_3$ reflétant l'importance des critères pour le candidat.

$$\boxed{u_c(c,e) = \alpha_1 u_{\text{distance}}(c,e) + \alpha_2 u_{\text{salaire}}(c,e) + \alpha_3 u_{\text{réputation}}(c,e),}$$

   où $\alpha_1, \alpha_2, \alpha_3$ sont des poids.

2. **Pour les établissements :** Plus la valeur de la fonction $u_e(e, c)$ est élevée, plus c est préféré par e. Un établissement peut évaluer un candidat en fonction de plusieurs critères pondérés, tels que : Compétences, Expérience, Compatibilité culturelle... La fonction d'utilité globale pour un établissement est donc une combinaison linéaire des différentes utilités pondérées par des coefficients $\beta_1, \beta_2, \beta_3$ reflétant l'importance des critères pour l'établissement.

$$\boxed{u_e(e,c) = \beta_1 u_{\text{compétences}}(e,c) + \beta_2 u_{\text{expérience}}(e,c) + \beta_3 u_{\text{culture}}(e,c),}$$

   où $\beta_1, \beta_2, \beta_3$ sont des poids.

Chaque candidat et établissement utilise ces fonctions pour évaluer les correspondances. L'algorithme de cette méthode sera représentée comme suit :

Algorithm 3: Algorithme de Gale-Shapley pour le placement de candidats dans des établissements avec fonctions d'utilité (Question 3)

Input: C : ensemble des candidats, E : ensemble des établissements
Input: P_c, P_e : listes des préférences des candidats et des établissements
Input: α_i : poids des critères pour chaque candidat, β_i : poids des critères pour chaque établissement
Output: S : ensemble des affectations stables (candidats-établissements)

$S \leftarrow \emptyset$ `// Initialisation de l'ensemble des affectations`
$L \leftarrow C$ `// Ensemble des candidats libres`
while $L \neq \emptyset$ **do**
  Choisir un candidat $c \in L$;
  Calculer les utilités pour chaque établissement $e \in E$:

$$u_c(e) = \alpha_1 \cdot u_{\text{compétence}}(c, e) + \alpha_2 \cdot u_{\text{localisation}}(c, e) + \alpha_3 \cdot u_{\text{salaire}}(c, e)$$

  $e \leftarrow$ L'établissement qui maximise $u_c(e)$;
  if e *est libre* **then**
    $S \leftarrow S \cup \{(c, e)\}$ `//` c `et` e `forment une affectation`
    $L \leftarrow L \setminus \{c\}$ `//` c `n'est plus libre`
  else
    $c' \leftarrow$ Candidat actuellement affecté à e;
    Calculer les utilités pour c et c' :

$$u_e(c) = \beta_1 \cdot u_{\text{compétence}}(e, c) + \beta_2 \cdot u_{\text{localisation}}(e, c) + \beta_3 \cdot u_{\text{salaire}}(e, c)$$

$$u_e(c') = \beta_1 \cdot u_{\text{compétence}}(e, c') + \beta_2 \cdot u_{\text{localisation}}(e, c') + \beta_3 \cdot u_{\text{salaire}}(e, c')$$

    if $u_e(c) > u_e(c')$ **then**
      $S \leftarrow (S \setminus \{(c', e)\}) \cup \{(c, e)\}$ `// Remplacer` (c', e) `par` (c, e)
      $L \leftarrow (L \setminus \{c\}) \cup \{c'\}$ `//` c' `devient libre,` c `est affecté à` e

return S

6.4 Question 4

6.4.1 Algorithme de Gale-Shapley avec les variantes

Voici une version modifiée de l'algorithme de Gale-Shapley qui prend en compte les relations de préférences incomplètes et avec égalité :

Algorithm 4: Algorithme de Gale-Shapley modifié pour l'affectation stable avec préférences incomplètes et égalité (Question 4)

Data: C : ensemble de candidats, E : ensemble d'établissements, P_c, P_e : listes de préférences, cap_e : capacités des établissements

Result: S : ensemble de correspondances stables

```
S ← ∅ F ← C // Ensemble des candidats libres
while F ≠ ∅ do
    Choisir c ∈ F e ← premier établissement dans P_c que c n'a pas encore demandé
    if |S_e| < cap_e then
        S ← S ∪ {(c, e)}
        F ← F \ {c}
    else
        C_e ← {candidats actuellement affectés à e}
        c_min ← candidat le moins préféré dans C_e selon P_e
        if c est préféré à c_min dans P_e then
            S ← (S \ {(c_min, e)}) ∪ {(c, e)}
            F ← (F \ {c}) ∪ {c_min}
        else if c est à égalité avec c_min dans P_e then
            S ← S ∪ {(c, e)}
            F ← F \ {c}
        end
    end
end
```

Cette version modifiée de l'algorithme de Gale-Shapley prend en compte les relations de préférences incomplètes et avec égalité. Voici les principales modifications apportées :

1. Les établissements ont maintenant une **capacité d'accueil**, permettant d'accepter plusieurs candidats.
2. Les listes de préférences peuvent être incomplètes. Si un candidat ou un établissement n'est pas dans la liste de préférences, **il est considéré comme moins préféré que tous ceux qui sont listés**.
3. En cas d'égalité de préférence, l'établissement peut accepter le nouveau candidat sans rejeter le candidat actuel.

Ces modifications permettent de mieux représenter des scénarios réels, comme le système Parcoursup, où les préférences peuvent être incomplètes et les établissements ont des capacités d'accueil variables.

6.4.2 Déroulement de l'algorithme de Gale-Shapley avec les relations de préférences incomplètes et/ou avec égalité

Les préférences des candidats sont :

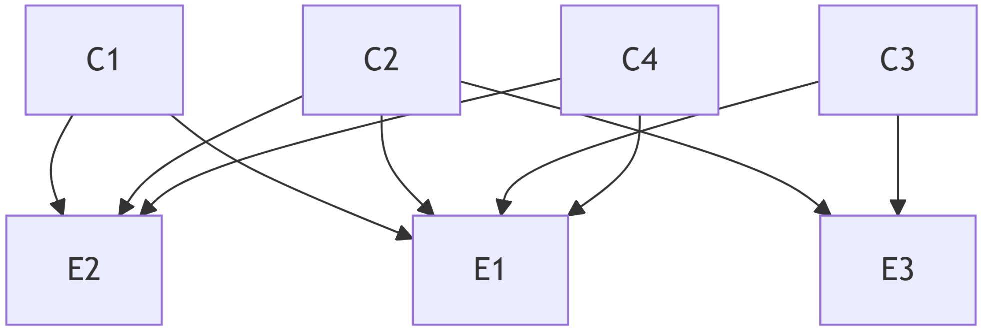


Figure 6.8 – Préférences incomplet des candidats (exemple 3)

Et les préférences des établissements sont :

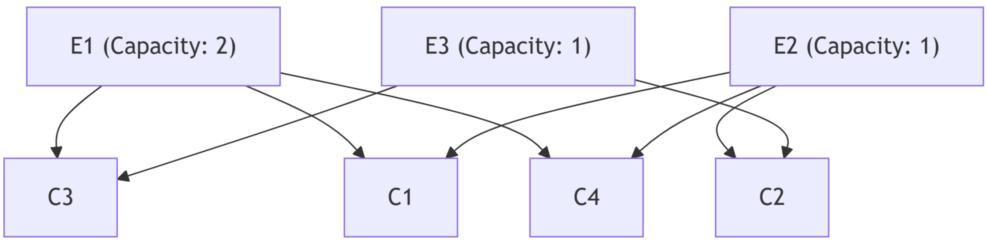


Figure 6.9 – Préférences et Capacités incomplet d'établissements (exemple 3)

L'application de l'algorithme de Gale-Shapley modifié pour l'affectation stable avec préférences incomplètes et égalité se déroule comme montré dans la figure 6.10.

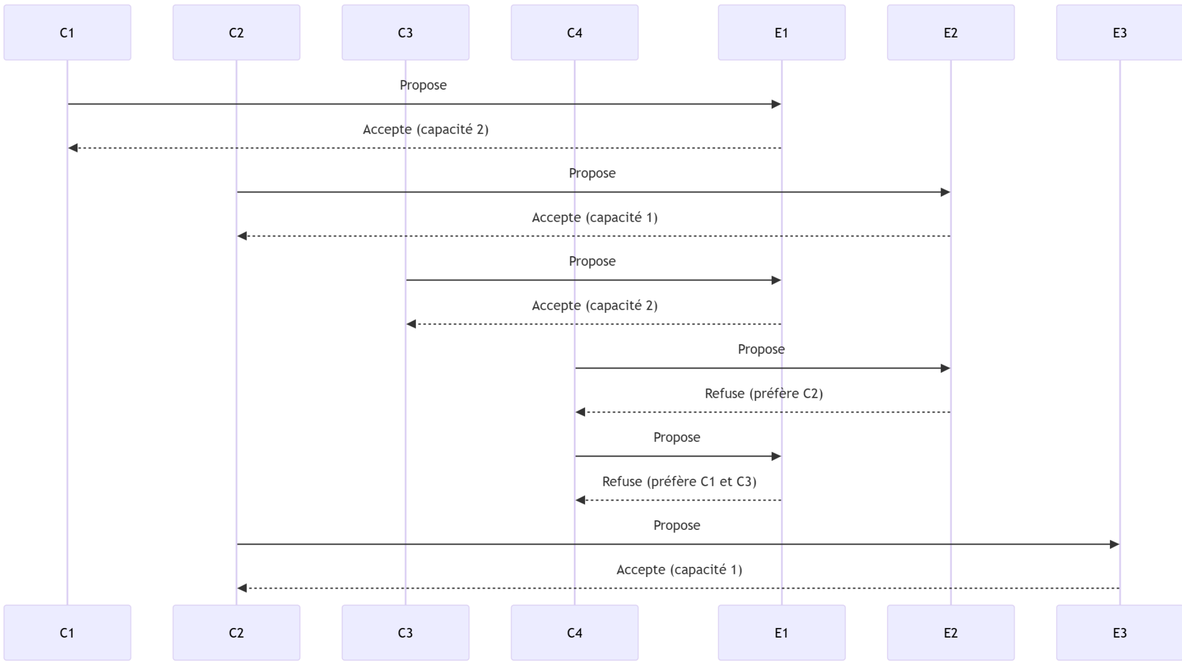


Figure 6.10 – Déroulement de l'algorithme de Gale-Shapley modifié (exemple 3).

Le résultat final d'exécution de cette dernière est montré dans la figure 6.11.

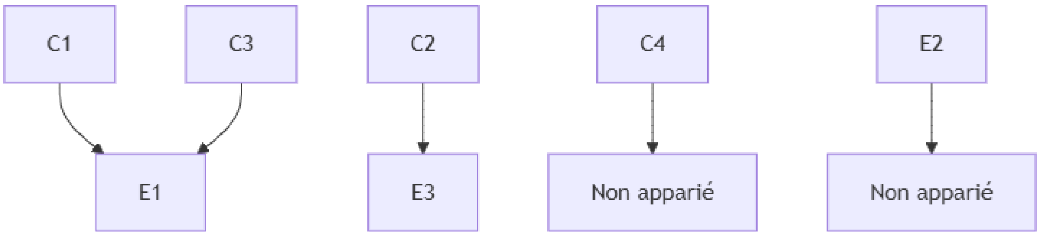


Figure 6.11 – Résultats d'exécution de Gale-Shapley modifié (exemple 3).

6.4.3 Implémentation de l'algorithme en Python

La partie modifiée dans l'implémentation précédente de l'algorithme de Gale-Shapley, présentée précédemment (figure 6.12), est illustrée dans la figure ci-dessous :

```python
def gale_shapley_modified(candidates_prefs, establishments_prefs):
    free_candidates = list(candidates_prefs.keys())
    proposals = {candidate: [] for candidate in candidates_prefs}
    matches = {establishment: [] for establishment in establishments_prefs}
    candidate_to_establishment = {candidate: None for candidate in candidates_prefs}
    while free_candidates:
        candidate = free_candidates.pop(0)
        for establishment in candidates_prefs[candidate]:
            if establishment not in proposals[candidate]:
                proposals[candidate].append(establishment)
                print(f"{candidate} propose à {establishment}.")
                if len(matches[establishment]) < establishments_prefs[establishment]["capacity"]:
                    matches[establishment].append(candidate)
                    candidate_to_establishment[candidate] = establishment
                    print(f"{establishment} accepte {candidate} (place disponible).\n")
                    break
                else:
                    current_candidates = matches[establishment]
                    worst_current = min(current_candidates, key=lambda c: establishments_prefs[establishment]["preferences"].index(c)
                                        if c in establishments_prefs[establishment]["preferences"] else float('inf'))
                    if candidate in establishments_prefs[establishment]["preferences"]:
                        if establishments_prefs[establishment]["preferences"].index(candidate) < establishments_prefs[establishment]["preferences"].index(worst_current):
                            matches[establishment].remove(worst_current)
                            matches[establishment].append(candidate)
                            candidate_to_establishment[candidate] = establishment
                            candidate_to_establishment[worst_current] = None
                            free_candidates.append(worst_current)
                            print(f"{establishment} préfère {candidate} à {worst_current}. {worst_current} redevient libre.\n")
                            break
                        elif establishments_prefs[establishment]["preferences"].index(candidate) == establishments_prefs[establishment]["preferences"].index(worst_current):
                            matches[establishment].append(candidate)
                            candidate_to_establishment[candidate] = establishment
                            print(f"{establishment} accepte {candidate} (égalité de préférence).\n")
                            break
                    print(f"{establishment} préfère garder ses candidats actuels.\n")
    return candidate_to_establishment
```

Figure 6.12 – Implémentation de l'algorithme de Gale-Shapley modifié.

On a arrivé a trouver les mêmes résultats, une capture d'écran dans la figure 6.13 montre ces derniers.

```
Correspondances finales:
C1 - E1
C2 - E2
C3 - E1
C4 - None
C2 - E2
C3 - E1
C4 - None
C3 - E1
C4 - None
C4 - None
```

Figure 6.13 – Résultats d'exécution de l'algorithme de Gale-Shapley modifié (exemple 3).

7 Conclusion

Ce projet a mis en lumière la richesse et la complexité des algorithmes du mariage stable, notamment dans leur capacité à résoudre des problèmes d'appariement dans divers contextes. En explorant les variantes de l'algorithme classique, nous avons pu démontrer à quel point il est essentiel d'adapter ces outils pour répondre aux exigences spécifiques des problèmes à traiter, tels que les préférences incomplètes ou l'égalité dans les choix.

Toutefois, une difficulté majeure réside dans l'estimation nécessaire pour satisfaire pleinement les attentes des différentes parties. En effet, dans la majorité des cas, il est pratiquement impossible d'obtenir une solution parfaite, où tous les individus ou entités atteignent un niveau de satisfaction optimal. Ce constat met en évidence une caractéristique fondamentale de ce type de problématique : la nécessité de faire des compromis.

Ces compromis peuvent porter sur plusieurs aspects : équilibrer la satisfaction des deux groupes en présence, prioriser les préférences d'un groupe sur l'autre, ou encore accepter des résultats suboptimaux pour certaines parties afin de garantir la stabilité globale de l'appariement. Cette notion de compromis, inhérente aux algorithmes d'appariement, reflète également les contraintes et la complexité des situations réelles.

Pour l'avenir, il serait intéressant d'explorer des approches combinées qui intègrent les algorithmes classiques avec des outils plus modernes, tels que l'apprentissage automatique ou les systèmes multi-agents. Ces méthodes pourraient offrir une meilleure capacité à s'adapter aux spécificités des contextes, tout en minimisant les compromis nécessaires. En conclusion, l'algorithme du mariage stable reste un outil puissant et adaptable, mais son application exige toujours une analyse fine des besoins et des contraintes du problème à résoudre.