RAPPORT DU PROJET

OUTILS ALGO

MEMBRE DU GROUPE

DIALLO Abdou Wahab

KOUASSI Kofié Ismael

ROUGUI Rania

SATOURI Maha

14 NOVEMBRE 2021

Table des matières

SOMMAIRE

INTRODUCTION

Le but du projet est de vous initier à la lecture et la compréhension d'articles scientifiques. Dans ce but, nous demandons de lire des articles récents et d'implémenter certains algorithmes de graphes. Le contexte est celui des réseaux sociaux et de leur étude. Un réseau social peut facilement être vu comme un graphe où les nœuds représentent les utilisateurs et les arêtes les liens qui existent entre eux dans le réseau. La problématique du projet est d'énumération des cliques maximales dans un graphe. En particulier dans notre contexte, cela revient à chercher dans un graphe des communautés qui représentent l'ensemble de nœuds très densément connectés entre eux. Pour mener à bien ce travail, nous travaillerons avec le langage de programmation **python**. Le travail s'articulera autours de trois grandes parties. **La première partie** consistera à générer des graphes aléatoires de deux façons différentes. Ensuite, **la seconde partie** mettra en avant l'implémentation des algorithmes de complexité exponentielle pour l'énumération des cliques. Enfin dans **la troisième partie,** nous implémenterons à proprement dire les algorithmes d'énumération des cliques maximales

Partie 1 : Génération des graphes Aléatoires

I. Graphe construit avec probabilité aléatoire au niveau de chaque arrête

Soit un graphe G défini par **: G = (V, E)**

V : ensembles des sommets ou nœuds

E = V ×V : ensembles des arcs ou arrêtes

Le but de notre algorithme c'est de construire G de telle sorte que l'apparition de chaque arrête sois conditionné par une certaine probabilité.

```
Generation_Graphe (Nombre_de_Sommet)
{
    Pour i allant de 1 à Nombre_de_Sommet faire                    1
        Pour j allant de 1 à Nombre_de_Sommet faire                2
            p = random(0,1)          # p sera un nombre aléatoire compris entre 0 et 1
            si (i==j) alors
                pas de boucle sur un sommet
            sinon si ( 0.3 <= P <= 0.8) alors     #Le choix de l'intervalle est arbitraire   3
                il existe une arrête entre le sommet i et j
            sinon
                pas d'arrête
}
```

- Complexité En temps :
  - 1 : s'exécute en Nombre_de_Sommet opérations
  - 2 : s'exécute égalent en Nombre_de_Sommet opérations
  - 3 : s'exécute en un temps constant

Soit n : Nombre_de_Sommet

Donc T(n) = n*n

- Complexité En espace :
  - 1 : n cases mémoire pour les sommets
  - 2 : m cases mémoire pour les arrêtes
  - 3 : temps constant pour les calculs

Donc T(n) = n*m

II. Graphe construit avec Algorithme de **Barabàsi-Albert**

G un graphe triangle de trois (3) sommets.

Le but de l'algorithme c'est d'agrandi G en ajoutant **des sommets** suivant un paramètre **m** qui représentera le nombre d'arrête maximum partant d'un sommet

```
Generation_Graphe (n : Nombre de sommet à ajouter)
{
    m : Nombre d' arrête de chaque nouveau sommet (ici m = 2)
    G : Graphe initial complet avec 3 sommets
    Pour i allant de 1 à n faire                                   1
        Pour j allant de 1 à m faire                               2
            Choix_Sommet = random (liste des sommets de G) #pour établir une arrête entre le sommet (i) et ceux existant dans G
            p(i) = (m)/ ( ∑degré du graphe) #probabilité d'apparition du sommet (i)
            G_tempo = {sommet(i) : [choix_sommet(1), choix_sommet(2)] } #Cela signifie qu'il existe une arrete entre i et choix_sommet(1) et entre i et choix_sommet(2)
                                                                   3
        G = G + G_tempo  #Mise a jour du graphe initial            4
}
```

- Complexité En temps :
  - 1 : s'exécute en n opérations
  - 2 : s'exécute égalent en 2 opérations car m = 2
  - 3 : s'exécute en un temps constant
  - 4 : s'exécute en un temps constant

Donc T(n) = n

- Complexité En espace :
  - 1 : n cases mémoire pour les nouveaux sommets
  - 2 : 2 cases mémoire (pour les arrêtes) associé à chaque sommet
  - 3 : temps constant pour les calculs
  - 4 : temps constant pour ce calcul

Donc T(n) = n+2

Partie 2 : Enumération des cliques

I. Algorithme de **Bron Kerbosch** version standard

1. Sans pivot

Dans sa forme élémentaire, l'algorithme de Bron–Kerbosch est un algorithme de retour sur trace récursif qui cherche toutes les cliques maximales dans un graphe *G*. Plus précisément, étant donnés trois ensembles de nœuds disjoints *R*, *P* et *X*, il cherche toutes les cliques maximales contenant tous les sommets de *R*, certains de *P*, mais aucun de *X*. Précisément :

- L'ensemble *R* est un sous-ensemble des sommets de la potentielle clique.
- L'ensemble *P* est l'ensemble des sommets candidats pour être ajoutés à la potentielle clique.
- L'ensemble *X* contient des sommets déjà traités ou appartenant déjà à une clique maximale.

À chaque appel de l'algorithme, *P* et *X* sont des ensembles disjoints dont l'union forme des cliques lorsqu'on les ajoute à *R*. Autrement dit, $P \cup X$ est l'ensemble de nœuds qui sont connectés à tous les éléments de *R*. Quand *P* et *X* sont tous les deux vides, on ne peut plus ajouter d'éléments à *R*, qui est donc une clique maximale que l'algorithme retourne.

```
BronKerbosch_Sans_Pivot (R, P, X)
{
        R = {}
        X = {}
        P = ensemble des sommets du graphe
        si P et X sont vides alors
                déclarer R clique maximale
        pour tout sommet v de P faire
                BronKerbosch1(R ∪ {v}, P ∩ N(v), X ∩ N(v))
                P := P \ {v}
                X := X ∪ {v}
}
```

- ❖ Complexité En temps :

  $T(n) = O(3^{n/3})$

- ❖ Complexité En espace :

  $T(n) = O(n + q\Delta)$ Avec q : la taille maximum de la clique et Δ : degré max

2. Avec pivot

La forme élémentaire de l'algorithme décrite ci-dessus est inefficace pour les graphes qui contiennent beaucoup de cliques non maximales car on effectue un appel récursif par clique, maximale ou non. Pour économiser du temps en permettant à l'algorithme de retourner sur traces plus rapidement lorsqu'il parcourt des branches qui ne contiennent pas de cliques maximales, Bron et Kerbosch ont introduit une variante qui utilise un « nœud pivot » u, choisi dans P ou plus généralement dans $P \cup X$, comme remarqué dans des travaux postérieurs[5]. Toute clique maximale doit inclure soit u, soit un nœud qui n'est pas parmi les voisins de u, sans quoi la clique pourrait être agrandie en y ajoutant u. Par conséquent, il n'est nécessaire de tester que u et l'ensemble des nœuds de P qui ne sont pas voisins de u en tant que nœud qu'on tente d'ajouter à R à chaque appel récursif de l'algorithme.

```
BronKerbosch_Avec_Pivot (R, P, X)
{
        si P et X sont vides alors
                déclarer R clique maximale
        choisir un sommet pivot u dans P ⋃ X
        pour tout sommet v dans P \ N(u) faire
                BronKerbosch_Avec_Pivot (R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
                P : = P \ {v}
                X : = X ⋃ {v}
}
```

II. Algorithme de **Bron Kerbosch** version améloirée

1. Ordre de dégénérescence

En théorie des graphes, la dégénérescence est un paramètre associé à un graphe non orienté. Un graphe est k-dégénéré si tout sous-graphe contient un nœud de degré inférieur ou égal à k, et la dégénérescence d'un graphe est le plus petit k tel qu'il est k-dégénéré.

```
Ordre_Degenerescence (G : graphe)
{
    pour i allant de 1 à n faire
        di = min( d(Gi) )        #Choisi le plus degré dans le graphe Gi
        vi = sommet qui a pour degré di dans Gi
        Gi+1 = Gi – vi           #On enlève le sommet vi ainsi que tous ses voisins
    k = max(di)
    L = liste des sommets ordonnée de degré minimum
}
```

La complexité de cet algorithme est linéaire. T(n) = O(n)

2. Avec ordre de dégénérescence

L'algorithme de Bron–Kerbosch visite les sommets dans un ordre de dégénérescence, on garantir alors l'ensemble P des sommets candidats à chaque appel (les voisins de v qui sont situés après dans l'ordre) est au plus de taille k. L'ensemble X des sommets exclus est constitué de tous les voisins de v qui sont situés avant dans l'ordre et peut être bien plus grand que k.

```
BronKerbosch_ Ordre_Degenerescence(G : Graphe)
{
    P = ensemble des sommets de G
    R = Ø
    X = Ø
    pour tout sommet v visités dans Ordre_Degenerescence faire
        BronKerbosch2({v}, P ⋂ N(v), X ⋂ N(v))
        P : = P \ {v}
        X : = X ⋃ {v}
}
```

- Complexité En temps :

  $T(n) = O(dn3^{d/3})$ où d est la dégénérescence du graphe

Partie 3 : Enumération des cliques maximales

I. Algorithme d'énumération des cliques version 1

Dans cet algorithme, le temps d'énumération dépend uniquement de la dégénérescente du graphe. Les grandes lignes du premier algorithme sont les suivantes. Nous considérons chaque sous-graphe G_i, de manière itérative, à partir de G_1 jusqu'à G_n. Nous trouvons toutes ses cliques maximales et nous cherchons pour eux dans un arbre de suffixes. S'il y a une correspondance, la clique est rejetée, sinon elle est sortie

```
Find_clique_V1 (G : Graphe)
{
        Calculer k et trouver l'ordre des sommets L
        Calculer la liste adjacente générée par G
        Intialiser un abre T
        pour j de 1 à n
            Trouver les cliques de Gj
            pour tout chaque clique K de Gj faire
                ordonner les sommets de K suivant L
                trouver = rechercher K dans T
                si (trouver) alors
                        rejetée K
                sinon
                        retourner K
}
```

- Complexité En temps :

  $T(n) = O(m + s(k + 1)n)$: temps de prétraitement

  $T(n) = \alpha\, O(f(k + 1)q)$: temps d'énumération

Avec k : nombre dégénéré

α : nombre maximal de cliques

q : taille maximum de la clique

II. Algorithme d'énumération des cliques version 2

Le deuxième algorithme est une version légèrement modifiée du premier.

```
Find_clique_V2 (G : Graphe)

{
        Pour j allant de 1 à n faire
                Calculer toutes les cliques maximales de G

                Pour chaque clique maximal K de G faire
                        pour chaque sommet x pris dans K faire
                                si un des voisins de x dans G a un rang inférieur à v j dans

                                  l'ordre des Sommets L est adjacent à tous les sommets dans K

                                        Rejeté K

                                sinon

                                        Retourné K

}
```

- Complexité En temps :

  $T(n) = O(m + s(k + 1)n)$: temps de prétraitement

  $T(n) = \alpha\, O(f(k + 1)q)$: temps d'énumération

- Complexité En espace :

  $T(n) = O(g(k+1))$

CONCLUSION

Au terme de notre travaille, nous avons d'une part générer des graphes de manières aléatoires en respectant des probabilités et d'autre parts énumérer des cliques maximales en basant sur les l'algorithme de Bron Kerbosch.

Pour notre part, le projet nous a permis de nous a permis d'apprendre comment se comporter avec des articles scientifiques chose qui était nouvelle pendant notre cursus. Cela a été l'occasion de développer davantage nos compétences langage Python

BIBLIOGRAPHIE

Lien

[1] : https://fr.wikipedia.org/wiki/Algorithme_de_Bron-Kerbosch

[2] : https://www.tutorialspoint.com/python/python_dictionary.htm

[3] : https://developers.google.com/edu/python/lists

[4] : https://compucademy.net/generating-random-graphs-in-python/

[5] : https://networkx.org/documentation/stable/tutorial.html

[6] : https://en.wikipedia.org/wiki/Degeneracy_(graph_theory)#Algorithms

PDF

Papier1.pdf

Papier2.pdf

Sujet.pdf