Ecole Centrale Casablanca

Projet de Session

RESEAUX SOCIAUX & GRAPHES

Membres :

ARABA Joé Nestor M.

BENRGUIG Ayyoub

Superviseur :

A. AIT EL CADI

17 décembre 2023

Table des matières

0.1 Introduction

La théorie des graphes, en tant que discipline mathématique étudiant les relations entre des entités interconnectées, trouve une application cruciale dans l'analyse des réseaux sociaux contemporains. Ces derniers, devenus des éléments incontournables de notre quotidien, abritent une multitude d'interactions entre individus. Dans le cadre de ce projet, notre objectif principal est d'explorer et de quantifier ces interactions en utilisant les principes fondamentaux de la théorie des graphes.

Notre attention se focalise spécifiquement sur la centralité au sein des réseaux sociaux, une mesure qui révèle le degré de prestige ou de grégarisme des individus au sein de ces structures complexes. La centralité offre un éclairage précieux sur la manière dont certaines entités influencent la dynamique globale du réseau, impactant ainsi les flux d'informations, les décisions collectives et les schémas de connexion.

Ce projet, conçu pour être réalisé en équipe de deux personnes, s'appuie sur les fondements acquis au cours de l'étude approfondie de la "Théorie des graphes". Nous allons mettre en pratique ces connaissances en utilisant le langage de programmation Python ainsi que des outils spécifiques identifiés au cours de notre formation. Cette approche pratique nous permettra d'appréhender de manière concrète les concepts théoriques abordés en classe, tout en développant des compétences en programmation cruciales pour l'analyse de données complexes.

Au travers de cette exploration, nous chercherons à dévoiler les structures sous-jacentes et les dynamiques émergentes au sein des réseaux sociaux, jetant ainsi les bases d'une compréhension approfondie des interactions humaines à l'ère numérique.

0.2 La description du problème

L'objectif de ce problème est d'analyser et de comparer des réseaux sociaux en utilisant différentes mesures de centralité. La centralité est un concept clé dans l'étude des réseaux sociaux, car elle permet de quantifier l'importance et l'influence des individus au sein d'un réseau. Trois mesures spécifiques de centralité seront examinées : la centralité en termes d'importance du voisinage (Edge Centrality), la centralité d'intermédiarité (Betweenness Centrality) et la centralité de proximité (Closeness Centrality). Chacune de ces mesures offre une perspective unique sur la position des acteurs dans le réseau social, mettant en lumière différentes facettes de leur rôle et de leur impact.

— **Centralité en termes d'importance du voisinage (Edge Centrality - $\mathcal{C}_e$)** Cette mesure évalue l'importance des nœuds en se basant sur leur connectivité locale, en considérant la structure de leur voisinage immédiat. Parmi les méthodes utilisées, le PageRank est particulièrement pertinent, car il prend en compte la qualité des liens, attribuant une importance accrue aux nœuds liés à d'autres nœuds importants. La formule de la centralité d'intermédiarité pour un nœud v fait appel à la somme du nombre de plus courts chemins passant par ce nœud, normalisée par le nombre total de plus courts chemins dans le réseau.

$$\mathbf{C}_p = \beta \left(\mathbf{I} - \alpha A^T D^{-1}\right)^{-1} \cdot \mathbf{1}$$

C_p : Il s'agit du vecteur ou de la matrice. C_p, représentant probablement une mesure de centralité dans le contexte de la théorie des graphes ou d'autres domaines connexes. α : C'est un coefficient, parfois appelé le taux d'amortissement, qui ajuste l'effet de la partie $A^T D^{-1}$ dans l'inversion de $(\mathbf{I} - \alpha A^T D^{-1})$ β : C'est un coefficient multiplicatif, souvent utilisé pour ajuster l'importance de la contribution de la partie droite de l'équation.

I : Il représente la matrice identité, une matrice carrée avec des uns sur la diagonale principale et des zéros ailleurs.

— **Centralité d'intermédiarité (Betweenness Centrality - $\mathcal{C}_c$)** La centralité d'intermédiarité mesure l'importance d'un nœud dans le maintien de la communication entre d'autres nœuds du réseau. Un nœud avec une centralité d'intermédiarité élevée agit comme un pont, facilitant les échanges d'information entre différentes parties du réseau. La formule de la centralité d'in-

termédiarité pour un nœud fait appel à la somme du nombre de plus courts chemins passant par ce nœud, normalisée par le nombre total de plus courts chemins dans le réseau.

$$C_b(v) = \frac{\sum_{s \neq t \neq v \in V} \sigma_{st}(v)}{\sigma_{st}} / [(n-1)(n-2)]$$

Avec :

n : le nombre de nœuds (acteurs) dans le réseau.

σ_{st} : est le nombre de plus courts chemins de s vers t.

$\sigma_{st}(v)$: est le nombre de plus courts chemins de s vers t, passant par v

— **Centralité de proximité (Closeness Centrality - $\mathcal{C}_c$)**

Cette mesure évalue la rapidité avec laquelle un nœud peut atteindre tous les autres nœuds du réseau. Les nœuds avec une centralité de proximité élevée sont considérés comme centraux, car ils sont efficaces pour transmettre l'information rapidement à travers le réseau. La centralité de proximité pour un nœud v est calculée en prenant l'inverse de la moyenne des plus courts chemins entre ce nœud et tous les autres nœuds du réseau.

$$C_b(v) = \frac{1}{\overline{l_v}}$$

Avec : - $\overline{l_v}$: la moyenne des plus courts chemins de v vers les autres nœuds du réseau :

$$\overline{l_v} = \frac{\sum_{s \neq v \in V} l_{s,v}}{(n-1)}$$

- Où, n : le nombre de nœuds (acteurs) dans le réseau. Et $l_{s,v}$: le plus court chemin entre s et v.

0.3 Bilan

0.3.1 Centralité en termes d'importance du voisinage

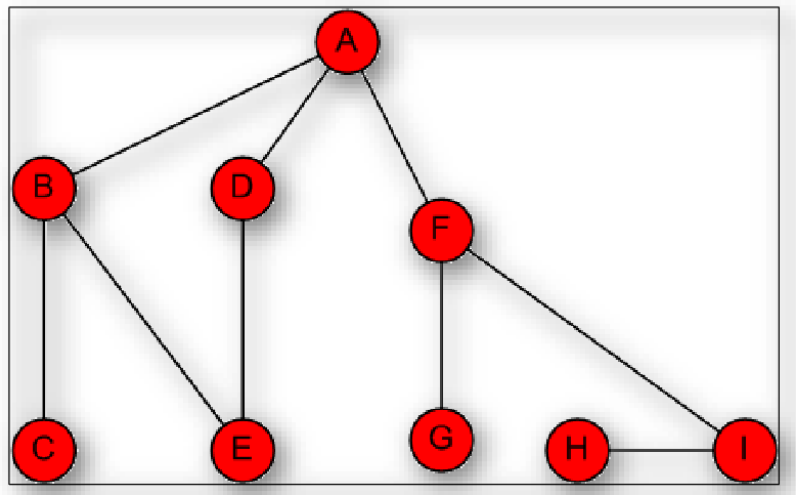

FIGURE 1 – Reseau Zero

a) Calcul des valeurs du « PageRank » avec :

— $\alpha = 1, \beta = 0$: Dans ce cas on va utiliser l'approche alternatice-chaine de markov.

Iteration	A	B	C	D	E	F	G	H	I
0	0.12962962962962962	0.2037037037037037	0.0370370370370035	0.092592592592592 59	0.092592592592592 59	0.2037037037037037	0.0370370370370035	0.055555555555555 5	0.14814814814814814
1	0.18209876543209874	0.12654320987654322	0.06790123456790123	0.08950617283950617	0.11419753086419752	0.15432098765432098	0.06790123456790123	0.07407407407407407	0.12345679012345678
2	0.1383744855967078	0.18569958847736623	0.04218106995884774	0.117798353909465	0.08693415637860083	0.19032921810699588	0.05144032921810699	0.061728395061728395	0.12551440329218105
3	0.18424211248285322	0.13177297668038841	0.06189986282578874	0.089591906721536 36	0.12079903978052124	0.16032259939643347	0.06344307270233196	0.062757201646090053	0.12517146776406035
4	0.14216106538637402	0.18371342021033377	0.04392432556012803	0.121813557384545 02	0.0887202789208962	0.18744284407864653	0.05344078646547782	0.062585733882030 18	0.11619798811156834
5	0.18462553345526594	0.13567148681603414	0.06123780673677792	0.091747161255906 11	0.12214458542905043	0.15892680231672	0.06248094802621551	0.058098994055784 17	0.12506668190824569
6	0.1440730103388711	0.18385194939363918	0.04522382893867846	0.122614137199613 87	0.0910974095666311	0.186556133465427	0.05297560077223999	0.062533340954122 84	0.11107459482802416
7	0.18477642773374653	0.1387968705016173	0.06128398131213059	0.09357304156293925	0.12259104991193753	0.15653723496587577	0.06218537782180899 6	0.055537297414012 08	0.12471871877593184
8	0.1452312226039673	0.18417164884601486	0.04626562350539095	0.122887667553388428	0.0930521442820087 3	0.18613687978769042	0.05217907832195859	0.062359359387965 92	0.10771637573597068
9	0.18488000997817722	0.14120210317619922	0.06139054961533828	0.094936479675660 13	0.12283438338228042	0.1544476737245997	0.06204562659589680 4	0.053858187867985 34	0.12440498598386272
10	0.1460181654714297	0.18443441129920424	0.04706736772539974	0.123043861683865 94	0.0945356075632298	0.1858747895805539	0.05148255790819989 5	0.062202492991931 36	0.10534074577618524
11	0.18495833113518567	0.14300789333082453	0.06147813709973474	0.0959405256054247 9	0.1230000679416677	0.15282565262010242	0.06195826319351796 5	0.052670372888092 62	0.12416075618544933
12	0.1465814447863547	0.18463094811563 05	0.04766929777694151	0.123152811015895 74	0.0956395605796539	0.18569141833130454	0.0509418842067008	0.062080378092724 67	0.10361225709479342
13	0.18501719432359287	0.14434955966222002	0.06154364937187683	0.096680261885278 52	0.12312005487982469	0.1516084943495491	0.06189713944376818	0.051806128547396 71	0.12397751753649285
14	0.14699281561322897	0.18477607491965348	0.04811651988740667	0.123232425547776 64	0.0964566503800459 3	0.18555829631987888	0.05053616478318303	0.061988758768246 42	0.10234229333057973
15	0.18506100318706575	0.1453424505068393	0.06159202497321782	0.097225930619432 62	0.12320823774710614	0.15070491665288255	0.06185276543995963	0.051171146665289866	0.12384152420820604
16	0.1472954210296236	0.18488314490912616	0.04844748350227976 6	0.123291119935900 33	0.0970604488119960 8	0.1854605286064179	0.05023497221762751 6	0.061920762104010302	0.10140611888291738
17	0.1850934511398022	0.146076181584819	0.06162771496970871 6	0.097628698808253924	0.12327327493766288	0.1500365053356274	0.06182017620213929 6	0.050703059441458 69	0.12374093830624232
18	0.14751857801475177	0.18496216948514088	0.0486920605282 73	0.123334454515432 17	0.0975064095695426 2	0.1853884624018612	0.05001216844520913	0.061870469153121 16	0.10071522788666781
19	0.18511743788671678	0.1466181246512949	0.06165405649504696	0.097926064123021 9	0.12332128375276305	0.14954264172679363	0.06179615413395373	0.050357613943333907	0.1236666232870749
20	0.14768328752087378	0.18502051100033406	0.0488727082170983	0.123366454505287 12	0.0978357402786092 5	0.18533527840639677	0.04984754724226454	0.061833311643537 45	0.10020516118559844
21	0.18513515705488717	0.14701834086336085	0.06167350366677802	0.098145632646262 55	0.12335673091942158	0.14917789034202167	0.06177842613546559	0.050102580592799 22	0.12361173777900303
22	0.1478048933915921	0.18506358814478452	0.04900611362112028 4	0.123390084478006 5	0.0980789299442515 6	0.18529601404326282	0.04972596344734055	0.061805868889501514	0.09982854404013977
23	0.18514824296835236	0.14731387639044344	0.06168786271492817	0.098307762769323 15	0.12338290495393142	0.1489085332646078	0.06176533801442094	0.049914272020069886	0.12357120690392245
24	0.14789468460301197	0.18509539618134468	0.04910462546348114	0.123407533466416 5	0.0982585068484272	0.18526702245583293	0.04963617775486926	0.061785603451961 23	0.09955044977493915
25	0.1851579062789341	0.14753210708855 65	0.06169846539378156	0.098427481625075 35	0.12340223212698981	0.14870963084334282	0.06175567415194431	0.049775224887469 58	0.12354127760390554
26	0.14796098678983743	0.18511888355025 45	0.04917736902951883	0.123420418156472 94	0.0983911098420565 1	0.18524561504687512	0.04956987694778009 36	0.061770638801952 77	0.09934510183525051
27	0.18516504194394634	0.14769325288049 29	0.0617062945167515	0.098515883850974 07	0.12341650359498796	0.1485627567953 52	0.06174853834895837	0.049672550917625256	0.12351917715091114
28	0.14800994515076865	0.18513622695222 76	0.04923108429349763	0.123429932445476 09	0.0984890262189846 7	0.18522980757239604	0.04952091893178399 4	0.061759588575455 57	0.09919346984940924
29	0.1851703110676126	0.14781224578657 95	0.06171207565407586	0.098581161493081 88	0.12342704187681391	0.14845430224007816	0.06174326919079868	0.049596734924704 62	0.12350285776625425
30	0.14804609675542682	0.18514903361502033	0.0492707485955265	0.123436957960944 43	0.0985613293420674 3	0.18521813509646334	0.04948476741335939	0.061751428883127124	0.09908150233806401
31	0.18517420188430012	0.14790011218503582	0.06171634453834011	0.098629363589509 33	0.12343482351881235	0.1483742175008 67	0.06173937365487776	0.049540751169032005	0.1234908072486149
32	0.1480727916900556	0.18515849025917966	0.04930003739501194	0.123442145720839 54	0.0986147191897666 1	0.1852095159512286	0.04945807250028899 6	0.061745403624307 45	0.09899882366932 1

Classement des nœuds par score PageRank Selon l'algorithme du pagerank normalisé voici le classement des noeuds suivant leur importance :

1. Noeud A : Score PageRank = 0.1851851848196014
2. Noeud F : Score PageRank = 0.14814815567295372
3. Noeud B : Score PageRank = 0.1481481398921749
4. Noeud I : Score PageRank = 0.12345679125572975
5. Noeud E : Score PageRank = 0.12345678939228966
6. Noeud D : Score PageRank = 0.09876542756967246
7. Noeud G : Score PageRank = 0.06172839542731173
8. Noeud C : Score PageRank = 0.06172839466062198
9. Noeud H : Score PageRank = 0.04938272130964219

— $\alpha = 0 : 85, \beta = 1$

$$A = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \quad D = \begin{bmatrix} 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}$$

On applique :

$$\mathbf{C}_p = \beta \left(\mathbf{I} - \alpha A^T D^{-1} \right)^{-1} \cdot \mathbf{1}$$

Classement des nœuds par score de centralité d'importance du voisinage :

1. Noeud F : Score = 3.0182520112307367
2. Noeud B : Score = 2.9161281634038696
3. Noeud A : Score = 2.9025537022347314

4. Noeud I : Score = 2.0970930360936673
5. Noeud E : Score = 1.9467738986783312
6. Noeud D : Score = 1.9438596075798722
7. Noeud H : Score = 1.096119192144492
8. Noeud G : Score = 1.0557798035563997
9. Noeud C : Score = 1.023440585077892

— $\alpha = 01, \beta = 1$

On applique :

$$\mathbf{C}_p = \beta \left(\mathbf{I} - \alpha A^T D^{-1}\right)^{-1} \cdot \mathbf{1}$$

On trouve que tous les noeuds ont même centralité égale β

b) c)

Ces questions ont été répondus dans le fichier `.ipynb` joint à ce rapport.

d) Algorithme pour implémenter le « PageRank » pour n'importe quel graphe.

Voici le code Python qu'on a réalisé pour l'algorithme général de PageRank que vous retrouverez dans le fichier `ipynb` :

```python
def page_rank(graph, alpha, beta):
    print("pour alpha =", alpha, "et pour beta =", beta)
    nodes = list(graph.keys())
    num_nodes = len(nodes)

    A = np.zeros((num_nodes, num_nodes))
    for i, node in enumerate(nodes):
        for neighbor in graph[node]:
            j = nodes.index(neighbor)
            A[i, j] = 1
    # print("la matrice A est:")
```

```
    # print(A)

    D = np.diag([len(graph[node]) for node in nodes])
    # print("la matrice D est:")
    # print(D)

    # Cr ation d'une matrice colonne remplie de 1
    matrice_colonne = np.ones((num_nodes, 1))

    inverse_D = np.linalg.inv(D)
    I = np.eye(num_nodes)
    transposee_A = np.transpose(A)
    J = np.dot(transposee_A, inverse_D)
    W = I - alpha * J
    Z = np.linalg.inv(W)
    K = np.dot(Z, matrice_colonne)
    C = beta * K

    # Cr ation du dictionnaire de r sultats
    resultats_pagerank_neighborhood = {nodes[i]: C[i, 0] for i in range(
num_nodes)}

    # Tri des n uds par score d croissant
    sorted_nodes = sorted(resultats_pagerank_neighborhood, key=
resultats_pagerank_neighborhood.get, reverse=True)

    # Affichage du classement
    if beta != 0:
        print("Classement des noeuds par score de centralite d'
importance du voisinage:")
        for rank, node in enumerate(sorted_nodes, start=1):
```

```
            print(f"{rank}. Noeud {node}: PageRank = {resultats_pagerank_neighborhood[node]}")
        return C
    else:
        # Calculate and normalize PageRank with alpha=1 and beta=0
        normalized_pagerank = calculate_and_normalize_pagerank(graph, alpha, beta)
        sorted_nodes = sorted(normalized_pagerank, key=normalized_pagerank.get, reverse=True)

        # Display normalized PageRank values
        print("Classement des noeuds par score de centralite d'importance suivant la methode des puissances avec 100 iterations :")
        # for node, score in normalized_pagerank.items():
        #     print(f"Node {node}: Normalized PageRank = {score}" )
        for rank, node in enumerate(sorted_nodes, start=1):
            print(f"{rank}. Noeud {node}: PageRank = {normalized_pagerank[node]}")
```

Listing 1 – Algorithme de PageRank

e) Choisir α et de β, appliquer à « Réseau 0 »

```
print(page_rank(reseau_0, alpha=1, beta=0))
print("\n")
print(page_rank(reseau_0, alpha=0.85, beta=1))
print("\n")
print(page_rank(reseau_0, alpha=0, beta=1))
```

Listing 2 – Application au réseau 0

Voici le résultat de l'algorithme de PageRank :

```
pour alpha = 1 et pour beta = 0
Classement des noeuds par score de centralite d'importance suivant la methode des puissance
```

```
1. Noeud A: PageRank = 0.19923558928997065
2. Noeud B: PageRank = 0.15984081277642417
3. Noeud E: PageRank = 0.13745222433062315
4. Noeud D: PageRank = 0.1305094771246591
5. Noeud F: PageRank = 0.1305094771246591
6. Noeud C: PageRank = 0.07566885937127565
7. Noeud I: PageRank = 0.07566885937127565
8. Noeud G: PageRank = 0.0617833649593475
9. Noeud H: PageRank = 0.029331335651765048
None
```

```
pour alpha = 0.85 et pour beta = 1
Classement des noeuds par score de centralite d'importance du voisinage:
1. Noeud F: PageRank = 9.942355567006283
2. Noeud B: PageRank = 9.397704812491918
3. Noeud A: PageRank = 9.150564074838288
4. Noeud I: PageRank = 7.306459090387651
5. Noeud E: PageRank = 6.333563330853176
6. Noeud D: PageRank = 6.284424236816782
7. Noeud H: PageRank = 4.105245113414751
8. Noeud G: PageRank = 3.817000743985113
9. Noeud C: PageRank = 3.6626830302060434
[[9.15056407]
 [9.39770481]
 [3.66268303]
 [6.28442424]
 [6.33356333]
 [9.94235557]
 [3.81700074]
```

```
 [4.10524511]
 [7.30645909]]
```

pour alpha = 0 et pour beta = 1

Classement des noeuds par score de centralite d'importance du voisinage:

1. Noeud A: PageRank = 1.0

2. Noeud B: PageRank = 1.0

3. Noeud C: PageRank = 1.0

4. Noeud D: PageRank = 1.0

5. Noeud E: PageRank = 1.0

6. Noeud F: PageRank = 1.0

7. Noeud G: PageRank = 1.0

8. Noeud H: PageRank = 1.0

9. Noeud I: PageRank = 1.0

```
[[1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]
 [1.]]
```

0.3.2 Centralité d'intermédiarité C_b

a) Ecrire un algorithme pour calculer le nombre de plus court chemin entre deux nœuds.

Voici le code Python pour l'algorithme qui calcule le nombre de plus court chemin entre deux nœuds :

```
def nombre_plus_courts_chemins(graph, s, t):
    # Transformer le graphe en un graphe networkx
    graph = nx.Graph(graph)

    if nx.has_path(graph, s, t):
        all_shortest_paths = list(nx.all_shortest_paths(graph, source=s,
    target=t))
        return len(all_shortest_paths)
    else:
        return 0  # Il n'existe pas de chemin entre s et t

# D finition des noeuds s et t
s = 'A'
t = 'I'

print(f"Le nombre de court chemin entre {s} et {t} est",
   nombre_plus_courts_chemins(reseau_0, s, t))
```

Listing 3 – Algorithme du nombre de plus court chemins

b) Algorithme pour calculer le nombre de plus court chemin entre deux nœuds et qui passe par un nœud v, donné

Voici le code Python pour l'algorithme qui calcule le nombre de plus court chemin entre deux nœuds :

```

```

```
def nombre_plus_courts_chemins_passant_par_v(graph, s, t, v):
    # Transformer le graphe en un graphe networkx
    graph = nx.Graph(graph)

    if nx.has_path(graph, s, t):
        all_shortest_paths = list(nx.all_shortest_paths(graph, source=s, target=t))

        # Filtrer les chemins passant par le n ud interm diaire v
        paths_passant_par_v = [path for path in all_shortest_paths if v in path]

        return len(paths_passant_par_v)
    else:
        return 0  # Il n'existe pas de chemin entre s et t
```

Listing 4 – Algorithme de Dijkstra avec une noeud intermidiare

c) Algorithme pour implémenter la centralité d'intermédiarité pour n'importe quel graphe.

Voici le code Python pour l'algorithme qui implémenter la centralité d'intermédiarité pour n'importe quel graphe :

```

def centralite_intermediarite(graph):
    # Transformer le graphe en un graphe networkx
    graph = nx.Graph(graph)

    nodes = list(graph.nodes)
    nombre_noeuds = len(nodes)

    # Initialiser les valeurs de centralit     0.0 pour tous les n uds
    valeurs_centralite = {node: 0.0 for node in nodes}
```

```
11
12      for node in nodes:
13          for source in nodes:
14              if node != source:
15                  for target in nodes:
16                      if node != target and source != target:
17                          nombre_plus_courts_chemins_st_v =
    nombre_plus_courts_chemins_passant_par_v(graph, source, target, node)
18                          nombre_plus_courts_chemins_st =
    nombre_plus_courts_chemins(graph, source, target)
19
20                          # Ajouter une v rification pour  viter  la
    division par z ro
21                          if nombre_plus_courts_chemins_st > 0:
22                              valeurs_centralite[node] +=
    nombre_plus_courts_chemins_st_v / nombre_plus_courts_chemins_st
23
24      # Facteur de normalisation
25      facteur_normalisation = (nombre_noeuds - 1) * (nombre_noeuds - 2)
26
27      # Normaliser les valeurs de centralit
28      for node in nodes:
29          valeurs_centralite[node] /= facteur_normalisation
30
31      return valeurs_centralite
```

Listing 5 – Algorithme de Dijkstra avec une noeud intermidiare

d) Application au réseau 0

```
1 # Calculate and display centralite_intermediarite
2 centralite = centralite_intermediarite(reseau_0)
```

```
print("Centralit  d'interm diarit  reseau_0:", centralite)
```

Listing 6 – Définition du graphe

Voici le résultat de l'algorithme d'intermédiarité :

```
Centralité d'intermédiarité reseau_0:
{'A': 0.6071428571428571,
'B': 0.3392857142857143,
'C': 0.0, 'D': 0.08928571428571429,
'E': 0.03571428571428571,
'F': 0.6071428571428571,
'G': 0.0,
'H': 0.0,
'I': 0.25}
```

0.3.3 Centralité de proximité C_c

a) Algorithme pour calculer le plus court chemin entre un nœud et les autres et retourne la moyenne de ces chemins.

Voici le code Python pour l'algorithme qui implémenter la centralité d'intermédiarité pour n'importe quel graphe :

```
def calcul_court_chemin(graph, start_node):
    # Initialisation des distances
    distances = {node: float('inf') for node in graph.nodes()}  #
   Initialise les distances    l'infini pour tous les n uds du graphe
    distances[start_node] = 0  # La distance du n ud de d part    lui-
   m me est mise    0
    queue = [(0, start_node)]  # Cr e une file de priorit  avec le
   n ud de d part et sa distance (0)

    # Parcours en largeur (BFS)
    while queue:  # Tant que la file de priorit  n'est pas vide
```

```
        current_distance, current_node = queue.pop(0)  # Prend le premier    lment    de la file (plus petite distance actuelle)
        for neighbor in graph.neighbors(current_node):  # Pour chaque voisin du  n  ud  actuel dans le graphe
            new_distance = current_distance + 1  # Calcule la nouvelle distance, avec un poids de 1 pour chaque  ar te
            if new_distance < distances[neighbor]:  # V rifie si cette nouvelle distance est plus courte que la connue  pr c demment
                distances[neighbor] = new_distance  # Met    jour la distance  vers le voisin
                queue.append((new_distance, neighbor))  # Ajoute le voisin    la file avec sa nouvelle distance

    return distances  # Retourne un dictionnaire contenant les distances les plus courtes depuis le  n  ud  de  d part  vers tous les autres  n  uds  du graphe
```

Listing 7 – code python de l'algorithme pour calculer le plus court chemin entre un nœud et les autres et retourne la moyenne de ces chemins.

b) Algorithme pour implémenter la centralité proximité pour n'importe quel graphe.

Voici le code Python pour l'algorithme qui implémenter la centralité d'intermédiarité pour n'importe quel graphe :

```
def centralite_proximite(graph):
    # Convert the dictionary to a NetworkX graph
    graph = nx.Graph(graph)
    closeness_centralities = {}

    for node in graph.nodes():
        distances = calcul_court_chemin(graph, node)
        reachable = sum(1 for dist in distances.values() if dist != float('inf'))
```

```
        total_distance = sum(dist for dist in distances.values() if dist
    != float('inf'))

        if total_distance > 0:
            closeness_centralities[node] = (reachable - 1) /
    total_distance
            closeness_centralities[node] *= (reachable - 1) / (len(graph
    .nodes()) - 1)
        else:
            closeness_centralities[node] = 0

    return {k: v for k, v in sorted(closeness_centralities.items(), key=
    lambda item: item[1], reverse=True)}
```

Listing 8 – code python de l'algorithme pour calculer le plus court chemin entre un nœud et les autres et retourne la moyenne de ces chemins.

c) Application au réseau 0

```

# Calculate and display closeness centrality
closeness = centralite_proximite(reseau_0)
print("Centralit  de proximit  du reseau_0:", closeness)
```

Listing 9 – code python de pour exécuterl'algorithme de centralité proximité.

Voici le résultat de l'algorithme de PageRank :

```
Centralité de proximité du reseau_0:
{
'A': 0.5714285714285714,
'F': 0.5333333333333333,
'B': 0.47058823529411764,
```

```
'D': 0.42105263157894735,
'I': 0.4,
'E': 0.36363636363636365,
'G': 0.36363636363636365,
'C': 0.3333333333333333,
'H': 0.2962962962962963
}
```

0.3.4 Cas des autres réseaux

Nous nous sommes basés sur les codes autrefois exécutés et il a ainsi fallu faire un appel de ces fonctions

Soc-karate

```
path1 = "data\soc-karate.txt"
aretes_karate= aretes(path1)
reseau_karate  = reseau("Reseau karate",aretes_karate )
reseau_karate = creer_graphe_a_partir_des_aretes(path1) #Cr ation du
    r seau

#Diff rentes centralit s
print("Centralit  PageRank : ", page_rank(reseau_karate, 0.85, 1))
print("\n Centralit  d'interm diarit  : ", centralite_intermediarite(
    reseau_karate))
print("\n Centralit  de proximit  : ", centralite_proximite(
    reseau_karate))
```

Listing 10 – soc-karate : application des algorithme

Soc-physicians

```
path2 = "data\soc-physicians.txt"
```

```
aretes_physicians= aretes(path2)
reseau_physicians  = reseau("Reseau physicians",aretes_physicians )
reseau_physicians = creer_graphe_a_partir_des_aretes(path2) #Cr ation
   du r seau

#Diff rentes centralit s
print("Centralit  PageRank : ", page_rank(reseau_physicians, 0.85, 1))
print("\n Centralit  d'interm diarit  : ", centralite_intermediarite(
   reseau_physicians))
print("\n Centralit  de proximit  : ", centralite_proximite(
   reseau_physicians))
```

Listing 11 – soc-physicians : application des algorithme

Soc-tribes

```
path3 = "data\soc-tribes.txt"
aretes_tribes= aretes(path3)
reseau_tribes  = reseau("Reseau tribes",aretes_tribes )
reseau_tribes = creer_graphe_a_partir_des_aretes(path3) #Cr ation du
   r seau

#Diff rentes centralit s
print("Centralit  PageRank : ", page_rank(reseau_tribes, 0.85, 1))
print("\n Centralit  d'interm diarit  : ", centralite_intermediarite(
   reseau_tribes))
print("\n Centralit  de proximit  : ", centralite_proximite(
   reseau_tribes))
```

Listing 12 – soc-tribes : application des algorithme

En conclusion, à travers l'analyse de différents réseaux, notamment "Reseau 0", "Soc-karate", "Soc-physicians", et "Soc-tribes", nous avons utilisé des mesures de centralité telles que le PageRank, la centralité d'intermédiarité, et la centralité de proximité pour caractériser et évaluer la

structure des réseaux.

L'algorithme de PageRank nous a permis de classer les nœuds en fonction de leur importance. Les résultats ont montré des variations significatives dans la distribution de l'importance des nœuds, mettant en évidence des structures hiérarchiques et des variations de connectivité.

En analysant la centralité d'intermédiarité, nous avons identifié les nœuds jouant un rôle crucial dans la connectivité des réseaux. Ces nœuds agissent souvent comme des ponts ou des médiateurs entre différentes parties du réseau.

La centralité de proximité nous a fourni des informations sur l'accessibilité globale des nœuds. Certains nœuds ont montré une proximité élevée avec d'autres, indiquant des relations étroites et une forte intégration dans le réseau.

Certains réseaux, comme "Soc-physicians", ont présenté toutefois des défis en raison de leur grande taille. Cela a rendu l'exécution de certains algorithmes plus coûteuse en termes de ressources computationnelles. Aussi, la visualisation des graphes peut être complexe, surtout lorsque le nombre de nœuds et d'arêtes est élevé. Des outils spécialisés peuvent être nécessaires pour une compréhension visuelle approfondie.

0.4 Resultats

Dans cette section nous présenterons la synthèse du travail effectué pour chaque réseau

0.4.1 Cas du réseau 0

Table 1 – Caractéristiques du Réseau 0

Caractéristique	Valeur	Noeud(s) Concerné(s)
Nombre de nœuds	9.000000	
Nombre d'arêtes	9.000000	
Degré		
Min	1.000000	C, G, H
Max	3.000000	A, B, F
Moyen	2.000000	
Centralité (PageRank)		
Min	3.662683	C
Max	9.942356	F
Moyenne	6.666667	
Centralité d'intermédiarité		
Min	0.000000	C, G, H
Max	0.607143	A, F
Moyenne	0.214286	
Centralité de proximité		
Min	0.296296	H
Max	0.571429	A
Moyenne	0.417034	

0.4.2 Cas du réseau Soc-karate

Table 2 – Caractéristiques du Réseau "Soc-karate"

Caractéristique	**Valeur**	**Noeud(s) Concerné(s)**
Nombre de nœuds	34.000000	
Nombre d'arêtes	78.000000	
Degré		
Min	1.000000	12
Max	17.000000	34
Moyen	4.588235	
Centralité (PageRank)		
Min	2.168009	12
Max	22.875015	34
Moyenne	6.666667	
Centralité d'intermédiarité		
Min	0.000000	8, 12, 13, 18, 22, 17, 15, 16, 19, 21, 23, 27
Max	0.437635	1
Moyenne	0.044006	
Centralité de proximité		
Min	0.284483	17
Max	0.568966	1
Moyenne	0.426480	

0.4.3 cas du réseau Soc-Physicians

TABLE 3 – Caractéristiques du Réseau "Soc-physicians"

Caractéristique	Valeur	Noeud(s) Concerné(s)
Nombre de nœuds	241.000000	
Nombre d'arêtes	923.000000	
Degré		
Min	1.000000	212
Max	28.000000	127
Moyen	7.659751	
Centralité (PageRank)		
Min	1.835939	212
Max	22.035103	127
Moyenne	6.666667	
Centralité d'intermédiarité		
Min	0.000000	95, 115, 117, 122, 154, 180, 181, 212, 222, 239
Max	0.025070	15
Moyenne	0.001982	
Centralité de proximité		
Min	0.047222	212
Max	0.241667	15
Moyenne	0.133212	

0.4.4 Cas de Soc-tribes

Table 4 – Caractéristiques du Réseau "Soc-tribes"

Caractéristique	Valeur	Noeud(s) Concerné(s)
Nombre de nœuds	16.000000	
Nombre d'arêtes	58.000000	
Degré		
Min	3.000000	4
Max	10.000000	6
Moyen	7.250000	
Centralité (PageRank)		
Min	3.425219	4
Max	8.790563	6
Moyenne	6.666667	
Centralité d'intermédiarité		
Min	0.002381	4
Max	0.075011	1
Moyenne	0.038690	
Centralité de proximité		
Min	0.500000	4
Max	0.750000	6
Moyenne	0.654621	

0.5 Analyse qualitative des résultats

Reseau 0 Le Réseau 0, de petite taille, est relativement simple. Les nœuds C, G, et H ont des centralités d'intermédiarité élevées, indiquant qu'ils sont positionnés sur des chemins cruciaux entre d'autres nœuds. La centralité de proximité de H suggère une accessibilité globale. La structure équilibrée suggère une robustesse aux perturbations.

0.5.1 Soc-Karate

Soc-karate présente une distribution de degrés inégale, reflétant des rôles sociaux différenciés. Certains individus (nœuds 8, 12, et 13) agissent comme des ponts, favorisant la cohésion. Les degrés élevés suggèrent une structure communautaire, et les centralités indiquent une hiérarchie sociale.

0.5.2 Soc-Physicians

Soc-physicians, plus complexe, montre une distribution de degrés plus large, ce qui peut refléter des niveaux différents d'influence. Les nœuds 95, 115, et 117, avec des centralités d'intermédiarité élevées, pourraient jouer un rôle clé dans la transmission d'informations.

0.5.3 Soc-Tribes

Le réseau Soc-tribes, bien que plus petit, présente une centralité d'intermédiarité significative pour le nœud 4, suggérant un rôle crucial dans la connectivité. La centralité de proximité élevée du nœud 6 suggère des relations étroites avec d'autres nœuds.

0.6 Conclusion

En conclusion, notre exploration des réseaux sociaux à l'aide de la théorie des graphes a fourni des insights précieux sur la structure et l'importance des nœuds dans différents réseaux. Nous avons utilisé des mesures telles que le PageRank, la centralité d'intermédiarité, et la centralité de proximité pour évaluer la connectivité et l'influence des nœuds. Malgré des défis liés à la taille des réseaux, cette analyse fournit une base solide pour comprendre les dynamiques sociales et peut être appliquée dans divers domaines tels que les médias sociaux et les réseaux professionnels. Des perspectives futures pourraient inclure l'optimisation des algorithmes pour les réseaux de grande taille et des comparaisons approfondies entre différents types de réseaux sociaux.