INFO-F106 : Projet d'année
Rappport intermédiaire

Mardi 16 décembre 2014

Carlos Requena López - *410031*

Table des matières

1 Introduction

Ce projet s'attaque à la simulation de la propagation d'une rumeur dans un réseau social. Une certaine personne commence à diffuser une rumeur dans un premier temps, étant limité par la relation avec les autres gens du réseau. Après une valeur discrète de tours (unités de temps), la rumeur aura voyagé et peut être changé.

Il est réalisé avec PYTHON 3. Comportant quatre parties, chacune devrait ajouter quelques fonctionnalités à la précédente.

2 Réalisation

2.1 Première partie

Dans la première partie de ce projet, la fonctionnalité de base a été mise en place. Ayant une liste de personnes et leurs amitiés, on peut simuler la propagation d'une rumeur (qui n'est pas déterminée) à travers le temps.

Les fonctions de base comme `update()`, `printState()` et `inputData()` servent à donner un squelette centrale, et seront gardés (dans une certaine mesure) pour la partie 2. Par contre, la création des structures de données est lente et ennuyant (aussi à cause de la troisième fonction mentionnée) et devra être changée.

Les structures de données implémentées ici sont :

— `stagesNumber` : donne le temps de simulation (le nombre d'unités de temps à faire travailler le simulateur)

— `names` : liste de noms des personnes composant notre réseau social

— `network` : matrice de dimension `names` × `names` qui donne les amitiés et relations du réseau

— `informedPeople` : liste booléenne indiquant si les gens connaissent la rumeur ou pas

— `theGossiper` : personne qui commence à diffuser une rumeur

— `rumorSpread` : nombre de personnes ayant appris la rumeur lors d'une mise à jour

Il faut mentionner que la source de la rumeur n'est pas tenue en compte, et plusieurs personnes peuvent raconter la rumeur à un de ses amis au même temps.

2.2 Deuxième partie

Cette deuxième partie s'occupe des différents aspects, comme le chargement du réseau à partir d'un fichier, la modification de la rumeur, la propagation de la rumeur selon des nouveaux protocoles, et la récupération de paramètres en ligne de commande à l'aide de la librairie `argparse`. L'utilisateur a beaucoup plus de pouvoir et d'options avec les arguments en ligne de commande. Le contenu de la première partie est non seulement agrandi mais aussi amélioré.

Par exemple, la rumeur (qui est maintenant bien définie), peut subir des modifications, avec un pourcentage de probabilité *-p*, et en différentes formes, selon *-m*.

Aussi, le simulateur peut s'arrêter dès que tout le monde connait la rumeur. Ceci est fait en utilisant une boucle `while` et non pas `for`. On pose : *ne t'arrête pas jusqu'à ce que tout le monde connaît une rumeur*. Dans ce cas, en utilisant *-d*.

En effet, il existe un grand choix parmi les options disponibles, voir Table 1 pour une liste complète.

Un changement important dans cette deuxième partie, c'est l'ajout des rumeurs à `informed People`. Si la personne ne connaît pas la rumeur, alors `None` est la valeur par défaut. Cet objet, étant une séquence mutable, était un bon candidat pour garder les valeur des rumeurs.

Tous les algorithmes implémentés sont expliqués dans la section 2.4.

Option	Paramètre	Comportement par défaut	Description
-s	nom d'une personne	personne choisie aléatoirement	La personne qui commence à diffuser la rumeur.
-r	rumeur initiale	rumeur choisie aléatoirement	La rumeur doit être un entier sur 8 bits.
-t	temps de simulation	s'arrêter dès que tout le monde connaît la rumeur	Le temps de simulation est aussi le nombre de fois que le simulateur va être appelé
-d	Pas de paramètre	`store_false`	Ignorer les gens qui connaissent déjà la rumeur
-m	type de modification de la rumeur	`None`	type de modification éventuelle de la rumeur lorsqu'elle est racontée (choix : incremental, bitflip, none)
-p	probabilité de modification	0.1	probabilité qu'une rumeur soit modifiée lorsqu'elle est racontée (réel entre 0 et 1)
-u	règle de mise à jour	`stable`	règle de mise à jour lorsqu'une personne apprend une nouvelle version de la rumeur (choix : stable, rewrite, mixture)

TABLE 1 – Options disponibles dans la partie 2

2.3 Difficultés rencontrées, analyse et solution

Plusieurs enjeux ont été retrouvés tout au long de la réalisation de cette partie :

— **Charger le fichier réseaux** : Le fichier contient des lignes qui terminent par `\n` et elles sont lues tout au début avec la fonction `open()`, qui donne un flux de texte tamponné (`TextIOWrapper`). Il n'est pas impératif d'utiliser la méthode `readlines()` ici. Ces lignes vont être séparées en utilisant la méthode `.split()` en sachant que les séparateurs sont le deux-points " :" et la virgule ",". Étant donné qu'il y a un seule deux-points, l'élément qui se trouve à gauche sera un nom qui appartient à `names` et le reste sont ses amis (`friends`). À partir de ce moment, la création de la matrice d'amitiés est simple ; pour toute personne dans `names` d'indice *i* et ami(e) d'indice *j* qui se trouve dans `friends`, on marque `True` à l'endroit `[i][j]` dans matrice . La fonction `loadNetwork()` renvoie à la fois cette liste des noms et la matrice énoncée.

— **`argparse`** : un objet est tout d'abord assigné à la méthode `argparse.ArgumentParser`, et ensuite il faut spécifier quels sont les arguments que le programme prévoit attend de l'utilisateur, avec `argparse.addArgument()`. Cette méthode intègre plusieurs arguments mots-clés très utiles. A savoir :

`action`
Donne l'instruction de quoi faire avec le paramètre.

`dest`
Indique où enregistrer le paramètre indiqué. Sera accessible par : `args.dest`, si `args` est l'objet assigné au parser.

`type`
Parfois, les chaînes de caractères en ligne de commande doivent être interprétées comme un autre type de donnée. Ceci, ainsi que `choices`, fait la plus part de la

gestion d'erreurs. Deux types (classes) de données ont été créés pour vérifier le bon typage de la rumeur (ne doit pas dépasser 0 ou 255) et la probabilité (réelle entre 0 et 1).

`default`
La valeur par défaut du paramètre

`choices`
L'utilisateur peut choisir entre ces options.

`help`
Ce message s'affichera à côté de l'argument en question.

Le module `argparse` s'occupe de toute la gestion d'erreurs, sauf pour la personne qui commence la rumeur, car la liste `names` est chargée après 'parser' les arguments, et on ne peut pas savoir si le nom est dans la liste ou pas à ce moment. Donc, pour vérifier cela, et pour générer un nom aléatoire (pas fait avec `default`) qui se trouve dans la liste `names`, une structure conditionnelle extra est implémentée.

— **L'affichage** : la longueur maximale d'un des noms est recherchée, pour pouvoir afficher toute l'information en conséquence dans le terminal. La valeur booléenne de `informedPeople` est testé pour afficher convenablement (`None` résultera en `False` et n'importe quelle rumeur - 0 incluse - résultera en `True`). La méthode `.format()` contrôle le bon affichage des nombres en binaires et les espaces.

— **La mise à jour** : les fonction `update()` et `predictRumor` sont en charge de faire ceci. Les options suivantes doivent être gérer :

  — Le paramètre *dontTell*, option *-d* : pour arriver à ignorer les gens qui connaissent déjà la rumeur, la portée des gens doit être réduite. L'ami concerné doit avoir `None` dans `informedPeople`, pour pouvoir être informé.

  — Le paramètre *modifType*, option *-m* : selon le type de modification, la fonction `predictRumor` choisira une modification (qui, à son tour, se fera seulement avec un probabilité *-p*) pour changer la rumeur avec.

  — Le paramètre *updateRule*, option *-u* : est traité de la même façon que *modifType*. L'ami adoptera cette nouvelle version de la rumeur selon l'option donné.

  Remarquons aussi quelques points importants :

  — Pour contrôler d'où viennent les rumeurs, la rumeur passée à `predictRumor` est `informedPeople[i]` et non pas la rumeur initiale.

  — Si l'option *-d* est présente, l'option *-u* n'aura pas d'effet, puisque les gens qui connaissent déjà la rumeur sont automatiquement exclus de l'éventail des gens possibles.

  — Deux personnes peuvent choisir le même ami en commun pour transmettre la rumeur.

2.4 Découpage algorithmes

Quelques structures de code pas si évidentes sont décrites ci-dessous :

Listing 1 – Bout de code de la fonction loadNetwork

```
1 for i = 0 to lenght names do
2     network[i][i] = True
3     for j = 0 to len(friends[i])
4         indexOfFriend = names.index(friends[i][j])
5         network[i][indexOfFriend] = True
```

Le pseudo-code 1 montre comment la matrice d'amitiés est crée.

Listing 2 – Fonction bitflip

```
function bitflip(rumor):
    return rumor XOR 2 ^ (random(0, 7))
end
```

Dans le pseudo-code 2 on apprécie la fonction bitflip. Un bit de la rumeur peut être ainsi renversé. La porte logique XOR en combinaison avec une puissance de deux laisse tous les bits inchangés sauf celui qui correspond au 1, qui est renversé.

Listing 3 – Fonction incremental

```
function incremental(rumor)
    if random() > 0.5:
        rumor += 1
    else:
        rumor -= 1
    return rumor MOD 256
end
```

Le listing 4 montre (en version simplifiée) comment la fonction `update` gère la création et transmission des rumeurs.

Listing 4 – Fonction update

```
for i = 0 to lenght of informedPeople
    if informedPeople[i]:
        <Guess friend range according to dontTell flag>
        if dontTell:
            possibleRange = [ses ami(e)s, et si ils/elles ne sont ↩
               deja informe(e)s]
        else:
            possibleRange = [tous ses ami(e)s]
        if possibleRange:   # Checking for non-empty sequence
            chosenOne = possibleRange[randint (0, len(possibleRange↩
               ))]
            <Guess rumor modification according to modifType>
            rumor = predictRumor(informedPeople[i])
            <Write rumor to person according to updateRule>
            rumorSpread += eval(updateRule)()
            .
            .

informedPeople[:] = newInformedPeople
```

Et comme dernier exemple, le code 5 montre le fonctionnement de la fonction `mixture`. Dans ce cas ci, la rumeur a du être transformé dans un string (et puis liste) binaire pour sa manipulation.

Listing 5 – Fonction mixture

```
function mixture(newInformedPeople, chosenOne, rumor):
    rumorSpread = 0
    if newInformedPeople[chosenOne] does not know any rumor:
        rumorSpread += 1
    try:
        ownRumor = list en binaire de la rumeur que la personne connait
    except Exceptions that can occur:
        pass
    else:
        rumor = meme format que ownRumor, avec la rumor (peut etre changee)
        for i  = 0 to lenght rumor - 1:
            if ownRumor[i] is different from rumor[i]:
                if random() > 0.1:
                    rumor[i] = ownRumor[i] # Change this bit
        rumor = integer(rumor, 2)

    newInformedPeople[chosenOne] = rumor
    return rumorSpread
end
```

3 Conclusion

Le projet essaie de modéliser un phénomène, comme la propagation d'une rumeur, dans un réseau social. Même s'il n'est pas très précis ou spécifique, il nous donne une idée approximative de la propagation d'une rumeur dans un environnement comme celui décrit.

Références

[1] Argparse tutorial. `https://docs.python.org/3/howto/argparse.html`. Python Software Foundation.

[2] KAZMIERCZAK, M. Argparse cookbook. `https://mkaz.com/2014/07/26/python-argparse-cookbook/`.

[3] PYMOTW. argparse. `http://pymotw.com/2/argparse/`.

A Exécution

Des exemples d'exécution du programme ont été ajoutés sous forme de capture d'écran.

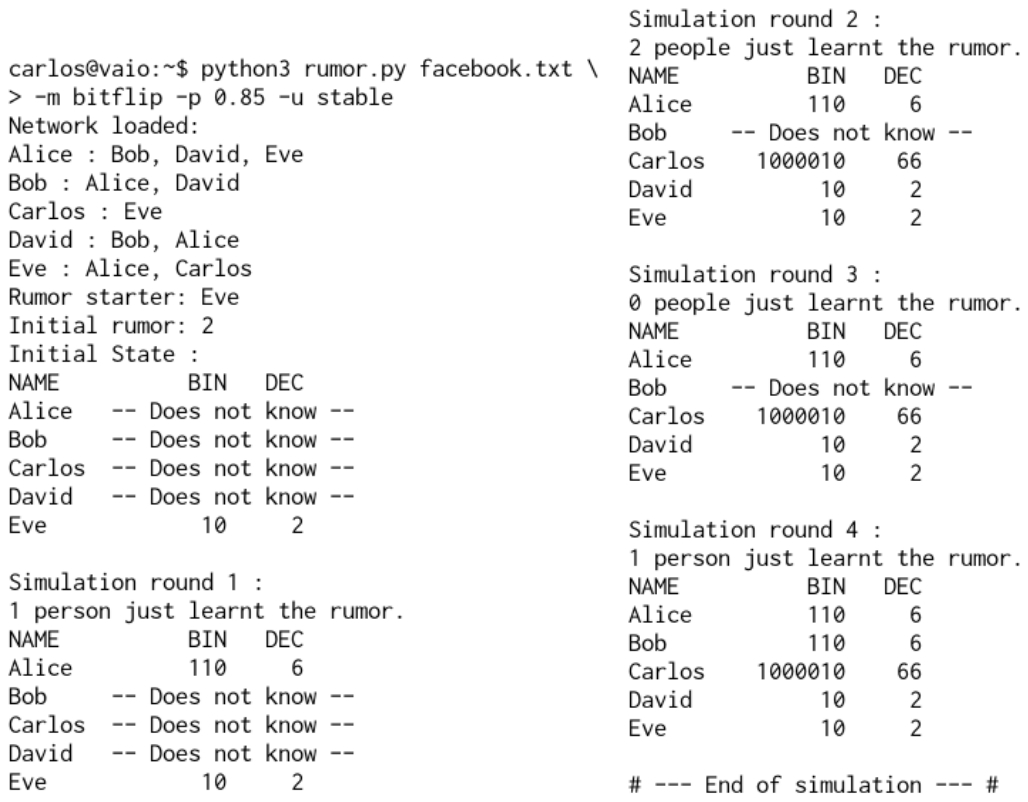

FIGURE 1 – Exécution du programme

```
carlos@vaio:~$ python3 rumor.py facebook.txt \
> -m incremental -p 0.5 -d
Network loaded:
Alice : Bob, David, Eve
Bob : Alice, David
Carlos : Eve
David : Bob, Alice
Eve : Alice, Carlos
Rumor starter: Bob
Initial rumor: 110
Initial State :
NAME          BIN   DEC
Alice   -- Does not know --
Bob       1101110   110
Carlos  -- Does not know --
David   -- Does not know --
Eve     -- Does not know --

Simulation round 1 :
1 person just learnt the rumor.
NAME          BIN   DEC
Alice     1101111   111
Bob       1101110   110
Carlos  -- Does not know --
David   -- Does not know --
Eve     -- Does not know --

Simulation round 2 :
1 person just learnt the rumor.
NAME          BIN   DEC
Alice     1101111   111
Bob       1101110   110
Carlos  -- Does not know --
David     1101111   111
Eve     -- Does not know --

Simulation round 3 :
1 person just learnt the rumor.
NAME          BIN   DEC
Alice     1101111   111
Bob       1101110   110
Carlos  -- Does not know --
David     1101111   111
Eve       1110000   112

Simulation round 4 :
1 person just learnt the rumor.
NAME          BIN   DEC
Alice     1101111   111
Bob       1101110   110
Carlos    1110000   112
David     1101111   111
Eve       1110000   112

# --- End of simulation --- #
```

FIGURE 2 – Exécution du programme

B Propositions de fonctionnalités

1. Les quatre premiers bits (MSB) de la rumeur pourraient indiquer qui est la personne concernée par cette rumeur. Ceci sera fait sur 8 ou 12 (au cas où on veut garder le format rumeur à 8 bits). Ce sera une sorte de système d'adressage, où le nombre maximale de gens à qui la rumeur peut affecter (et à une seule à la fois) sera une puissance de deux. 2^4 dans ce cas.
2. Le fichier texte pourrait être enrichi pour contenir plus d'informations sur les gens. Par exemple, il pourrait contenir le type de modification et règle de mise à jour que chaque personne souhaite adopter. Tous ces options extra sont plus facilement maniable en créant une instance de class (Person par exemple) pour chaque personne.
3. Pour finir, on pourrait connecter en quelque sorte, notre programme avec un vrai réseau social, comme Facebook, Google+ ou un autre logiciel de réseautage pour évaluer l'impact de la propagation d'une rumeur.