Etudes de graphes du Web

Projet de fin de sermestre

`20STIOCA`

[ISTY]

- UVSQ, Paris Saclay -

Olivier Benaben, Maxime Vincent,
Hugo Henrotte & Maxence Bringuier

04 Janvier 2021

Table des matières

1 Introduction

Le but du projet est d'étudier différents paramètres de graphes issus du web.Typiquement, ces graphes sont grands en termes de nombre d'arêtes et de sommets, et leur étude est souvent complexe, car elle requiert un nombre important de calculs. Dans ce projet, on s'intéressera à différents graphes du Web. D'une part on étudiera des graphes générés aléatoirement modélisant des graphes réels du web. D'autres part on utilisera des graphes issus de la base de données des grands graphes de Stanford, issus de différents réseaux sociaux. Typiquement,ces graphes sont représentés sous forme de fichier texte, dans lequel sont donnés leurs arêtes et sommets.

Nous avons donc créé un programme permettant de pouvoir répondre aux besoins de l'énonce. Nous pouvons y passer des arguments qui vont pouvoir nous aider à créer un graphe selon deux manières possibles (le modèle proposé par Edgar Gilbert et les graphes de Barabàsi-Albert).

Modèle proposé par Edgar Gilbert : chaque arête apparaît avec probabilité $p = \frac{1}{2}$.

Graphes de Barabàsi-Albert : probabilité degré(i)/Σ degrés.

Puis le programme propose aussi une option d'analyse comprenant :

- Nombre de sommets.
- Nombre d'arêtes.
- Degré maximal.
- Degré moyen.
- La distribution des degrés, sous forme de graphique.
- Le diamètre du graphe.

Nous avons decidés de développer en python pour se concentrer sur le coté algorithmique et sur l'analyse et la qualité des graphes.

Les graphes seront stockés sous forme de (.csv) et les rapport seront génerées en (.pdf) et bien sûr pour la compilation et l'exécution vous retrouverez la procédure à suivre dans le README.md dans la pièce jointe.

2 Génération de graphes aléatoires

2.1 Modèle de Edgar Gilbert

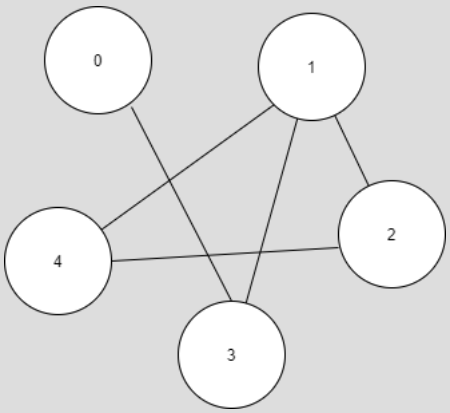
Pour la génération de graphes, la méthode de Edgar Gilbert est la plus intuitive.
Le principe est le suivant : pour chaque sommet on tire à pile ou face pour chaque autre sommet du graphe s'il existe une arête entre les deux.

Implémenté en python cela donne :

```
def genRandom_EG(vertices):

    graph = Graph()

    for i in range(nVertices):
        for j in range(i+1, nVertices):
            if random.choice([True, False]):
                graph.add_edge(i, j)

    return graph
```

Exemple de génération de graphe :
`python ./src/main.py --gen-graphe --method EG --vertices 5 --out eg_graph.csv`
permet de générer un graphe selon la méthode de Edgar Gilbert, et de le conserver dans un fichier au format CSV afin, par exemple, de l'analyser par la suite.
Avec comme nombre de sommets `5` on peut par exemple trouver un graphe tel que :

```
0 3
1 2
1 3
1 4
2 4
```



2.2 Modele de Barabási–Albert

C'est un modèle bien moins simple mais plus intéressant que le précédent. En effet on pense qu'il est plus représentatif des systèmes existants (*eg.* le world wide web, les réseaux sociaux, etc).

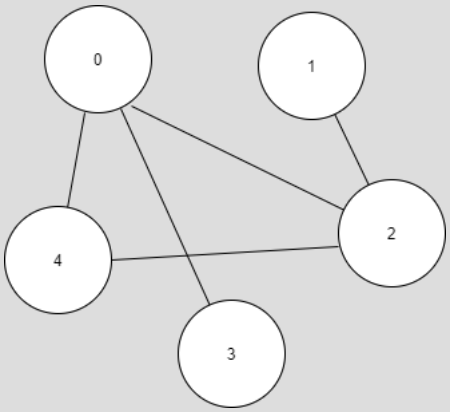
S'il est si efficace pour représenter les réseaux humains, c'est parce qu'il utilise un mécanisme d'attachement préférentiel. Cette notion peut être interprétée par la tendance à suivre les gens "connus" sur les réseaux sociaux, ou encore la tendance à ne naviguer sur internet qu'entre quelques mêmes sites web.

Implémenté en python cela donne :

```
def genRandom_BA(vertices, m):

    if m < 1 or  m >= vertices:
        return -1          # Barabási{Albert network must have 0 < m < nVertices

    graph = Graph()

    targets=list(range(m))  # Target nodes for new edges
    repeated_nodes=[]       # List of existing nodes,
                            # with nodes repeated once for each adjacent edge

    source = m              # Start adding the other n-m nodes.
                            # The first node is m.

    while source < vertices:
        for target in targets:
            graph.add_edge(source, target)  # Add edges to m nodes from the source.

        repeated_nodes.extend(targets)      # Add one node to the list for each
                                            #   new edge just created.
        repeated_nodes.extend(range(m))     # And the new node "source" has
                                            #   m edges to add to the list.

        # Now choose m unique nodes from the existing nodes
        # Pick uniformly from repeated_nodes
        targets = utils.random_subset(repeated_nodes, m)
        source += 1

    return graph;
```

Exemple de génération de graphe :
`python ./src/main.py --gen-graphe --method BA --vertices 5--m 2 --out ba_graph.csv`
permet de générer un graphe selon la méthode de Barabási–Albert, et de le conserver dans un fichier au format CSV afin, par exemple, de l'analyser par la suite.
On a spécifié un nombre de `sommets = 5`, et le paramètre `m = 2`. Dans le fichier `ba_graph.csv` on peut par exemple trouver un graphe tel que :

```
2 0
2 1
0 3
0 4
3 0
```



3 Analyse

3.1 Les fonctions d'analyse

L'analyse d'un graphe commence dans la méthode `analysis` de la classe `Graph`. Cette méthode fait appel aux routines suivantes :

- `compute_nVertices()`
- `compute_nEdges()`
- `compute_maxValency()`
- `compute_avgValency()`
- `computeValenceDistributionData()`
- `compute_diameter()`

Enfin, elle appellera la fonction `MakeReportPDF` afin de mettre en forme les résultats de l'analyse.

Description des méthodes :

`compute_nVertices()`
Cette méthode renvoie le nombre de sommets du graphe.
Dans un premier temps on ajoute à une liste de type `set` toutes les clés de la liste d'adjacence. Puis on parcours tous les sommets, et pour chaque sommet on parcours les sommets adjacents pour enfin les ajouter à la liste. Pour terminer on renvoie la taille de ce `set` (en python un `set` n'admet pas de doublons).

`compute_nEdges()`
Cette méthode renvoie le nombre d'arêtes du graphe.
Dans un premier temps on parcours tous les sommets, pour chacun d'entre eux on parcours les sommets adjacents ; pour chaque sommet adjacent on incrémente une variable `edgesCount`. Pour terminer on renvoie `edgesCount/2` car les arêtes sont dupliquées (`(0,2)` $\equiv$ `(2,0)`).

`compute_maxValency()`
Cette méthode renvoie le degré maximal des arêtes du graphe.
On parcours tout les sommets, pour chaque sommet on regarde si le nombre de voisins est supérieur aux autres sommets précédemment parcourus. On retourne la valeur la plus grande.

`compute_avgValency()`
Cette méthode renvoie le degré moyen des arêtes du graphe.
On parcours tout les sommets, pour chaque sommet on fait la somme du nombre de ces voisins avec les autres sommets. On renvoie la somme divisée par le nombre de sommet.

`computeValenceDistributionData()`
Cette méthode renvoie la distribution des degrés des arêtes en fonction de leur fréquences d'apparition dans le graphe.
Dans un premier temps on remplit un `dict` avec chaque sommet et son nombre

d'apparition dans la matrice d'adjacence, *ie.* sa valence.
Ensuite il nous faut parcourir ce `dict` pour en remplir un second, mais cette fois avec comme clefs les différentes valences trouvées plus tôt, associées à leur nombre d'apparition dans le premier tableau.
Finalement il ne reste plus qu'à en calculer le pourcentage d'apparition : nous avons le nombre total des valences en additionnant toutes les valeurs dans le dernier tableau, on calcule donc le pourcentage de chaque valence existante et on retourne un dictionnaire final mettant en rapport chaque valence présente dans le graphe et sa fréquence d'apparition en pourcentage.

`compute_diameter()`
Cette méthode renvoie le diamètre du graphe.
On parcours tous les sommets du graphe, pour chaque sommet on applique l'algorithme de Dijkstra vers tous les autres sommets.
L'algorithme renvoie la distance minimale du sommet à tous les autres sommets du graphe. Puis pour chaque sommet on récupère la longueur maximale parmi les longueurs minimales du sommet aux autres sommets.
Pour finir on renvoie la longueur maximale parmi les longueurs minimales de parcours entre chaque sommet du graphe. Cette valeur correspond au diamètre du graphe.
La complexité de cet algorithme correspond à la complexité de l'algorithme de Dijkstra multiplié par le nombre de sommets, donc :

$$O(n \cdot (a+n) \cdot log(n))$$

Avec `n` le nombre de sommets et `a` le nombre d'arêtes.

3.2 Options d'analyse

Pour exécuter l'analyse d'un graphe nous avons mis en place différentes options :

```
--analyse [FILE.csv] : lance l'analyse du graphe stocke dans file.csv
--debug : mode "verbose"
--diameter : calcule le diamètre du graphe - LENT
```

Exemple :

```
$ python3 ./src/main.py --analyze ./ba_graph.csv
$ python3 ./src/main.py --analyze ./ba_graph.csv --debug
$ python3 ./src/main.py --analyze ./ba_graph.csv --diameter
```

Proposer le calcul du diamètre en option s'est avéré nécessaire car la complexité de la méthode, et donc son temps d'exécution, est trop élevé pour être exécuté sur de très gros graphe.

3.3 Mise en forme des résultats

Nous avons choisis d'inclure la génération de rapports dans le programme principal pour plus de cohérence. Cette fonctionnalité est rendue accessible via l'ajout du flag `--analyse` dans l'appel du programme, en précisant le graphe (au format csv) à analyser.

Dans un premier temps il faut donc charger le graphe dans la mémoire, puis il est possible de lancer les fonctions d'analyse décrites ci-dessus.

Un rapport est alors généré au format `pdf` avec les informations calculées. On utilise les librairies `xhtml2pfd` pour formater le texte, et `matplotlib` pour afficher une courbe de la distribution des degrés des arêtes en fonction de leur fréquence d'apparition dans le graphe.

4 Conclusion

L'issue de ce projet était d'analyser certains graphes issus de la base de donnée des grands graphes de Stanford. L'intérêt de ces graphes étant qu'ils proviennent de véritables réseaux, on peut maintenant les comparer avec ceux que nous avons pu générer, et particulièrement les méthodes utilisées.
Nous avons donc générés deux grands graphes avec des tailles du même ordre que ceux donnés par Stanford, un avec la méthode de Edgar Gilbert et un avec celle de Barabasi-Albert :

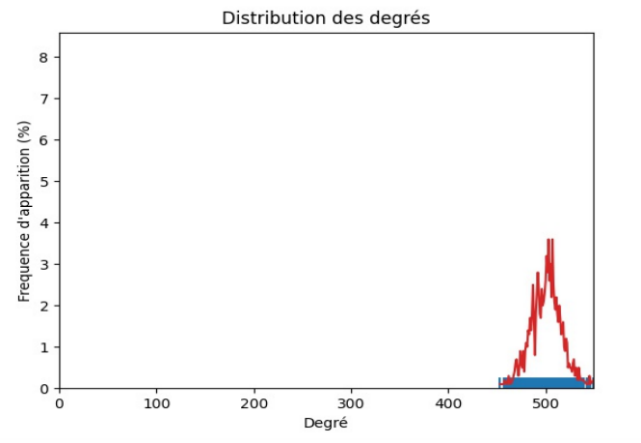


FIGURE 1 – Distribution des degrés pour le modèle de Edgar Gilbert

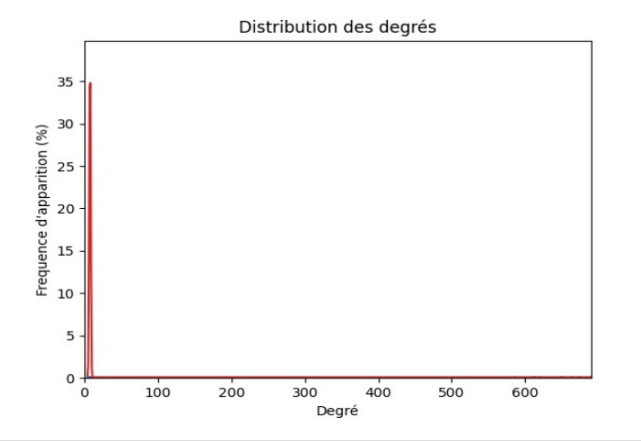


FIGURE 2 – Distribution des degrés pour le modèle de Barabasi-Albert

On remarque rapidement que le modèle de Edgar Gilbert est très loin de la réalité en terme de connections, alors que, même si notre implémentation n'est pas parfaite, le modèle de Barabasi-Albert donne des résultats assez proche de ce qu'on peut retrouver dans les graphe de Facebook, Wikipedia ou encore Twitch par exemple.

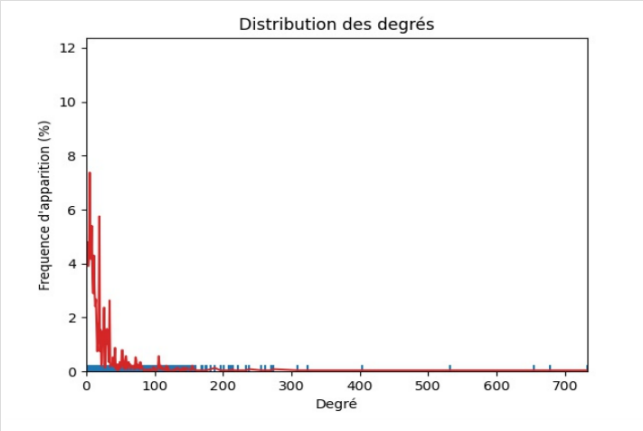


FIGURE 3 – Distribution des degrés pour le graphe de Wikipedia

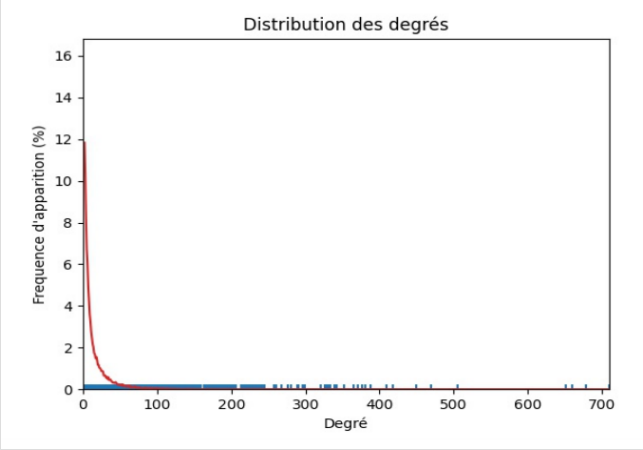


FIGURE 4 – Distribution des degrés pour le graphe de Facebook

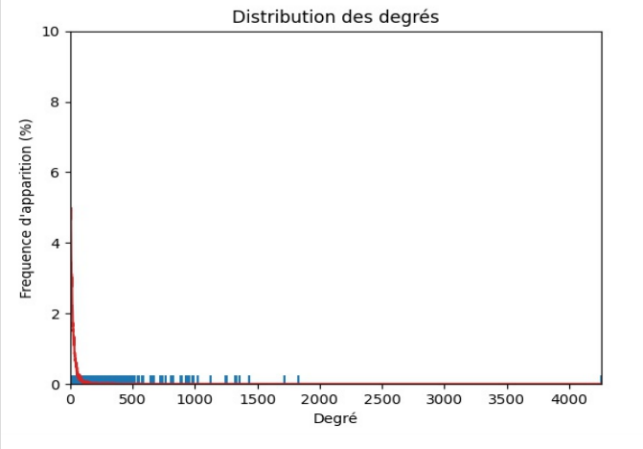


FIGURE 5 – Distribution des degrés pour le graphe de Twitch

5 Annexe

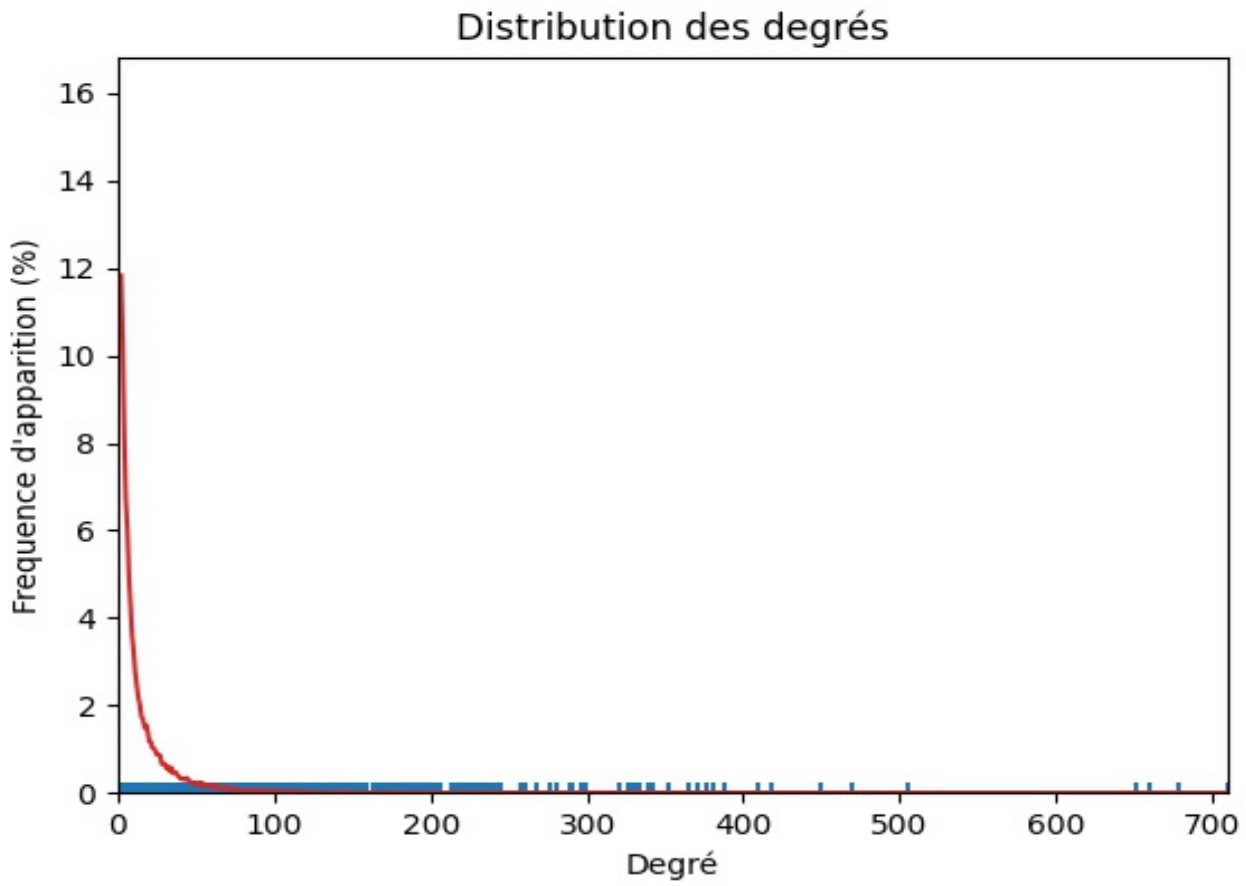
Les pages suivantes sont les rapports détaillés générés pour les grands graphes de Stanford.

Des rapports pdf pour des graphes générés sont trouvables dans le dossier `rapports/` à la racine de l'archive du projet.

Les instructions pour générer et analyser des graphes sont disponibles dans le `README`.

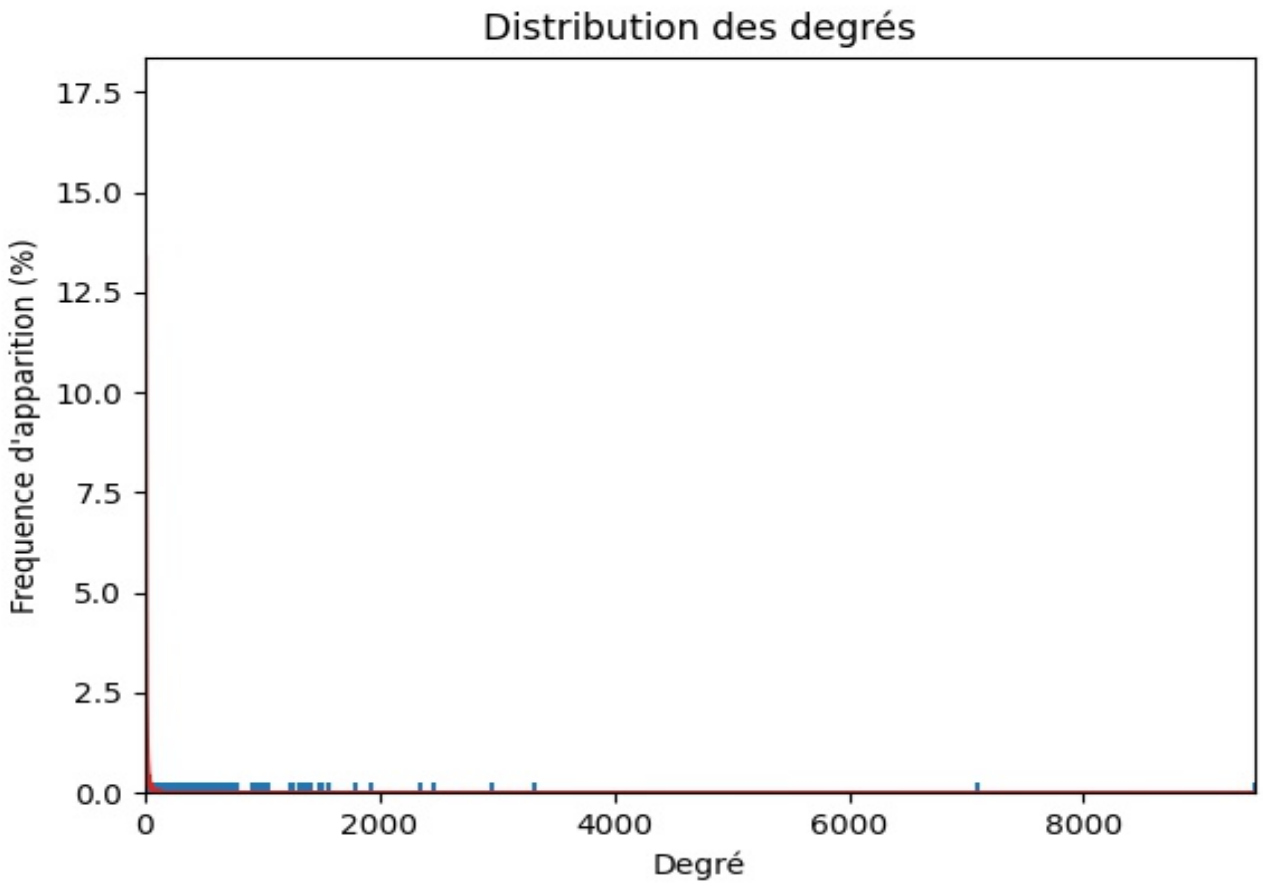
GRAPH REPORT : ./assets/FacebookSites.csv

- **Nombre de sommets :** 22470
- **Nombre de d'arretes :** 170912
- **Degré maximal :** 709
- **Degré moyen :** 15
- **Diamètre du graph :** not computed



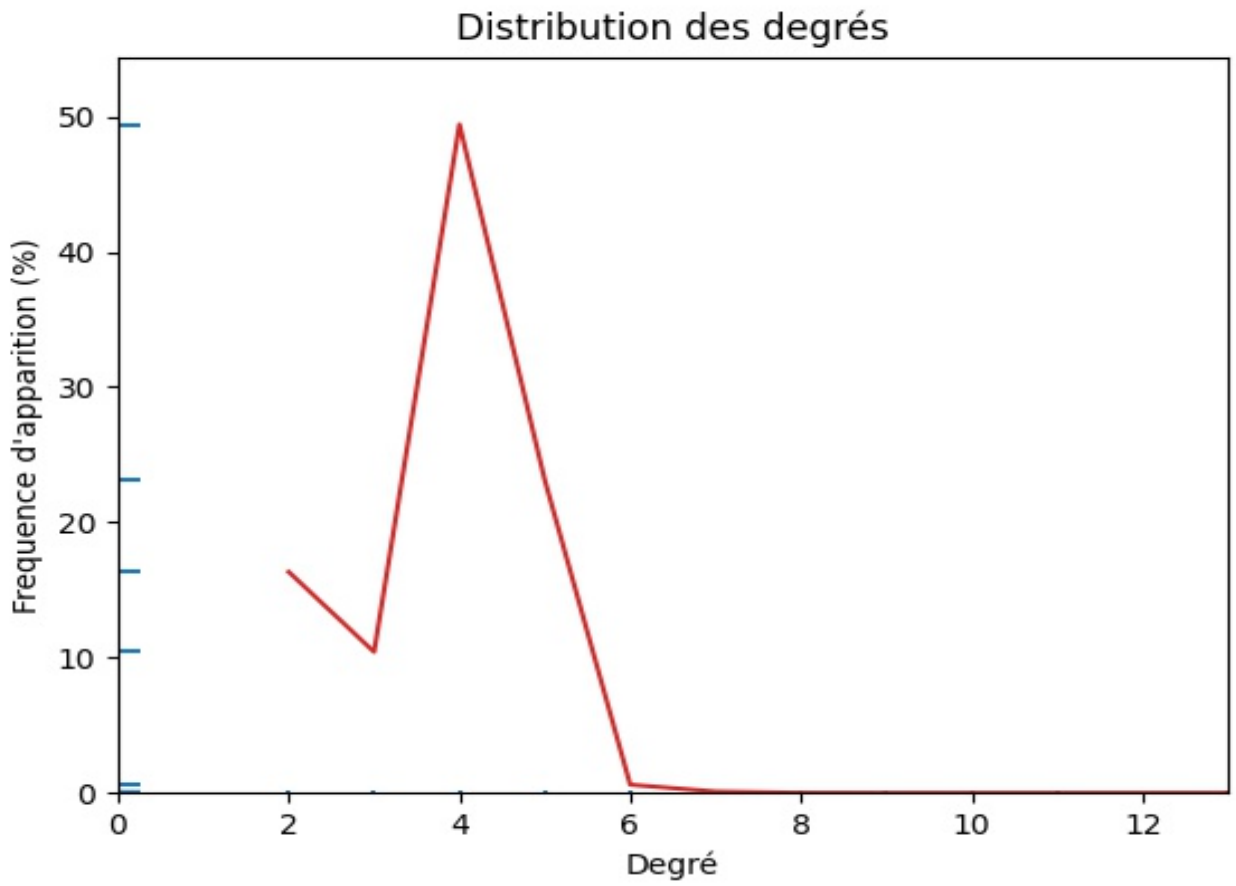
GRAPH REPORT : ./assets/GitHub.csv

- **Nombre de sommets :** 37700
- **Nombre de d'arretes :** 289003
- **Degré maximal :** 9458
- **Degré moyen :** 15
- **Diamètre du graph :** not computed



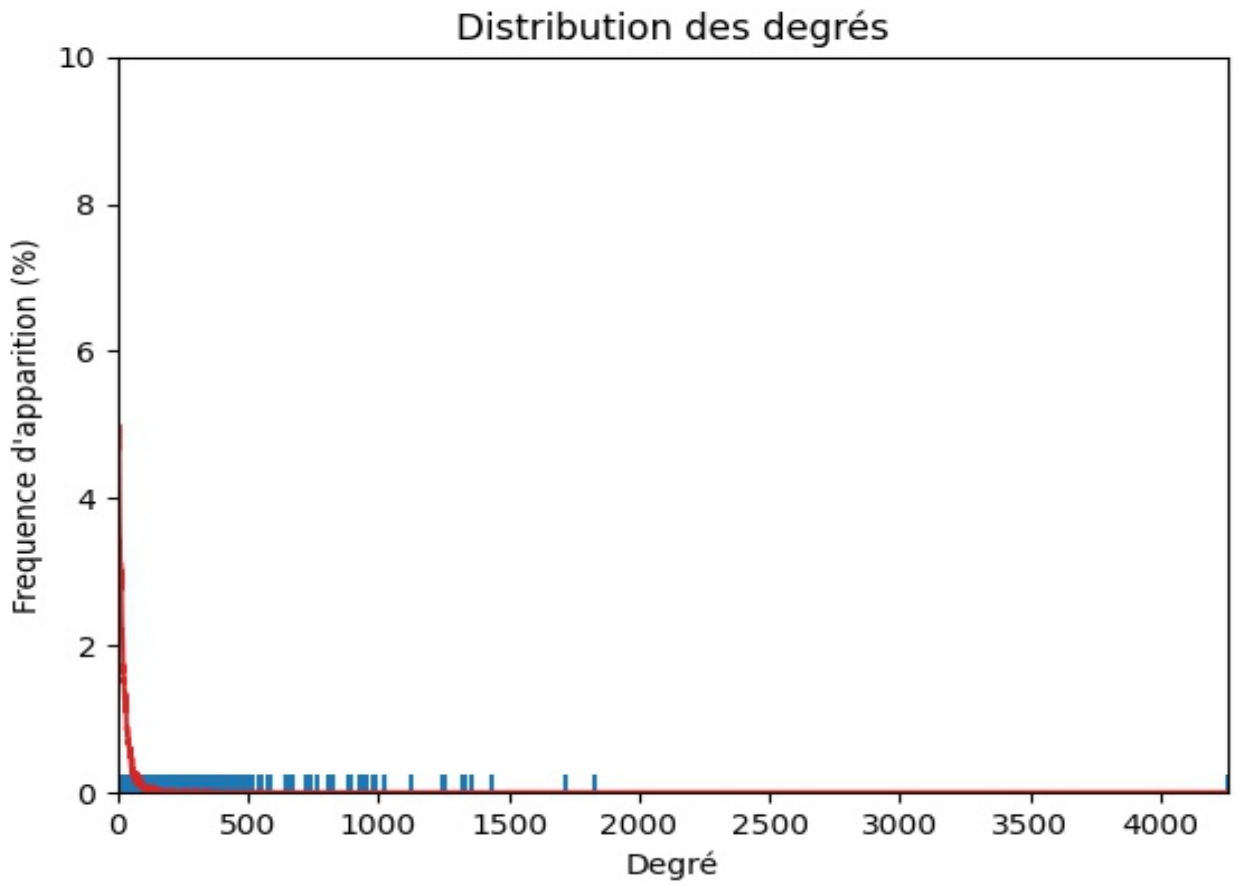
GRAPH REPORT : ./assets/RoadNetwork.txt

- **Nombre de sommets :** 1965206
- **Nombre de d'arretes :** 2766607
- **Degré maximal :** 12
- **Degré moyen :** 2
- **Diamètre du graph :** not computed



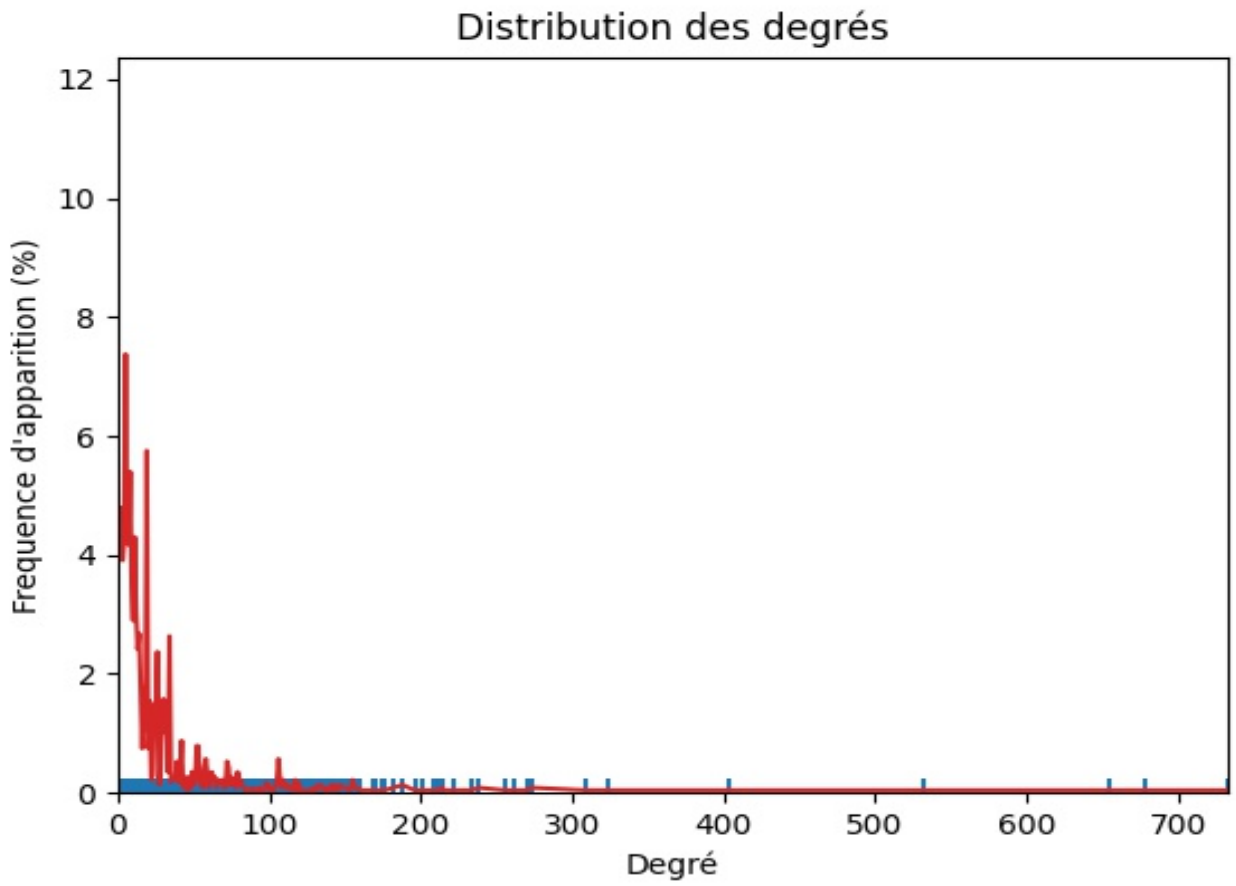
GRAPH REPORT : ./assets/twitchDE.csv

- **Nombre de sommets :** 9498
- **Nombre de d'arretes :** 153138
- **Degré maximal :** 4259
- **Degré moyen :** 32
- **Diamètre du graph :** not computed



GRAPH REPORT : ./assets/Wikipedia1.csv

- **Nombre de sommets :** 2277
- **Nombre de d'arretes :** 31396
- **Degré maximal :** 732
- **Degré moyen :** 27
- **Diamètre du graph :** 11



GRAPH REPORT : ./assets/Wikipedia2.csv

- **Nombre de sommets :** 11631
- **Nombre de d'arretes :** 170845
- **Degré maximal :** 3546
- **Degré moyen :** 29
- **Diamètre du graph :** not computed