Projet de programmation :

Simulation de l'évolution d'un réseau social et de la propagation de l'information au sein de ce réseau

BORDES Martin - ROSUNEE Avichaï

25 Mai 2021

Xavier DUPRÉ

1 Introduction

Un réseau social évolue et grandit au cours du temps avec la formation de nouvelles connexions et relations entre les utilisateurs. Au cours des dernières années nous avons assisté à une croissance massive des réseaux sociaux en ligne comme Facebook, Twitter, etc. Il est donc devenu capital de connaître et comprendre le destin de ces réseaux. Ainsi, prédire l'évolution d'un réseau social est aujourd'hui une question d'une extrême importance. Un bon modèle d'évolution d'un réseau social peut aider à comprendre les propriétés responsables des changements qui se produisent dans la structure d'un réseau. Nous avons donc décidé de coder un réseau social afin d'observer son évolution et comment l'information peut se propager au sein de celui-ci.

2 Synthèse

Notre premier objectif était de modéliser l'évolution d'un réseau social. Nous avons commencé par simuler le cas d'un réseau social du même type que Facebook, c'est-à-dire où les relations, appelées amitiés, sont à double sens. Pour cela, nous avons utilisé une structure de graphe. Chaque nœud correspond à un internaute et chaque arête à une amitié. On crée une liste de facteurs, qui pourraient par exemple être le lieu de résidence, qui ont tous une importance propre. À chaque internaute on associe certains facteurs. On crée d'abord un graphe de départ, avec des amitiés choisies aléatoirement. Pour les règles d'évolution, nous nous sommes inspirés de l'article "*Evolving Social Networks via Friend recommendations*" de A.K Verma et M.Pal. Dans ce modèle, deux personnes doivent atteindre un certain score à partir de leurs amis communs et de leurs différents facteurs communs pour qu'il y ait une recommandation d'amis entre eux. Ils ont alors une certaine probabilité de le devenir. Si ce n'est pas le cas, cette recommandation ne sera plus proposée. Les deux variables à définir sont donc le score à dépasser, le seuil, et la probabilité qu'une amitié soit créée lorsqu'elle est recommandée. La simulation s'arrête lorsqu'il n'y a plus d'évolution. Le graphe final est alors un réseau social simulé.

Ensuite, nous avons voulu modéliser un réseau social du même type que Twitter, où les relations, appelées follows, sont à sens unique. Nous avons gardé une structure de graphe, en utilisant cette fois une matrice d'adjacence à la place d'une liste de tuple. On utilise toujours la notion de facteur. En revanche, les règles d'évolution sont bien différentes. Elles ne se basent plus sur des recommandations. On définit la force de la possible relation d'une personne envers une autre en fonction de leurs facteurs communs et du nombre de followers de la seconde. À chaque tour, en fonction d'une probabilité calculée à partir de cette force, la première personne peut suivre la seconde, ne pas la suivre ou choisir de ne jamais la suivre.

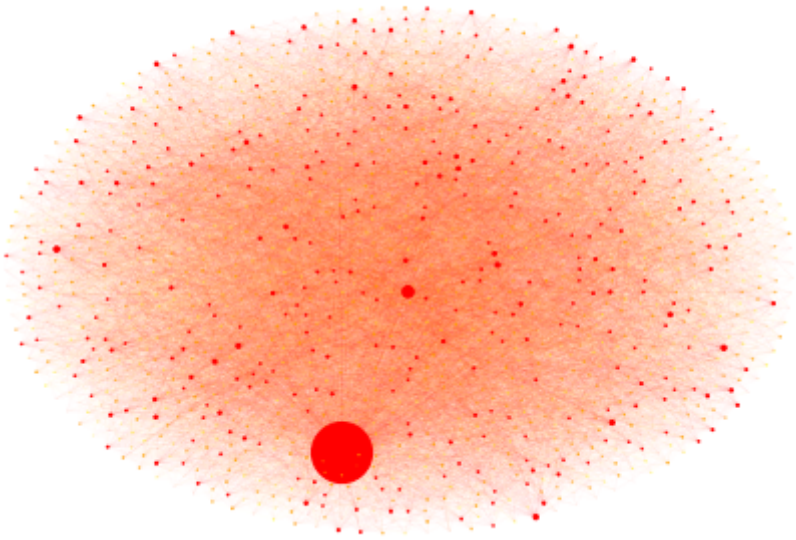
Nous avons codé ces règles d'évolution car elles nous paraissaient plus intuitives que la fonction assez complexe utilisée dans l'article qui a inspiré le modèle précédent. On arrête également la simulation lorsque le graphe n'évolue plus. On peut alors comparer les différences entre ces deux modèles.

La complexité étant assez importante, il a fallu se restreindre à un nombre d'internautes assez faible. Nous avons évalué jusqu'à combien d'internautes un ordinateur pouvait effectuer la simulation dans un temps raisonnable.

À partir du graphe final modélisant un réseau social du type de Twitter, nous avons étudié la propagation d'un tweet. Pour cela, il a fallu associer à chaque internaute une probabilité de retweeter. Nous faisons entrer les likes et les commentaires dans la catégorie du retweet puisqu'ils ont à peu près le même impact. Nous avons regardé la corrélation entre le nombre de personnes atteintes et le nombre de followers.

Puis, nous avons voulu déterminer s'il existait des communautés dans ce réseau social. Pour vérifier la valeur du calcul des communautés, nous avons regardé si les tweets d'un internaute touchaient plus sa communauté et si les personnes appartenant à une plus grande communauté touchaient plus de monde.

Figure 1 - Graphe final du modèle Twitter pour 1767 internautes



Enfin, nous voulions vérifier la pertinence de notre modèle en le confrontant aux vraies données de Twitter. Il nous fallait alors un moyen de récupérer l'ensemble des followers d'un compte Twitter. Nous avons donc créé un compte développeur Twitter afin d'obtenir les *Consumer Keys* et *Authentication Tokens* nécessaires à l'utilisation de l'API Twitter. Via Python, nous avons utilisé la librairie *Tweepy* qui permet d'accéder assez aisément à l'API de Twitter.

Pour des raisons explicitées dans la section suivante (3 Difficultés), nous avons décidé de partir du compte Twitter de l'ENSAE (https://twitter.com/ENSAEparis) qui compte 1767 followers à ce jour.

Nous avons donc, dans un premier temps, récupéré les ID Twitter de ces 1767 followers afin de créer un *dataframe* à deux colonnes (*source* = ID Twitter de l'ENSAE = 919247378196107266 et *target* = ID Twitter de chaque follower du compte Twitter de l'ENSAE).

Puis, afin d'obtenir ce que l'on peut appeler un "réseau social" nous avons bouclé pour obtenir les followers de chacun de ces 1767 followers de l'ENSAE. Nous obtenons donc un véritable "réseau de followers" sous la forme d'un *dataframe* qui a la même structure que précédemment mais où la *source* peut cette fois être n'importe quel follower du compte Twitter de l'ENSAE. Ce "réseau de followers" n'est cependant pas satisfaisant dans la mesure où il fait intervenir des comptes qui ne followent pas l'ENSAE de base. Nous avons donc limité notre réseau pour ne garder que les lignes du *dataframe* où *target* est un follower de l'ENSAE afin d'obtenir un "réseau social" où chaque noeud est un follower de l'ENSAE et donc de voir comment les relations se créent entre personnes suivant de base un même compte.

3 Difficultés

Complexité temporelle. Le temps d'exécution des algorithmes d'évolution du réseau social est assez long dès qu'on augmente le nombre d'internautes. De plus, il est difficile d'avoir une idée de la complexité et de ce temps d'exécution car les règles d'évolution utilisent de l'aléatoire et car l'arrêt dépend d'une condition. On ne peut donc pas connaître l'avancement de la simulation. Nous avons alors utilisé une méthode de *code profiling* pour comprendre ce qui nous coûtait le plus de temps et ainsi optimiser certaines parties de notre code.

Limitations de L'API twitter. L'API de Twitter permet de faire de nombreuses opérations et de récupérer des informations variées que ce soit sur un tweet ou sur un compte Twitter. Malheureusement la version *Standard* (gratuite) est assez limitée en termes de requêtes. Pour ce qui nous intéresse c'est-à-dire, la récupération de followers *(Follow lookup)* nous sommes limités à 15 requêtes / 15mins.
Ainsi pour récupérer les followers des followers du compte Twitter de l'ENSAE, le programme a dû tourner plus de 36h ! C'est pour cela que nous avons choisi le compte de l'ENSAE; il n'a pas trop de followers ce qui permet de récupérer les followers assez rapidement et en même temps, il a assez de followers pour en tirer des informations assez satisfaisantes.

4 Résultats

Tout d'abord, nous voulions étudier les différences entre deux modèles simulés s'inspirant respectivement des fonctionnements de Facebook et Twitter.

Figure 2 - Distribution du degré des noeuds du modèle Facebook pour 150 internautes

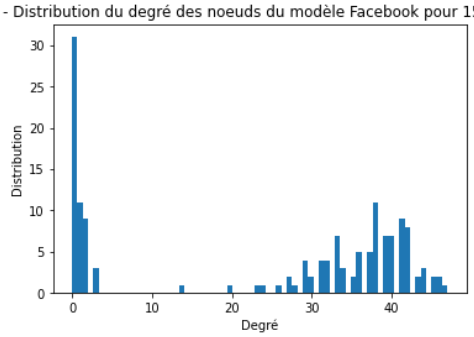
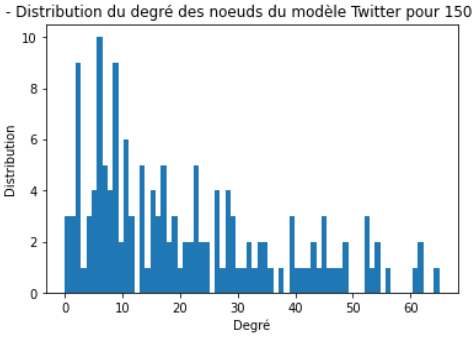
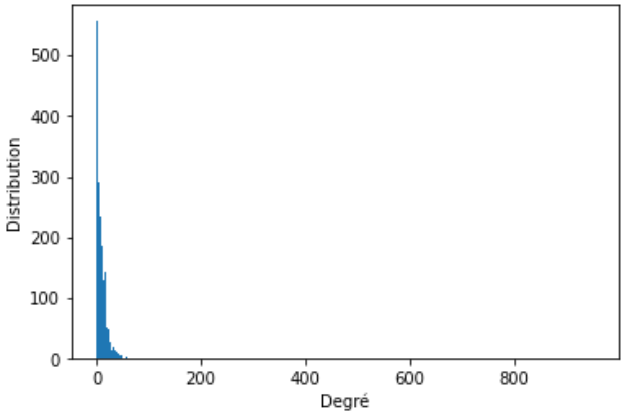


Figure 3 - Distribution du degré des noeuds du modèle Twitter pour 150 internautes



Alors que la moyenne des degrés est légèrement supérieure pour le modèle Facebook (23 contre 20), les évènements extrêmes y sont plus rares. Les gens ont à peu près tous le même nombre d'amis si l'on exclut les personnes qui ont très peu voire pas d'amis, à l'écart du réseau. Pour le modèle Twitter, on observe au contraire une grande diversité du nombre de followers.

Figure 4 - Distribution du degré des noeuds du modèle Twitter pour 1767 internautes



Lorsqu'on augmente le nombre d'internautes, on observe des évènements encore plus extrêmes. Ici, un internaute a presque 1000 followers alors que la moyenne est de 10.

Nous nous sommes alors posé la question des limites de calcul de nos ordinateurs. Pour le modèle Twitter, le meilleur de nos deux ordinateurs a pu simuler l'évolution d'un réseau social composé de 20 000 internautes en 35h. Nous avons décidé d'en prendre beaucoup moins car nous étions limités par la récolte des données de Twitter et car il nous était nécessaire de lancer un grand nombre de fois la simulation pour l'ajuster.

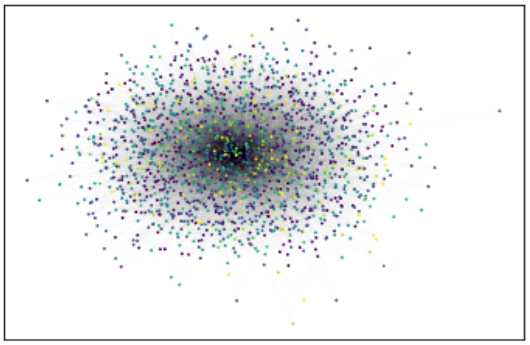
Ensuite, nous avons étudié le parcours de tweets. Après avoir fait tweeter 10 fois chaque internautes, on observe que les résultats suivants :

```
La moyenne du nombre de personnes atteintes sur tous les tweets est de 57.554916
L'écart-type des moyennes du nombre de personnes atteintes est de 70.92089
La moyenne des écart-types du nombre de personnes atteintes est de 86.36815
```

Il y a une grande variabilité entre les internautes et entre les tweets d'un même internaute. La variabilité entre les internautes peut-être expliquée par la diversité de degré entre les internautes, ce facteur étant assez explicatif puisque le facteur de corrélation entre ce dernier et le nombre de personnes atteintes par un tweet est de 0.75.

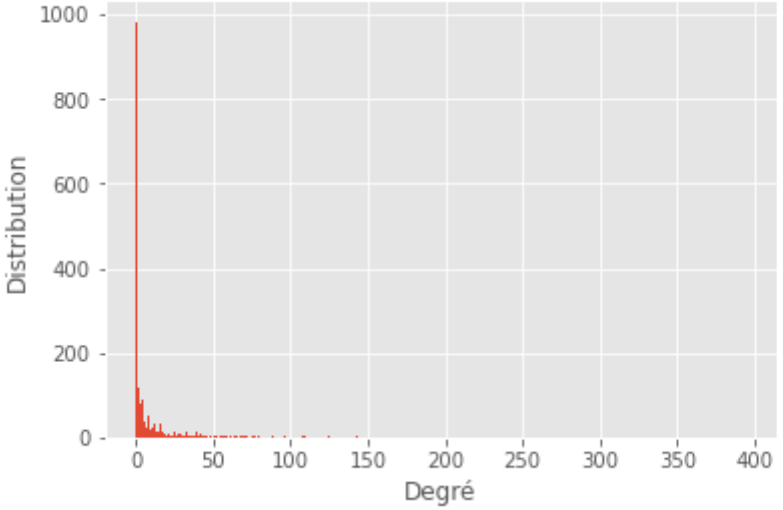
Nous avons par la suite calculé les communautés présentes dans le graphe final. Les données sur les parcours de tweets ont permis de montrer la pertinence de ces communautés. En effet, en moyenne, alors que 12.9% du réseau social fait partie de la communauté de l'internaute étudié, 33.8% des personnes qu'il atteint en font partie. Cependant, on observe une corrélation négative (-0.54) entre la taille d'une communauté et le nombre de personnes atteintes par les tweets de ses membres. Cela peut être dû au fait que les comptes très suivis ne font pas partie de grandes communautés.

Figure 5 - Graphe final décrivant les communautés du modèle Twitter avec 1767 internautes



Enfin, l'objectif en récupérant les données du compte Twitter de l'ENSAE était de comparer le réseau formé par ses followers avec celui construit avec notre modèle.

Figure 6 - Distribution du degré des noeuds des followers de l'ENSAE



La distribution du degré des nœuds est proche de celle de notre modèle. On observe également des évènements très extrêmes. En ce qui concerne la simulation de tweets, les résultats sont moins proches. Les vrais internautes atteindraient plus de personnes selon notre simulation de parcours. La variabilité entre eux est encore plus forte mais la variabilité entre les tweets d'une même personne est moins élevée.

```
La moyenne du nombre de personnes atteintes sur tous les tweets est de 122.80575
L'écart-type des moyennes du nombre de personnes atteintes est de 181.73358
La moyenne des écart-types du nombre de personnes atteintes est de 116.5485
```

5 Conclusion

Nos deux types de modèles ont donc bien montré des différences, au niveau de la répartition des arêtes des graphes, des relations. Notre modèle Twitter s'est révélé assez proche de la réalité observée sur le réseau social et nous a permis de mettre en avant des propriétés concernant la propagation de l'information.

Cependant, il est encore améliorable, notamment en simulant beaucoup plus d'internautes. Un modèle poussé à la perfection serait d'une grande aide pour les entreprises gérant les réseaux sociaux. Cela pourrait, par exemple, leur permettre de prédire l'impact d'une nouvelle fonctionnalité.

A Instructions d'exécution

Le code est fourni sous la forme d'un notebook Python disponible sur Github via le lien suivant : https://github.com/AV8-hub/chicken837

Dans ce même répertoire sont disponibles tous les autres fichiers qui pourraient vous être utiles.

Soyez attentif aux ⚠️ qui indiquent une remarque importante dans le code.

B Point de design du code

Utilisation de l'API Twitter via Tweepy et récupération des données

Pour utiliser l'API de Twitter via Python, nous avons besoin de la librairie Tweepy.

```
import tweepy
```

On commence par fournir nos identifiants de l'API Twitter disponibles sur la page développeur de Twitter.

```
consumer_key = 'XXXXXXXXXXXX'
consumer_secret = 'XXXXXXXXXXXXXXXXXXXXXXX'
access_token = 'XXXXXXXXXXXXXXXXXXXXXXX'
access_token_secret = 'XXXXXXXXXXXXXXXXXXXXXXX'
```

Les commandes suivantes permettent de s'identifier dans tweepy avec nos identifiants développeur

```
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)
```

Comme nous allons télécharger des ensembles de données assez volumineux, il est important de spécifier certains paramètres lorsque nous initialisons l'API. Nous définissons les paramètres "wait_on_rate_limit" et "wait_on_rate_limit_notify" à *True*.

```
api = tweepy.API(auth, wait_on_rate_limit=True,
wait_on_rate_limit_notify=True, compression=True)
```

Comme vu précédemment, il existe des limites de débit lors du téléchargement de données à partir de Twitter. En définissant ces paramètres sur *True*, nous ne romprons pas la connexion à l'API lorsque ces limites seront atteintes, nous attendons simplement que le délai d'attente se termine et nous pourrons continuer à télécharger des données.

On commence par récupérer l'ID twitter de l'ENSAE à partir du nom d'utilisateur (*screen name*)

```
ensae = api.get_user(screen_name = "ENSAEparis")
```

`ensae.id`
L'ID Twitter de l'ENSAE est : 919247378196107266
L'objectif maintenant est de récupérer les ID de l'ensemble des followers de l'ENSAE

La "pagination" est très utilisée dans le développement de l'API Twitter en itérant selon des timelines, des listes d'utilisateurs ou encore des messages.
La commande : `for page in tweepy.Cursor(api.followers_ids, user_id=user).pages():` permet de fournir un paramètre de curseur avec chacune de nos requêtes et ainsi de récupérer les ID des followers du compte user en itérant sur l'ensemble des pages.
`followers.extend(page)` permet d'ajouter tous les éléments de *pages* à la fin de la liste *followers*
On utilise une syntaxe `try | except` pour se prémunir d'une potentielle erreur.
L'instruction `TweepError` est l'exception principale utilisée par tweepy

C Calcul du coût

Il n'est pas possible d'estimer facilement la complexité de l'entièreté d'une simulation de l'évolution d'un réseau social, que ce soit pour un du type de Facebook ou de Twitter. En effet, on ne peut pas connaître le nombre de passages dans la boucle `while`.

Ainsi, nous avons choisi d'étudier la fonction `passage` pour le calcul de la complexité. On notera N le nombre d'internautes et n le nombre de facteurs. On se place dans le pire des cas, c'est-à-dire que personne ne se follow mais tout le monde le peut encore.

C.1 Fonctions hors des boucles

Toutes les fonctions hors de la boucle sont appelées une fois.

On commence par effectuer une `copy.deepcopy()` du graphe donné en entrée. On copie la liste des internautes et la matrice d'adjacence donc la complexité de cet appel est en $\Theta(N^2)$.

Ensuite, on calcule le degré de chaque nœud grâce à `poids_noeuds`. Dans cette fonction, on appelle une fois `colonne` pour chaque internaute. Cette fonction cherche les followers d'un internaute. Pour cela, elle effectue une boucle sur tous les internautes donc en $\Theta(N)$. Ainsi, la complexité de `poids_noeuds` est en $\Theta(N^2)$.

Enfin, on appelle la fonction `zero`. Elle contient deux boucles sur le nombre d'internautes donc sa complexité est en $\Theta(N^2)$.

Ainsi, la complexité de ce qui se trouve hors des boucles est en $\Theta(N^2)$.

C.2 Fonctions au sein des boucles

Comme dit précédemment, on considère le pire des cas. La boucle s'effectue alors sur toutes les relations, au nombre de N^2.

Dans la première boucle, on appelle une fois la fonction `force2`. Cette dernière utilise le degré déjà calculé avant la boucle et la fonction `valeur2`. En son sein, on appelle une fois la fonction `facteurs_communs2`. Cette dernière parcourt la liste des facteurs des deux internautes de la relation que l'on regarde dans ce passage de la boucle. Ainsi, elle possède une complexité en $\Theta(n^2)$. Ainsi, la complexité de la première boucle est en $\Theta(n^2)$.

La deuxième boucle appelle également `force2` une fois donc elle possède aussi une complexité en $\Theta(n^2)$.

Ainsi, la complexité totale des deux boucles est en $\Theta(N^2n^2)$.

En conclusion, l'ensemble de la fonction `passage` est caractérisé par une complexité en $\Theta(N^2) + \Theta(N^2n^2) = \Theta(N^2n^2)$ dans le pire des cas.