Sorbonne Université
Master ANDROIDE

Explorable explanations de notions de partage équitable

UE de projet M1

Baran Açikel – Kaan Dişli – Guillaume Lebreton

2025

Table des matières

1 Introduction **1**

2 Notion de partage équitable **2**
2.1 La notion d'envie . . . 2
2.1.1 Représentation par graphe d'envie . . . 2
2.2 Algorithmes de partage équitable . . . 3
2.2.1 Picking Sequences . . . 3
2.2.2 Algorithme de Lipton . . . 3

3 Explorable explanation **5**
3.1 Structure . . . 5
3.2 Interface . . . 6
3.3 Agents . . . 6
3.3.1 Représentation . . . 6
3.3.2 Déclaration . . . 7
3.4 Envies . . . 7
3.4.1 Calcul . . . 7
3.4.2 Illustration . . . 8
3.5 Algorithme . . . 8
3.5.1 Pool . . . 9
3.5.2 Structures de données communes . . . 9
3.5.3 Protocole de séquence de choix (picking sequences) . . . 9
3.5.4 Algorithme d'élimination de cycles de Lipton . . . 10
3.6 Limites et améliorations possibles . . . 12

4 Conclusion **13**

A Cahier des charges **15**

B Manuel utilisateur **17**
B.1 Fonctionnalités . . . 17
B.2 Prise en main . . . 17
B.2.1 Pré-requis . . . 17
B.2.2 Installation . . . 17
B.2.3 Développement . . . 18
B.2.4 Compilation pour la production . . . 18
B.2.5 Aperçu du build de production . . . 18
B.3 Pour en savoir plus . . . 18

Chapitre 1

Introduction

Les problèmes de **partage équitable** sont présents dans de nombreuses situations du quotidien : partage de ressources, organisation de tâches, répartition de biens, etc. Pourtant, les concepts fondamentaux d'équité, comme l'absence d'envie ou l'équité proportionnelle, restent souvent abstraits et difficiles à saisir pour un public non spécialiste. Bien qu'ils soient rigoureusement définis en théorie, leur compréhension nécessite des outils pédagogiques adaptés.

Dans ce contexte, notre projet s'inscrit dans une démarche de vulgarisation interactive, fondée sur le concept d'**explorable explanation**. L'objectif est de concevoir une interface interactive, à la fois pédagogique et intuitive, permettant d'expliquer visuellement et dynamiquement certaines notions clés de la théorie du partage équitable.

Pour cela, nous avons choisi d'utiliser SvelteKit, un framework moderne pour le développement d'interfaces web interactives. Ce choix nous permet une plus grande flexibilité dans la création de visualisations dynamiques et d'interactions complexes, par rapport à d'autres outils plus spécialisés comme Idyll. (Pas sur qu'on l'aborde ici mais peut etre bien de dire sur quoi on a travailler)

Nous nous attacherons plus particulièrement à explorer la notion d'envie et ses possibles relaxations dans certains contextes.

Le travail porte également sur deux protocoles bien connus :

- **Le protocole de Lipton**, fondé sur l'élimination des cycles d'envie.
- **Les picking sequences**, qui structurent la répartition à travers des séquences de choix ordonnés.

Ce projet s'inscrit dans le cadre de l'UE de projet M1 du master ANDROIDE de Sorbonne Université. Il est encadré par Nicolas Maudet, professeur à Sorbonne Université.

Le code source de ce projet est disponible sur github

Chapitre 2

Notion de partage équitable

Avant de concevoir une interface interactive, nous avons commencé par étudier les notions fondamentales de partage équitable. Cette phase théorique nous a permis de mieux comprendre les différents critères d'équité, ainsi que les protocoles existants pour répartir des biens indivisibles entre plusieurs agents.

Nos réflexions se sont notamment appuyées sur le chapitre ***Fair Allocation of Indivisible Goods*** de Sylvain Bouveret, Yann Chevaleyre et Nicolas Maudet [**bouveret2016fair**], qui présente de manière claire et rigoureuse les principaux concepts de la discipline. Ce travail nous a servi de base pour identifier les critères d'équité que nous souhaitions illustrer, ainsi que les protocoles que nous avons choisi de mettre en œuvre.

2.1 La notion d'envie

On dit qu'un agent *envie* un autre s'il préfère la part reçue par ce dernier à la sienne. L'absence d'envie (*envy-freeness*) est une condition centrale en partage équitable : une allocation est dite **envy-free (EF)** si aucun agent n'envie un autre.

On distingue plusieurs variantes selon le degré d'exigence :

- **Lot entiers (EF)** : les agents évaluent la totalité du lot des autres agents ; l'absence d'envie est évaluée globalement.
- **One-Free (EF1)** : une allocation est EF1 si, en retirant un bien du lot de l'autre, l'envie disparaît.
- **EFX (envy-free up to any eXcluded good)** : une condition plus forte que EF1 ; l'envie disparaît quel que soit le bien retiré.

Ces relaxations sont particulièrement utiles dans les cas où l'équité parfaite (EF) est difficile, voire impossible, à atteindre.

2.1.1 Représentation par graphe d'envie

Les relations d'envie entre agents peuvent être représentées par un **graphe orienté**, appelé *graphe d'envie*. Un arc de l'agent A vers B indique que A envie la part de B. Cette représentation permet d'identifier des structures conflictuelles dans l'allocation, comme les **cycles d'envie**, qui peuvent empêcher l'équité.

Cette notion est notamment utilisée dans l'algorithme de Lipton, qui s'appuie sur l'identification et la suppression de ces cycles.

2.2 Algorithmes de partage équitable

2.2.1 Picking Sequences

Ce protocole repose sur une séquence prédéfinie de tours de choix. Chaque agent, à son tour, choisit un bien encore disponible. La structure de la séquence (*picking sequence*) influe fortement sur le résultat en termes d'équité.

Bien que ce protocole soit simple à mettre en œuvre et compréhensible pour les utilisateurs, il ne garantit pas nécessairement l'absence d'envie, sauf dans des cas particuliers (par exemple, si les préférences sont alignées ou si la séquence est symétrique).

Dans notre projet, nous avons exploré trois types de séquences de choix :

— **Séquence répétée** : une même séquence d'agents est répétée jusqu'à épuisement des biens. Par exemple, avec trois agents A, B et C, une séquence répétée pourrait être $A \to B \to C \to A \to B \to C$, etc. Ce type de séquence assure une certaine régularité dans les tours, mais peut générer des situations d'envie si les biens les plus valorisés apparaissent tôt.
— **Séquence miroir** : une séquence est suivie de sa version inversée, et ainsi de suite (ex. $A \to B \to C \to C \to B \to A$). Cette structure vise à compenser les avantages liés à l'ordre de passage en équilibrant les positions avantageuses et défavorables sur deux tours successifs. Elle tend à limiter l'envie entre agents, sans pour autant la garantir totalement.
— **Séquence aléatoire** : l'ordre des tours est généré aléatoirement à chaque itération. Cela permet de simuler des répartitions plus variées, mais au prix d'une perte de contrôle sur la structure et la perception d'équité du protocole. Ce type de séquence peut introduire de la variabilité dans les résultats, utile pour l'exploration de cas limites ou non standards.

Ces différentes variantes permettent d'observer comment la structure même du protocole influence les résultats, et offrent à l'utilisateur une expérience interactive pour expérimenter les effets de différentes stratégies de choix.

Dans le cadre des utilités additives toute picking sequence, quelle que soit sa structure (répétée, miroir ou aléatoire), conduit à une répartition EF1 si les agents choisissent leur bien préféré à chaque tour

2.2.2 Algorithme de Lipton

L'algorithme de Lipton propose une méthode itérative pour construire une allocation EF. Il fonctionne comme suit :

1. Les biens sont attribués un par un.
2. Après chaque attribution, on construit le graphe d'envie.
3. Si un **cycle d'envie** est détecté, une réaffectation est effectuée pour l'éliminer.

Tant que le graphe d'envie reste acyclique, aucune envie cyclique persistante ne bloque la progression de l'allocation. Cet algorithme garantit une allocation EF1 dans le cas de préférences additives.

Chapitre 3

Explorable explanation

Après avoir présenté dans le chapitre précédent les fondements théoriques des notions d'envie et les algorithmes associés, nous abordons ici la mise en œuvre concrète de ces concepts dans le cadre d'une **explorable explanation**.

Une **explorable explanation** est une approche pédagogique interactive visant à faciliter la compréhension de notions complexes par l'exploration active. Elle combine généralement du contenu textuel, des visualisations dynamiques et des éléments interactifs, permettant à l'utilisateur de manipuler les paramètres d'un système et d'en observer immédiatement les conséquences. Ce type de dispositif est particulièrement adapté aux domaines théoriques ou abstraits, tels que la théorie du partage équitable, car il permet d'illustrer concrètement les mécanismes à l'œuvre et de favoriser une appropriation intuitive des concepts.

L'objectif principal de notre projet est donc de proposer une interface interactive permettant à un utilisateur de manipuler les paramètres du problème et d'en observer directement les conséquences sur l'équité de la répartition.

Dans ce chapitre, nous détaillons les choix de modélisation, les représentations graphiques adoptées, ainsi que les interactions mises en place pour illustrer de manière intuitive les notions théoriques. Nous mettons particulièrement l'accent sur la façon dont l'utilisateur peut explorer dynamiquement les situations de partage, afin de mieux comprendre les conditions d'absence d'envie et les mécanismes des protocoles étudiés.

3.1 Structure

Pour introduire la notion d'envie dans notre projet, nous avons choisi de représenter un scénario simplifié de répartition de biens entre plusieurs agents. Les biens sont symbolisés par des boules de couleur — rouge, verte, bleue, jaune et violette — chacune correspondant à un type de bien distinct. Chaque agent associe à ces types de biens des valeurs différentes, selon ses préférences individuelles, exprimées par des utilités comprises entre 0 et 10.

Nous faisons l'hypothèse que les agents évaluent les lots selon une fonction d'utilité additive : la valeur totale d'un lot correspond simplement à la somme des utilités des biens qu'il contient. Par exemple, si un agent possède deux boules rouges et une boule jaune, et qu'il attribue une valeur de 2 à une boule rouge et de 5 à une boule jaune, alors la valeur

totale de ce lot pour cet agent sera de $2 \times 2 + 5 \times 1 = 9$.

3.2 Interface

On a opté pour Svelte et SvelteKit afin de bénéficier d'une approche réactive et performante : contrairement aux frameworks traditionnels, Svelte compile les composants en code JavaScript optimisé, réduisant drastiquement la taille du bundle et le temps de chargement. Cette légèreté améliore l'expérience utilisateur, notamment sur mobiles ou connexions lentes, tout en simplifiant le développement grâce à une syntaxe déclarative et intuitive.

SvelteKit, quant à lui, offre un cadre complet pour le rendu côté serveur (SSR) et le pré-rendu statique, assurant une excellente SEO et des transitions ultra-fluides. L'écosystème mature, la prise en charge native du routage et la compatibilité avec Vite facilitent l'intégration de notre simulation d'allocation équitable, garantissant rapidité de mise en œuvre et maintenabilité.

L'utilisation d'Idyll (ce qui était recommandé) se révélait limitée pour construire une interface à la fois réactive et performante : le système de templates n'intégrait pas nativement de liaison bidirectionnelle des données, ce qui compliquait grandement la mise à jour dynamique des préférences et des résultats de simulation. Chaque modification nécessitait des manipulations manuelles du DOM ou des scripts additionnels, alourdissant le développement et pénalisant la fluidité de l'application.

En revanche, Svelte simplifie la gestion de l'état grâce à son modèle réactif : toute variable liée à l'interface se met à jour automatiquement, sans surcharge de runtime. Combiné à SvelteKit pour le rendu côté serveur et le pré-rendu, cela garantit une navigation instantanée et une excellente expérience utilisateur, même sur des jeux de données complexes.

3.3 Agents

3.3.1 Représentation

Les agents sont modélisés en TypeScript via une interface `IAgent`. Chaque agent possède trois attributs principaux :

- un nom (`name`) ;
- un ensemble d'attributions (`attributions`) représentant la quantité de chaque type de bien détenu ;
- une table d'utilité (`utilities`) indiquant la valeur que l'agent attribue à chaque type de bien.

Les types de biens sont codés sous forme de couleurs (`'red'`, `'green'`, `'blue'`, `'yellow'`, `'purple'`), chaque couleur représentant un type de bien distinct.

Pour faciliter l'interaction avec les agents dans l'interface, plusieurs fonctions utilitaires sont définies : `setName`, `setAttribution(s)`, `setUtility(ies)`. Elles permettent de modifier dynamiquement les données d'un agent lors des différentes étapes de la simulation.

La structure repose sur l'hypothèse que les utilités sont additives, ce qui permet d'évaluer facilement la valeur d'un lot comme la somme des utilités de ses biens.

3.3.2 Déclaration

Avant d'évaluer l'équité d'une distribution, l'utilisateur commence par définir les agents et leurs préférences. Pour cela, l'interface intercative ci-dessous permet de spécifier, pour chaque agent, les biens qu'il possède *(attributions)* ainsi que la valeur qu'il associe à une instance de chaque type de bien *(utilité)*. Il est possible d'ajouter ou de supprimer des agents, avec toutefois une contrainte minimale de deux agents pour garantir la pertinence de l'analyse. Afin de faciliter les tests, des boutons permettent également de générer aléatoirement les attributions ou les valeurs d'utilité se trouve en bas de l'interface. Cette structure permet d'expérimenter facilement différents scénarios de répartition.

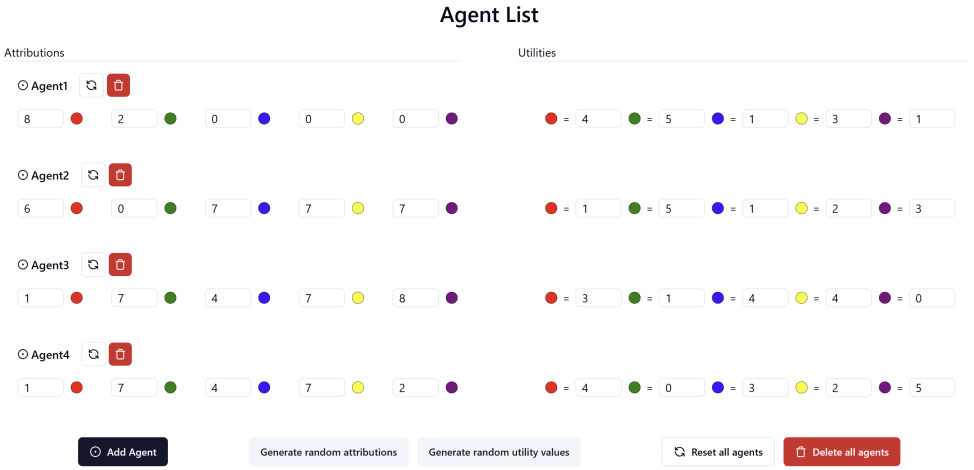

FIGURE 3.1 – Liste des agents avec leurs attributions et préférences

3.4 Envies

3.4.1 Calcul

Pour illustrer concrètement la notion d'envie, nous avons implémenté un histogramme interactif représentant la valeur que chaque agent attribue aux lots, selon ses propres préférences. L'utilisateur peut sélectionner un agent évaluateur ainsi qu'un mode d'évaluation parmi EF, EF1 ou EFX. L'outil permet alors de vérifier si la distribution actuelle satisfait la condition d'équité choisie, du point de vue de l'agent sélectionné.

Pour déterminer si une distribution est EF1 pour un agent donné, nous retirons dans chaque lot des autres agents le bien qu'il préfère. Si, après cette suppression, la valeur des lots restants est inférieure à celle de son propre lot, alors la condition EF1 est satisfaite pour cet agent.

De même, pour tester si une distribution est EFX, nous retirons dans chaque lot des autres agents le bien que l'agent évaluateur apprécie le moins. Si avec ce retrait les lots des autres est moins désirables que le sien, alors la distribution satisfait la condition EFX : en effet,

si l'envie disparaît même après avoir retiré le bien le moins significatif, elle disparaîtra a fortiori après le retrait de n'importe quel bien.

3.4.2 Illustration

L'histogramme affichera alors la valeur de chaque lot selon l'agent évaluateur et le mode d'envie choisis. Si son propre lot est celui auquel il accorde la valeur la plus élevée, alors la distribution est considérée comme *sans envie* pour lui. Dans le cas contraire, il *enviera* les agents possédant des lots qu'il juge meilleurs que le sien, comme dans l'exemple ci-dessous :

L'histogramme est accompagné d'un **graphe des envies** permettant d'avoir une vue d'ensemble des relations d'envie entre agents, en fonction de la répartition actuelle des biens et du *mode d'envie sélectionné*. Dans ce graphe :

— Les **nœuds rouges** représentent les agents qui envient au moins un autre agent.
— Les **flèches** indiquent les directions de ces envies (par exemple, une flèche de A vers B signifie que A envie le lot de B).
— Les **nœuds verts** désignent les agents qui, selon leurs propres préférences, estiment posséder le meilleur lot et ne ressentent donc aucune envie.

Ce mécanisme visuel permet ainsi de mieux comprendre la manière dont les préférences subjectives influencent les perceptions d'équité dans la répartition.

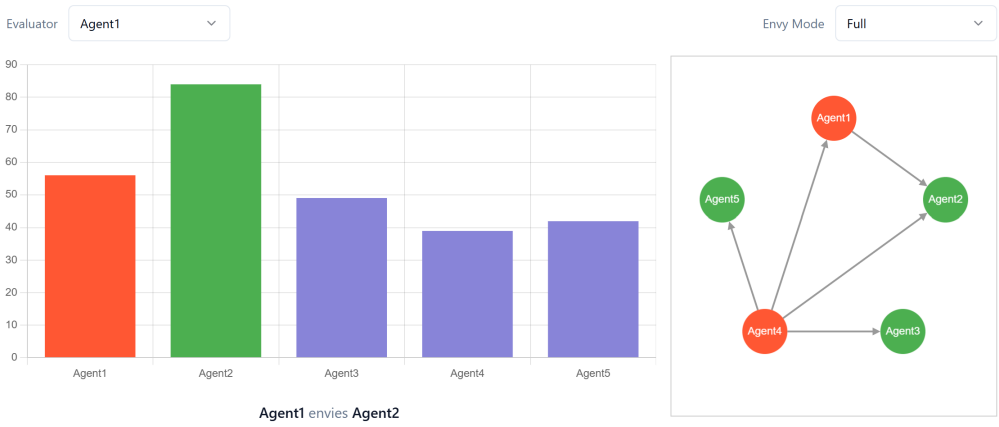

FIGURE 3.2 – Histogramme de la valeur des lots selon l'agent et le mode séléctionné

3.5 Algorithme

Afin d'illustrer concrètement les mécanismes de répartition équitable, nous avons intégré dans notre interface deux algorithmes classiques : le protocole de Lipton, basé sur l'élimination des cycles d'envie, et les *picking sequences*, qui organisent la répartition à travers une séquence ordonnée de choix.

Ces deux approches, bien que différentes dans leur logique et leur complexité, permettent de générer des répartitions que l'utilisateur peut analyser au travers des outils d'évaluation

développés. Nous détaillons ici la manière dont chacun de ces algorithmes a été implémenté et intégré dans l'interface.

3.5.1 Pool

Avant de pouvoir exécuter les algorithmes de répartition, il est nécessaire de définir un *pool* de biens, c'est-à-dire l'ensemble des objets à distribuer entre les agents. Ce pool est représenté dans notre application sous forme d'un `store` Svelte, qui suit la structure suivante :

— le pool est un objet associant à chaque type de bien (représenté par une couleur) une quantité entière ;
— les biens considérés sont : `red`, `green`, `blue`, `yellow`, et `purple`.

L'utilisateur peut initialiser manuellement ce pool via l'interface, ou bien générer automatiquement une configuration aléatoire.

Une fois le pool défini, il sert de point de départ pour les algorithmes de répartition. Ces derniers vont attribuer les biens disponibles aux agents selon les règles spécifiques de chaque protocole (Lipton ou picking sequence).

FIGURE 3.3 – Pool des biens à distribuer

3.5.2 Structures de données communes

1. **Preferences** Un mappage de l'identifiant de chaque agent vers une valeur numérique pour chaque objet. Autrement dit, pour chaque agent, on dispose d'une liste indiquant l'utilité qu'il retire de chaque objet.
2. **Allocation** Un mappage de chaque agent vers la liste des objets qu'il reçoit finalement.
3. **Envy Matrix** Une table bidimensionnelle dans laquelle l'entrée (i, j) enregistre à quel point l'agent i envie le lot de l'agent j.
4. **Utility Statistics** Un résumé incluant :
   — L'utilité totale de chaque agent pour son lot attribué.
   — Le bien-être social global (somme de toutes ces utilités).
   — L'identification de l'agent avec l'utilité la plus élevée et de celui avec l'utilité la plus faible.

3.5.3 Protocole de séquence de choix (picking sequences)

Aperçu

Il s'agit d'un protocole d'allocation séquentiel, au tour par tour, dans lequel les agents choisissent chacun à leur tour l'objet restant qu'ils préfèrent. L'ordre des tours (la "séquence") peut être généré de plusieurs manières, et l'allocation est toujours faite de manière gloutonne du point de vue de chaque agent.

Conversion des données du front-end en `Preferences`

Les données d'entrée des agents, généralement une liste d'enregistrements d'agents comprenant un nom et une carte d'utilité, sont converties en la structure interne `Preferences`. Cela standardise simplement la façon dont les valeurs d'utilité sont stockées.

Génération de la séquence de tours

Selon le style choisi, on construit une liste d'identifiants d'agents de longueur égale au nombre de choix :

- **Mirror (draft en « serpent »)** Les agents 1 à n choisissent dans l'ordre, puis n à 1, puis on répète si nécessaire. Cela équilibre l'avantage des premiers choix.
- **Random** Chaque choix est attribué à un agent tiré uniformément au hasard (avec une graine optionnelle pour la reproductibilité).
- **Repeated** Les agents choisissent dans l'ordre fixe de 1 à n, puis recommencent à 1 dès qu'ils atteignent n, jusqu'à ce que tous les choix soient attribués.

Réalisation de l'allocation

À partir d'une `Allocation` initialement vide et de l'ensemble complet des objets marqués comme « restants » :

1. Pour chaque tour de la séquence, identifier l'agent qui choisit.
2. Parmi les objets restants, sélectionner celui pour lequel cet agent a la plus grande utilité selon ses `Preferences`.
3. L'attribuer à l'agent, le retirer de l'ensemble des objets restants et l'ajouter à son lot.
4. Répéter jusqu'à ce qu'il n'y ait plus d'objets.

Évaluation des résultats

Une fois l'allocation terminée :

- Les `Utility Statistics` sont calculées en sommant, pour chaque agent, les utilités de son lot, puis en agrégant ces valeurs pour obtenir le bien-être social global et en repérant les extrêmes.
- L'`Envy Matrix` est remplie en comparant, pour chaque paire ordonnée (i, j), l'utilité de l'agent i pour le lot de l'agent j à celle de son propre lot ; toute différence positive est enregistrée comme envie. La valeur maximale de cette matrice est appelée « valeur d'envie ».

3.5.4 Algorithme d'élimination de cycles de Lipton

Objectif

Produire une allocation qui est sans envieux à l'exception d'au plus un objet (EF1) sous des utilités additives. Autrement dit, aucun agent ne doit envier un autre par plus de la valeur d'un seul objet.

Étapes clés

1. **Évaluation de lot** Calculer, pour tout agent et tout ensemble d'objets, l'utilité totale en additionnant les valeurs d'utilité de l'agent pour ces objets.
2. **Construction du graphe d'envie** Créer un graphe orienté dont les nœuds sont les agents. Tracer un arc de l'agent A vers l'agent B si A préfère strictement le lot actuel de B à son propre lot.
3. **Détection de cycles**
   — Rechercher d'abord des paires de mécontentement mutuel (cycles à deux nœuds).
   — Si aucun n'est trouvé, effectuer une recherche en profondeur pour découvrir des cycles plus longs.
4. **Rotation des lots le long d'un cycle** Étant donné un cycle d'agents $[A_1 \to A_2 \to \cdots \to A_k \to A_1]$, réaffecter de sorte que chaque agent reçoive le lot de son successeur dans le cycle. Cela améliore strictement le bien-être de tous les agents du cycle et supprime les arcs d'envie correspondants.

Boucle principale de l'allocation

1. **Initialisation** Commencer avec chaque agent détenant un lot vide. Attribuer arbitrairement un premier objet à un agent pour démarrer.
2. **Attribution séquentielle des objets** Pour chaque objet restant :
   — Éliminer tous les cycles d'envie existants en détectant et en faisant tourner les cycles jusqu'à ce que le graphe soit acyclique.
   — Identifier un agent « non envié », c'est-à-dire dont le degré d'entrée dans le graphe d'envie est nul.
   — Lui attribuer l'objet suivant, puis mettre à jour son lot.
3. **Élimination finale des cycles** Après que tous les objets ont été provisoirement attribués, effectuer une dernière détection de cycles et rotation pour garantir la condition EF1 dans l'allocation finale.

Contrôles de cohérence et résultats

Après avoir produit une allocation candidate, la procédure recalcule l'`Envy Matrix` et les `Utility Statistics` pour vérifier qu'aucune envie véritable ne subsiste. Si une entrée de l'`Envy Matrix` est encore positive, l'algorithme signale une erreur — bien que, sous utilités additives, cela ne devrait pas se produire si tout est correctement implémenté. Contrairement à la méthode de séquence de choix, il n'y a pas d'ordre explicite de choix à rapporter.

Notes comparatives

— *Picking sequence*
   — **Forces :** Conceptuellement simple, s'exécute rapidement en temps $O(n \cdot m^2)$.
   — **Faiblesses :** Peut générer une envie significative ; le résultat dépend fortement de l'ordre des tours.
— *Méthode de Lipton*
   — **Forces :** Garantit une allocation EF1 en temps polynomial $O(m \cdot (n + m))$.

— **Faiblesses :** Algorithmiquement plus complexe (construction de graphe, détection de cycles, rotations) et moins intuitive comme « draft ».

3.6 Limites et améliorations possibles

Plusieurs limites subsistent dans la version actuelle de notre explorateur : d'une part, la complexité algorithmique – notamment pour la détection et la rotation des cycles d'envie dans l'algorithme de Lipton – peut rapidement devenir prohibitive dès que le nombre d'agents ou de biens dépasse quelques dizaines, pénalisant la réactivité de l'interface. D'autre part, notre modèle d'utilités additives reste très simplifié : il ne prend pas en compte d'éventuelles interactions entre objets (effets de complémentarité ou de substituabilité), ni de critères non monétaires (temps, préférences qualitatives). Enfin, l'outil gagnerait à proposer davantage de scénarios préconfigurés et des aides à la prise en main pour un public non initié.

Pour lever ces verrous, plusieurs améliorations sont envisageables :

— **Optimisation algorithmique** : explorer des heuristiques ou des approches approximatives pour réduire drastiquement le temps de calcul sur de grandes instances.
— **Modèles d'utilité avancés** : intégrer des fonctions d'utilité non additives ou multicritères, voire des préférences floues.
— **Extension des protocoles** : ajouter des mécanismes de compensation monétaire ou de négociation bilatérale pour traiter les cas où l'équité pure est impossible.
— **Améliorations UX** : enrichir la bibliothèque de scénarios, proposer un mode tutoriel interactif et optimiser l'accessibilité (responsive design, support d'aides visuelles).
— **Validation sur données réelles** : collaborer avec des institutions pour tester l'outil sur des cas pratiques (attribution de logements, partage d'héritages), afin d'ajuster les algorithmes et les visualisations aux besoins concrets.

Chapitre 4

Conclusion

En conclusion, ce projet a permis de développer une interface interactive et performante pour l'exploration des notions fondamentales du partage équitable, en intégrant deux protocoles classiques (Lipton et picking sequences) et en mettant l'accent sur l'absence d'envie et ses relaxations (EF1, EFX). Le choix de SvelteKit s'est avéré particulièrement adapté, offrant un rendu rapide, une liaison réactive des données et une navigation fluide même sur de grands ensembles de biens et d'agents. Toutefois, la complexité algorithmique de la détection et de la rotation des cycles d'envie demeure un point critique à optimiser pour des instances de très grande taille. À l'avenir, il serait intéressant d'explorer des heuristiques plus efficaces, d'ajouter de nouveaux critères d'équité et d'étendre l'outil à des contextes mutli-critères ou dynamiques.

Bibliographie

[1] Sylvain Bouveret, Yann Chevaleyre et Nicolas Maudet. *Fair Allocation of Indivisible Goods*. In *Handbook of Computational Social Choice*, Cambridge University Press, 2016. Disponible : https://recherche.noiraudes.net/resources/papers/comsoc-chapter12.pdf

Annexe A

Cahier des charges

Introduction et Contexte

L'allocation équitable des biens indivisibles est une problématique centrale en économie et en théorie des jeux. Elle concerne la répartition de ressources entre plusieurs agents sans possibilité de division. Ce sujet trouve des applications dans divers domaines, notamment le partage d'héritages, l'attribution de logements sociaux ou encore la distribution de tâches dans des systèmes multi-agents.

Dans de nombreux contextes, les biens à partager ne peuvent être divisés sans perte de valeur. Par exemple, un logement ne peut être scindé entre plusieurs bénéficiaires sans compromettre son utilité. De même, dans un cadre organisationnel, la répartition de projets ou de tâches entre collaborateurs doit respecter certaines contraintes tout en garantissant une certaine justice perçue. L'étude de ces problèmes vise à identifier des méthodes garantissant une allocation perçue comme juste, efficace et réalisable.

Les travaux en allocation équitable s'inspirent souvent de concepts mathématiques et informatiques, notamment l'optimisation combinatoire, les algorithmes de répartition et la théorie de la décision. L'objectif est de trouver des méthodes d'allocation qui respectent différents critères d'équité et maximisent le bien-être global des agents concernés.

Problématique

Comment concevoir un mécanisme permettant d'allouer des biens indivisibles de manière équitable entre plusieurs agents ayant des préférences différentes ? Cette problématique soulève plusieurs questions, telles que la définition de l'équité, la prise en compte des préférences des agents et l'optimisation des allocations en fonction de critères justes et efficaces.

Besoins

1. Développer un modèle algorithmique (utilisant le langage Python) permettant de simuler l'allocation des biens.

2. Intégrer différents critères d'équité, tels que l'équité proportionnelle, l'absence d'envie (envy-freeness) et l'optimisation du bien-être social.
3. Concevoir une interface ou une simulation permettant de visualiser les résultats de l'allocation (initialement prévu en Idyll).
4. Proposer des scénarios d'application concrets pour valider le modèle.

Objectifs

— Concevoir une méthode d'allocation garantissant une répartition la plus équitable possible.
— Comparer plusieurs algorithmes d'allocation pour identifier les plus performants.
— Expérimenter et analyser les résultats sur des cas réels ou simulés.
— Rédiger un rapport détaillé (dans une interface interactive et dynamique) sur les méthodes utilisées et les conclusions obtenues.

Contraintes

— Respecter des critères d'équité bien définis (e.g. équité proportionnelle, absence d'envie, etc.).
— Prendre en compte des préférences hétérogènes parmi les agents.
— Assurer une complexité algorithmique raisonnable pour garantir l'applicabilité du modèle.
— Adapter la solution aux cas où le nombre de biens est inférieur au nombre d'agents.
— Proposer une implémentation testable avec des jeux de données pertinents.

Annexe B

Manuel utilisateur

Fair Division est un projet SvelteKit conçu pour implémenter une application de *partage équitable*. L'objectif de ce projet est de proposer une application web moderne, rapide et évolutive pour résoudre des problèmes de répartition équitable de ressources ou de tâches entre plusieurs participants.

B.1 Fonctionnalités

— Développé avec SvelteKit, un framework pour construire des applications web.
— Prise en charge du rendu côté serveur (SSR), de la génération statique (SSG) et du rendu côté client (CSR).
— Haute performance et optimisation pour le développement web moderne.

B.2 Prise en main

B.2.1 Pré-requis

— Node.js (version 16 ou supérieure recommandée)
— npm ou pnpm

B.2.2 Installation

1. Cloner le dépôt :

```
git clone <repository-url>
cd <repository-name>
```

2. Se placer dans le répertoire du projet SvelteKit :

```
cd fair-division-app
```

3. Installer les dépendances :

```
npm install
```

B.2.3 Développement

Démarrer le serveur de développement :

```
npm run dev
```

Ouvrez ensuite `http://localhost:5173` dans votre navigateur.

B.2.4 Compilation pour la production

Pour construire l'application en mode production :

```
npm run build
```

Les fichiers compilés seront générés dans le répertoire `build`.

B.2.5 Aperçu du build de production

Pour prévisualiser localement le build de production :

```
npm run preview
```

B.3 Pour en savoir plus

- Documentation SvelteKit
- Documentation Svelte

Auteurs

Ce projet a été réalisé par Baran Açikel, Kaan Dişli et Guillaume Lebreton.