L3 Informatique – Université de Bourgogne-Franche-Comté

Rapport de projet

Etude de la dégénérescence des grands graphes réels

VanTai NGUYEN - Théophile Romieu
01/05/2022

Table des matières

Introduction

Ce projet de graphes a pour but d'étudier la dégénérescence des grands graphes réels. La dégénérescence est une propriété des graphes non-orientés utilisée pour l'étude des graphes afin de résoudre certaines problématiques comme la coloration de graphe. Pour donner une définition « avec les mains », la dégénérescence d'un graphe peut être trouvée en exécutant l'algorithme suivant : prendre un k = 1 et supprimer tous les sommets de degré inférieur ou égal à 1 (cette action donne un sous-graphe), il faut ensuite continuer en incrémentant k tant qu'il reste un sommet dans le sous-graphe. Quand tous les sommets sont supprimés, le k obtenu est la dégénérescence du graphe (on dit alors que le graphe est k-dégénéré). Au cours de ce projet, nous allons implémenter un algorithme de calcul de dégénérescence créé par nos soins. Nous allons ensuite nous intéresser à l'algorithme de Matula & Beck permettant de calculer la dégénérescence d'un graphe en un temps linéaire, et nous implémenterons une fonction permettant d'avoir une représentation graphique du graphe avec les sommets disposés en cercles suivant leurs numéros de centre.
Pour tester le projet, veuillez suivre les étapes suivantes : vérifiez que la commande « python » est disponible quand vous l'utilisez dans un terminal, si elle ne l'est pas il faut installer python. Dézippez l'archive de notre projet et allez dans le dossier « projet-graphes », ouvrez un terminal et utilisez la commande « python "src/main.py" ». Cette commande lance le script et doit vous afficher les numéros de centre des sommets du graphe ainsi qu'une représentation visuelle de ces numéros de centre.

Travail effectué pour répondre au projet

Nous avons choisi le langage de programmation Python (version 3) pour répondre aux problématiques du sujet. Nous sommes habitués à ce langage, de plus il permet une gestion très simple des listes ce qui nous fera gagner du temps d'implémentation.

Traduction d'un fichier CSV en graphe

La première étape du projet consiste en la traduction d'un graphe provenant d'un fichier CSV. En effet, un graphe peut être représenté par un fichier CSV avec deux colonnes. Chaque colonne contient des numéros de nœud, un couple de nœuds indique une arête entre ces derniers. Prenons ce graphe qui sera utilisé en tant qu'exemple tout au long de ce rapport :

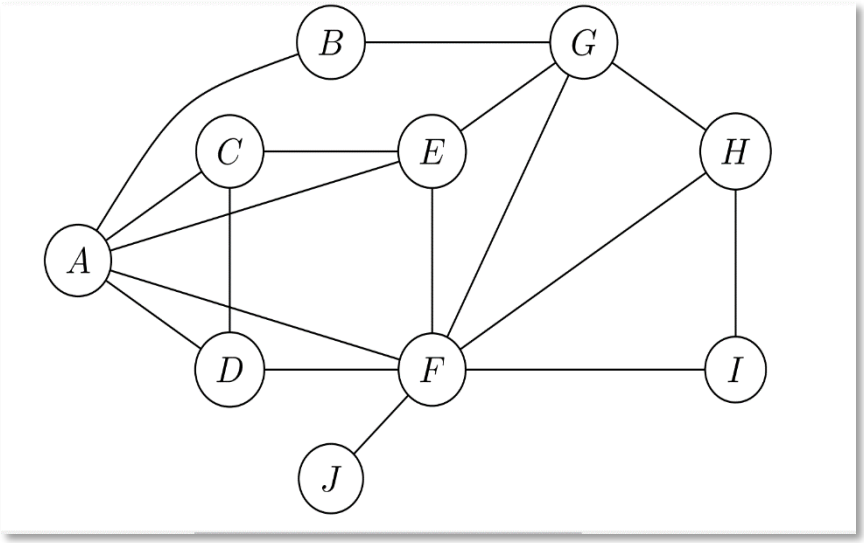

Figure 1 - Graphe d'exemple

Ce graphe peut être traduit par un fichier CSV, deux sommets sont séparés par une virgule et chaque couple traduit une arête entre ces sommets. Voici la représentation du graphe précédent en fichier CSV :

NETWORK,NUMBER_OF_PARTICIPANTS
0,1
0,2
0,3
0,4
0,5
1,6
2,3
2,4
3,5
4,6
4,5
5,9
5,6
5,7
6,7
7,8
8,5
9,5

Figure 2 - Traduction en CSV du graphe d'exemple

Nous devons désormais écrire une fonction permettant de lire le fichier et le traduire en graphe. Pour stocker le graphe en mémoire, nous allons le représenter grâce à sa matrice d'adjacence. La matrice d'adjacence sera une liste à deux dimensions dont la taille est équivalente au nombre de sommets différents dans le graphe (dans notre exemple, la matrice d'adjacence du graphe sera une liste de taille 10x10).
Voici la fonction utilisée pour créer cette matrice d'adjacence :

```
def read_file(filename):
    #On récupère le sommet ayant l'indice le plus haut
    size = max_vertex(filename)
    file = open(filename,"r")
    #Création d'une matrice vide
    mat = [[0 for  i in range(size+1)]for j in range(size+1)]
    for couple in file:
        #Supression des retours à la ligne et des virgules
        coupleWithoutReturn = couple.replace("\n", "")
        vertices = coupleWithoutReturn.split(",")
        if(vertices[0].isdigit() and vertices[1].isdigit()):
            vertexA = int(vertices[0]) #Sommet A
            vertexB = int(vertices[1]) #Sommet B
            G.add_edge(vertexA,vertexB)
            #Nouvelle arête dans la matrice d'adjacence
            mat[vertexA][vertexB]  = 1
            mat[vertexB][vertexA]  = 1
    file.close()
    return mat
```

Cette fonction utilise « `max_vertex` », voici le code de cette fonction :

```python
def max_vertex(filename):
    max = -1
    file = open(filename,"r")
    for couple in file:
        coupleWithoutReturn = couple.replace("\n", "")
        vertices = coupleWithoutReturn.split(",")
        if(vertices[0].isdigit() and vertices[1].isdigit()):
            #Test du numéro de sommet pour savoir s'il est supérieur au max
            #S'il l'est, ce sommet devient le sommet max
            if int(vertices[0]) > max:
                max = int(vertices[0])
            if int(vertices[1]) > max:
                max = int(vertices[1])
    file.close()
    #Retour du sommet ayant l'indice le plus grand
    return max
```

En appelant la fonction « `read_file("graph/graphe3.csv")` » nous obtenons la matrice d'adjacence du graphe contenu dans le fichier « graphe3.csv ».

Développement de l'algorithme de dégénérescence

Cette partie s'intéressera à notre algorithme de calcul de la dégénérescence d'un graphe. Cet algorithme renverra les numéros de centre des sommets du graphe (la k-dégénérescence du graphe peut être obtenue en prenant le maximum des numéros de centre).
Notre algorithme fonctionne de la manière suivante :

- Créer une liste initialisée à 0 des numéros de centre de chaque sommet
- Créer une liste initialisée à 0 qui contiendra les sommets supprimés (-1 si le sommet est supprimé)
- Initialiser k à 1
- Tant que tous les sommets ne possèdent pas de numéro de centre
    - Calculer les degrés de chaque sommet du graphe
    - S'il existe un sommet dont le degré est plus petit ou égal à k et qui n'est pas déjà supprimé
        - Dresser une liste des sommets possédant la même caractéristique
        - Pour chacun de ces sommets
            - Mettre la ligne et la colonne du sommet concerné à 0 dans la matrice d'adjacence (équivaut à supprimer le sommet)
            - Mettre son numéro de centre égal à k dans la liste prévue
            - Marquer le sommet à -1 dans la liste des sommets supprimés
    - Sinon
        - Incrémenter k de 1
- Renvoyer la liste des numéros de centre

Comme vous avez pu le remarquer, nous ne supprimons pas réellement les sommets de la matrice, nous préférons modifier la matrice d'adjacence pour modifier sa ligne et sa colonne à 0 (plus aucun sommet ne possède d'arête avec le sommet supprimé), et indiquer que ce sommet a été supprimé en mettant l'indice du sommet à -1 dans la liste des sommets supprimés.
Voici le code permettant l'exécution de cet algorithme :

```
def degenerescence(path_graphe):
    mat_adj = read_file(path_graphe)
    numero_centre = [0]*len(mat_adj)
    nb_adj = min_center(mat_adj)
    marque_cas =[0]*len(mat_adj)
    k = 1
    while (checkList(numero_centre)):
        nb_adj = min_center(mat_adj)
        if(test_cas(nb_adj,k,marque_cas)):
            nb_centre_iden = same_degree(nb_adj,k,marque_cas)
            for i in nb_centre_iden:
                delete_sommet(mat_adj,i)
                numero_centre[i] = k
                marque_cas[i] =-1
        else :
            k +=1
    return numero_centre
```

Code de la fonction « `min_center` » :

```
def min_center(mat):
    centre_min = []
    for i in mat :
        nb_arrete_adjacent = 0
        for j in i :
            nb_arrete_adjacent += j
        centre_min.append(nb_arrete_adjacent)

    return centre_min
```

Code des fonctions « `checkList` », « `test_cas` », « `same_degree` » et « `delete_sommet` » :

```
def checkList(list):
    cond = False
    for i in list :
        if (i == 0) :
            cond = True
    return cond

def test_cas(mat,it,marque):
    cond = False
    for i in mat:
        if (i <= it and marque[mat.index(i)] !=-1):
            cond=True
    return cond

def same_degree(mat,min,marque):
    same = []
    for i in range (len(mat)):
        if(min >= mat[i] and marque[mat.index(mat[i])]!=-1):
            same.append(mat.index(mat[i]))
```

```
            mat[i]=-1
    return same

def delete_sommet(mat_adj,sommet):
    for j in range(len(mat_adj)) :
        mat_adj[sommet][j] =0
        mat_adj[j][sommet] =0
    return mat_adj
```

Implémentation des extensions

Algorithme de Matula & Beck

L'algorithme de Matula & Beck permet de calculer la dégénérescence d'un graphe en temps linéaire. Nous allons implémenter cet algorithme en nous aidant de ce site qui indique les différentes étapes à suivre pour implémenter cet algorithme :

- Initialiser une liste de sortie L.
- Calculer un nombre d_v pour chaque sommet v dans G, le nombre de voisins de v qui ne sont pas déjà dans L. Initialement, ces nombres ne sont que les degrés des sommets.
- Initialiser un tableau D tel que D [i] contienne une liste des sommets v qui ne sont pas déjà dans L pour lesquels $d_v = i$.
- Initialisez k à 0.
- Répétez n fois :
    - Balayez les cellules du tableau D [0], D [1], ... jusqu'à trouver un i pour lequel D [i] est non vide.
    - Mettre k à max(k, i)
    - Sélectionnez un sommet v de D [i]. Ajoutez v au début de L et supprimez-le de D [i].
    - Pour chaque voisin w de v pas déjà dans L, soustrayez un de d_w et déplacez w vers la cellule de D correspondant à la nouvelle valeur de d_w.

Nous avons suivi chacune de ces étapes pour implémenter l'algorithme de Matula & Beck. Voici le code de cet algorithme (vous trouverez des commentaires dans le code indiquant les étapes de l'algorithme) :

```
def Algo_Matula_Beck(path_graphe):
    mat_adj = read_file(path_graphe)
    """Une liste de sortie L"""
    L = []
    """Liste numéros de centre"""
    k_center = [0]*len(mat_adj)
    """Nombre de voisin de sommet v"""
    d_v = min_center(mat_adj)

    """Un tableau D telque D[i] contient une list des sommets v qui ne sont
pas déja dans L pour lesquels d_v =i"""
    D = list_same_degree(d_v,L)
```

```
    """Intialiser k à 0"""
    k = 0
    while(len(L)!=len(mat_adj)):
        """Balayez les cellules du tableau D[0,D[1] jusqu'à trouver un i
pour le quel D[i] est non vide"""
        index_non_vide = k_core(D)
        """Mettre k à max(k,i)"""
        k = max(k,index_non_vide)
        """Sélectionnez un sommet v de D[i]. Ajoutez v au début de L et
supprimez-le de D[i]"""
        for i in range (len(D[index_non_vide])):
            if(D[index_non_vide][i] not in L):
                L.append(D[index_non_vide][i])
                delete_sommet(mat_adj,D[index_non_vide][i])
                k_center[D[index_non_vide][i]]=k
        d_v = min_center(mat_adj)
        D = list_same_degree(d_v,L)
    return k
```

De plus, voici le code des fonctions utilisées dans l'algorithme de Matula & Beck :

```
def list_same_degree(mat,L):
    liste = [[]]*len(mat)
    for i in mat:
        liste[i] = same_degree_v2(mat,i,L)
    return liste

def same_degree_v2(mat,min,L):
    same = []
    mat_copie = mat.copy()
    for i in range (len(mat_copie)):
        if(min == mat[i] and mat_copie.index(min) not in L):
            same.append(mat_copie.index(mat_copie[i]))
            mat_copie[i]= 0
    return same

def k_core(mat):
    list_k_core =[]
    for i in mat :
        if i :
            list_k_core.append(mat.index(i))
    return list_k_core[0]
```

Dessin du graphe suivant les numéros de centre

Pour mieux visualiser le graphe suivant les numéros de centre, nous avons créé une fonction créant le dessin du graphe passé en paramètre. Le principe est le suivant : plus le numéro de centre est élevé, plus le sommet du graphe se retrouvera au centre, à l'inverse plus son numéro de centre est faible et plus il sera loin du centre. Les sommets sont disposés sur des cercles qui représentent des numéros de centre, tous les sommets possédant un même numéro de centre sont placés sur le même cercle.

Nous utilisons les bibliothèques « `numpy` », « `networkx` » et « `pyplot` » de « `matplotlib` ». La fonction de dessin se nomme « `Dessin_Graphe` », elle affiche une nouvelle fenêtre avec le dessin du graphe. Voici le résultat de cette fonction sur le graphe d'exemple :

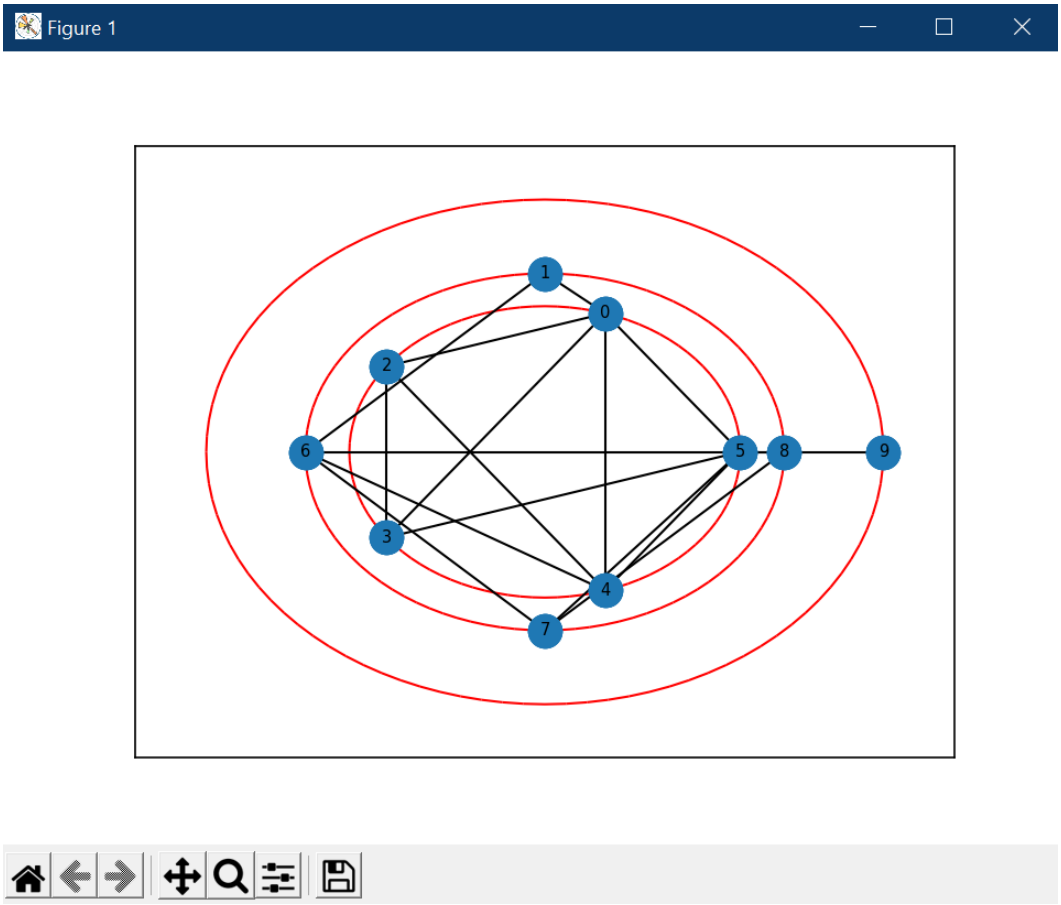

Figure 3 - Dessin du graphe suivant les numéros de centre des sommets

Grâce à cette illustration du graphe, nous pouvons voir que :

- Numéro de centre égal à 1 → sommet 9
- Numéro de centre égal à 2 → sommets 1, 6, 7 et 8
- Numéro de centre égal à 3 → sommets 0, 2, 3, 4 et 5