Projet : Théorie des Graphes

Julien Peffer, Alexandre Bourguignon

Décembre 2023

1 Introduction

Ce document vise à synthétiser notre projet en Théorie des Graphes qui a pour but, à partir d'un fichier qui contient les 20 decks les plus joués par les top players de Clash Royale, de créer un graphe qui à chaque sommet associe une carte (voir 2.1 pour plus d'informations). L'objectif était alors d'analyser ce graphe afin d'en tirer le plus d'informations pratiques sur le jeu. Nous avons donc poussé l'analyse en cherchant à savoir quelles étaient les cartes les plus populaires, les paires de cartes les plus jouées ensemble, les cartes les moins jouées ensemble, la possibilité d'afficher les cartes uniquement d'un certain type (pour rappel il y a 4 types de cartes à savoir les troupes aériennes, les troupes terrestres, les bâtiments et les sorts), etc. Nous allons donc présenter synthétiquement chaque fonction et points clés par lesquels nous sommes passés.

2 Affichage d'un graphe

Dans cette section nous verrons quelles sont les fonctions utilisées pour créer un graphe et l'afficher. Les fonctions en question sont $creer\_graphe\_type()$ et $afficher\_graphe()$

2.1 $creer\_graphe()$

- But : Cette fonction crée un graphe non orienté pondéré basé sur le fichier texte.
- Input : Un fichier texte et un type de carte : TrAerienne (Troupes aériennes), TrTerrestres (Troupes terrestres), Sorts ou Batiments. (NB : si le type n'est pas spécifié alors la fonction prend en compte toutes les cartes).
- Output : Le graphe contenant les sommets du type spécifié ou toutes les cartes dans le cas échéant.
- Explication : Le fichier contient sur chaque ligne un deck avec 8 cartes, chacune suivie par son type. Cette fonction parcourt donc toutes les cartes

en fonction du type ou non et crée des sommets pour chaque carte. Elle construit ensuite une arête entre les sommets si ces deux sommets sont jouées dans le même deck. De plus, si deux cartes sont jouées dans le même deck, alors une arête avec un poids de 1 relient ces deux sommets. Si ces deux cartes sont joués ensemble dans N decks, alors l'arête entre ces sommets a un poids de N. Le choix d'un graphe non orienté est trivial : supposons qu'on ait 2 cartes A et B qui soient dans le même deck, la carte A est jouée avec la B et la carte B est jouée avec la A, cela revient donc à créer deux arcs de A vers B et de B vers A, donc au final de créer une arête entre A et B.

2.2 $affiche r\_graphe()$

- But : Cette fonction affiche le graphe donné en paramètre.
- Input : Le graphe à afficher.
- Output : Une fenêtre s'ouvre et le graphe est affiché.
- Explication : Nous avons opté pour un graphe circulaire afin qu'il soit plus lisible.

3 Options/Fonctionnalités

3.1 Afficher les cartes les plus jouées avec $cartes\_les\_plus\_jouees()$

- But : Cette fonction a pour but d'afficher les 3 cartes les plus jouées.
- Input : Un graphe créé préalablement.
- Output : Le top 3 des cartes les plus jouées.
- Explication : Afin de trouver les cartes les plus jouées il suffit de calculer la somme des pondérations des arêtes incidentes de chaque sommet. La carte possédant la somme des pondérations la plus haute étant la carte la plus jouée. Cependant, nous ne possédons que la somme des pondérations des arêtes incidentes et non de l'occurence de chaque carte.Afin de donner l'occurence de chaque carte, nous avons utilisé la formule :

  nombreOcurrences = sommePonderation / (nbElementParLigne - 1)

  Etant donné que nous avons des decks de 8 cartes, nous savons que dans une ligne chaque carte fait partie d'un couple de carte avec (nbElementParLigne-1) = 7 cartes et qu'une carte se retrouve au maximum une fois dans un deck. Soit k $\in \mathbb{N}_0$, sommePonderation = 7k et lorsque l'on divise sommePonderation par (nbElementParLigne-1), nous trouverons l'occurence exacte

3.2 Afficher les couples de cartes les plus joués avec *couple_cartes_les_plus_joues*()

- But : Cette fonction a pour but d'afficher les 3 couples de cartes les plus jouées.
- Input : Un graphe.
- Output : Le top 3 des couples cartes les plus jouées ainsi que leur nombre d'occurrences est affiché.
- Explication : Afin de trouver les couples de cartes les plus joués, nous avons trié chaque arête du graphe par pondération croissante et nous en avons retiré le top 3. Etant donné que chaque couple de cartes est relié par une arête différente, les sommets liés à l'arête ayant la pondération la plus élevée se verront attitrés le rôle de couple de carte le plus joué et ainsi de suite jusqu'au top 3. De plus, cette option peut être utilisée sur un type de carte spécifique, on obtiendra alors à l'issue de cette fonction le top 3 des couples de cartes les plus joués uniquement basé sur un graphe avec les cartes du type spécifié.

3.3 Affiche les couples de cartes jamais joués ensemble avec *couple_cartes_jamais_joues_ensemble*()

- But : Cette fonction a pour but d'afficher les couples de cartes jamais joués ensemble
- Input : Un graphe.
- Output : Les couples de cartes jamais joués ensemble.
- Explication : L'objectif est de vérifier si dans tous les couples de cartes possibles donné à partir du graphe en input, il y a une arête existante entre ce couple de carte. Si il n'y a pas d'arête existante, nous avons trouvé une un couple de carte jamais joué ensemble.

3.4 Affiche le nombre d'Erdos pour la carte la plus jouée avec *nombre_erdos_carte_plus_jouee*()

Le nombre d'Erdos dans ce projet sera la carte la plus jouée. Toutes les comparaisons se feront avec cette carte source.

- But : Cette fonction a pour but d'afficher le nombre d'Erdos par rapport a toutes les cartes du graphe.
- Input : Un graphe.
- Output : Le nombre d'Erdos de chaque carte du graphe.

- Explication : En ayant la carte la plus jouée en possession, nous pouvons déterminer le nombre d'Erdos de chaque carte par rapport à la carte la plus jouée.Afin de réaliser cette fonctionnalité, nous utilisons la fonction $nx.shortest\_path\_length()$ qui prend en paramètre le graphe, la carte source, et la carte cible représentant toutes les cartes du graphe. Ainsi, la fonction donnée par la bibliothèque networkx fait la recherche du plus court chemin en utilisant sûrement l'algorithme de Djikstra. NB: Nous verrons par la suite que cette fonction n'est pas crée en vain, elle pourra nous servir pour déterminer quelles sont les cartes les moins jouées dans le graphe.

3.5 Affiche certaines cartes les moins jouées grâce au nombre d'Erdos avec *certaines_cartes_moins_jouees*()

Vous remarquerez que nous avons indiqué certaines et non toutes car il y a des exceptions.

- But : Cette fonction a pour but d'afficher certaines cartes les moins jouées.
- Input : La fonction aura besoin du graphe et du nombre d'Erdos pour chaque carte.
- Output : Certaines cartes les moins jouées.
- Explication : Si le nombre d'Erdos d'une carte est grand par rapport à la carte la plus jouée, alors elle a plus de chance de ne pas se retrouver dans les decks donc avoir moins d'occurences qu'une autre carte. En prenant également en compte le degré de chaque sommet, on trouve une formule qui nous permet de connaitre les cartes les moins jouées à une exception près :

  nbr_dErdos_normalise = nombre_dErdos / (connexions_totales[carte] + 1)

  La normalisation du nombre d'Erdos par le nombre total de connexions est une façon de mettre en balance ces deux facteurs. Cela permet de s'assurer que les cartes moins jouées ne sont pas simplement celles qui sont éloignées de la carte la plus jouée, mais aussi celles qui ont une faible connexité globale dans le graphe.
  Cette fonction nous donne presque toutes les cartes les moins jouées à l'exception d'une, celle qui se trouve dans le deck de la carte la plus jouée elle à d'office un nombre d'Erdos = 1 même si elle n'est jouée qu'une seule fois dans le graphe.

3.6 Afficher les cartes les moins jouées manquantes à la fonction précédentes avec *cartes_moins_jouees_manquantes*

On rajoute la/les cartes les moins jouées qui ont été manquées par la fonction précédente.

- But : Cette fonction a pour but d'afficher les cartes les moins jouées manquantes.
- Input : La fonction aura besoin du graphe.
- Output : les cartes moins jouées manquante.
- Explication : Nous prenons les cartes dont les sommes des pondérations des arêtes sont les plus basses ce qui nous donne les cartes les moins jouées. Ensuite, nous faisons une soustraction d'ensemble i.e, nous prenons les éléments que la fonction *certaine_carte_moins_joue*() renvoie et ceux de la fonction décrite ici, cela nous aidera à déterminer les cartes moins jouées manquantes.

4 Difficulté rencontrée/Idée de fonctionnalité

4.1 Déterminer le top 3 des trios de cartes les plus jouées

Ayant trouvé le top 3 des cartes les plus jouées et le top 3 des couples de carte les plus joués, nous avons naturellement cherché à créer une fonction qui renvoie le top 3 des trios de cartes les plus jouées. Cependant n'ayant pas d'idée de comment opérer, nous avons donc testé au cas par cas plusieurs méthodes. Nous avons alors réfléchi par rapport au graphe en se demandant quelles sont les conditions pour que 3 cartes soient jouées dans le même deck et nous en avons conclu que il fallait trouver toutes les cliques de taille 3. Il ne restait plus qu'à trouver une condition sur les pondérations. En supposant que la bonne condition soit que toutes les pondérations des arêtes d'une clique doivent être égales, nous n'avions pas de résultat satisfaisant (nous avions vérifiés au préalable dans le fichier txt quel était le trio de carte le plus joué et le résultat affiché n'était pas le bon).
Après réflexion, nous avons conclu qu'il n'est pas possible de trouver cette information étant donné qu'il n'y a pas de conditions sur les poids des arêtes : un trio de carte peut très bien être le plus joué mais avoir des pondérations toutes différentes ou bien toutes égales. Pour trouver le trio de carte le plus joué il faudrait pouvoir distinguer quelle carte est dans quel deck et par conséquent il faudrait rajouter une information de plus dans le fichier texte, or ce ne serait plus une analyse sur graphe mais une analyse du fichier texte.

4.2 Chercher les cartes les plus jouées d'un type spécifique

Après avoir ajouté la fonction qui permet d'afficher les cartes d'un certain type, nous avons cherché à savoir comment trouver les cartes les plus jouées pour un type spécifique. Cependant, reprendre la même formule de base n'était pas suffisant car en testant avec un type, le nombre d'occurrences de la carte la plus jouée est un nombre à virgule... Ceci vient du fait que diviser par nbElementParLigne - 1 n'a de sens que quand on prend tout le graphe en compte. N'ayant plus de notion de deck dans ce cas, trouver les cartes les plus jouées par

type n'était pas possible. De surcroît, puisque aucune carte du type Batiments n'est jouée plusieurs fois dans un même deck, le graphe associé ne contient aucune arête (or il existe bien une carte Batiments qui est la plue jouée!). Il est donc impossible de trouver le nombre d'occurrence d'une carte sans arêtes.

4.3 Déterminer si le graphe est hamiltonien et/ou eulérien

Contrairement aux points 4.1 et 4.2, il a été possible de trouver si un graphe était hamiltonien et/ou eulérien. Cependant après réflexion nous avons pas trouvé de lien à faire entre ceci et le jeu. Autrement dit savoir si le graphe est hamiltonien et/ou eulérien n'apportait pas d'information particulière sur comment construire un deck, quelles sont les meilleures cartes,...
Puisque les fonctions marchent tout de même nous avons quand même décider de les laisser, même si elles n'ont pas d'utilité.

5 Source

Nous avons utilisé un bon nombre de fonctions proposée par la bibliothèque networkx.
Le site qui nous a particulièrement aidé est celui-ci :
https://networkx.org/documentation/stable/reference/index.html
Il a été d'une grande aide afin d'implementer à peu près toutes les fonctions.