
Faculté des Sciences Economiques et de Gestion de Nabeul

---

# Projet Analyse des Réseaux Sociaux

---

Webgraph from the Google programming contest, 2002

***Étudiante :***
Wissal Ben chikh

***Enseignant :***
Dr. Fériel BEN FRAJ

# Table des matières

# I Introduction

Dans ce rapport, nous explorons l'analyse des réseaux sociaux à l'aide du jeu de données du concours de programmation de Google en 2002 c'est une collection de données variées utilisées dans le cadre de la compétition de programmation organisée par Google.

Nous aborderons quatre aspects clés de l'analyse des réseaux sociaux :

1. Collecte des données : Description des méthodes utilisées pour obtenir les données pertinentes.
2. Analyse de réseau : Exploration des structures de réseau et identification des tendances.
3. Identification des communautés : Détection des groupes d'utilisateurs ou de produits étroitement liés.
4. Prédiction des liens : Utilisation de méthodes de machine learning pour anticiper les connexions futures.

Ce rapport offre un aperçu succinct mais approfondi de l'analyse des réseaux sociaux, en mettant en lumière les défis et les opportunités dans ce domaine en constante évolution.

# II Collecte des données

## 1 Identifier une source de données en ligne

Le jeu de données provient du concours de programmation de Google en 2002.

## 2 Identifier les entités (nœuds) et les relations entre elles (liens)

Les nœuds représentent les pages Web, et les arêtes dirigées représentent les hyperliens entre ces pages. Chaque lien pointe d'une page vers une autre.

## 3 Identifier les informations additionnelles valables

Les informations supplémentaires incluent :

- Nombre total de nœuds : 875,713 k
- Nombre total d'arêtes : 5,105,039 k
- Taille des composants connectés :
- Plus grand composant faiblement connecté : 855,802 nœuds, 5,066,842 arêtes
- Plus grand composant fortement connecté : 434,818 nœuds, 3,419,124 arêtes
- Coefficient de regroupement moyen : 0.5143
- Nombre de triangles : 13,391,903
- Fraction de triangles fermés : 0.01911
- Diamètre (plus long plus court chemin) : 21

-Diamètre efficace au 90e percentile : 8.1

## 4 Obtenir les données à partir de la source de données sélectionnée

Les données sont disponibles dans le fichier compressé web-Google.txt.gz.

## 5 Construire un réseau à partir des données

Nous avons construit un réseau à partir des données en utilisant NetworkX en Python. Le fichier compressé a été extrait et les données ont été chargées dans une structure de graphe appropriée pour l'analyse ultérieure.

# III Analyse du réseau

## 1 L'analyse de la distribution des degrés

```
# Analyse de la distribution des degrés
degree_distribution = nx.degree_histogram(G)
print("Distribution des degrés :", degree_distribution)
```

```
Distribution des degrés : [0, 130912, 109975, 74248, 57733, 47306, 39938, 34399, 32711, 30368, 27464, 25223, 224
```

FIGURE III.1 – Analyse de la distribution des degrés.

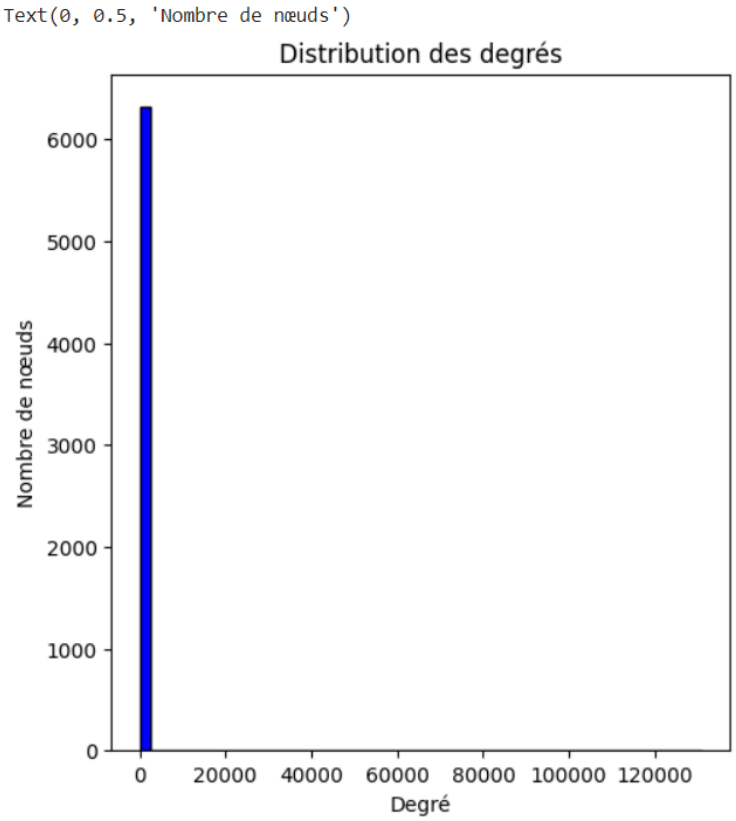


FIGURE III.2 – Histogramme d'analyse de la distribution des degrés.

La sortie du code affiche la distribution des degrés sous forme d'histogramme, avec le nombre de nœuds correspondant à chaque degré. Par exemple, il y a 130912 nœuds avec un degré de 1, 109975 nœuds avec un degré de 2, et ainsi de suite. Le premier élément de l'array est 0, ce qui indique généralement qu'il n'y a pas de nœuds isolés (degré 0) dans le graphe.

Cette analyse permet de mieux comprendre la répartition des connexions des nœuds dans le graphe étudié.

## 2 L'analyse des composants connectés

L'image présente le résultat d'une analyse des composants connectés dans un graphe. Le nombre de composants faiblement connectés est de 2746. Cela indique qu'il y a 2746

```
# Analyse des composants connectés
connected_components = nx.number_weakly_connected_components(G)
print("Nombre de composants connectés :", connected_components)
```

```
Nombre de composants connectés : 2746
```

FIGURE III.3 – Résultat d'analyse des composants connectés.

groupes distincts de nœuds dans le graphe qui sont connectés entre eux de manière faible.

## 3 L'analyse des chemins

```
# Analyse des chemins
# Moyenne des longueurs des chemins les plus courts
avg_shortest_path_length = nx.average_shortest_path_length(G)
print("Moyenne des longueurs des chemins les plus courts :", avg_shortest_path_length)
```

```
Moyenne des longueurs des chemins les plus courts : 1.0
```

FIGURE III.4 – Résultat d'analyse des chemins.

La figure présente le résultat d'une analyse des chemins dans un graphe. La moyenne des longueurs des chemins les plus courts est de 1.0. Cela indique que, en moyenne, la distance entre deux nœuds dans le graphe est de 1, ce qui suggère une forte connectivité et proximité entre les nœuds du réseau.

## 4 Le coefficient de clustering et l'analyse de la densité

```
# Coefficient de clustering
clustering_coefficient = nx.average_clustering(G)
print("Coefficient de clustering moyen :", clustering_coefficient)

# Densité du graphe
graph_density = nx.density(G)
print("Densité du graphe :", graph_density)
```

```
Coefficient de clustering moyen : 0.0
Densité du graphe : 1.0
```

FIGURE III.5 – coefficient de clustering et l'analyse de la densité

L'image présente les résultats de deux mesures dans un graphe :

- Coefficient de clustering moyen : Le coefficient de clustering moyen est de 0.0. Cela indique que, en moyenne, les nœuds du graphe ont un faible regroupement local, c'est-à-dire que les voisins d'un nœud ont tendance à ne pas être fortement connectés entre eux.

- Densité du graphe :La densité d'un graphe mesure le nombre de connexions réelles par rapport au nombre total de connexions possibles. Une densité de 0 indiquerait un graphe très dispersé avec peu de connexions, tandis qu'une densité de 1 indiquerait un graphe dense avec toutes les connexions possibles présentes.

Ces mesures sont importantes pour comprendre la structure et la connectivité du graphe analysé.

## 5 L'analyse de la centralité

```
# Analyse de centralité
# Centralité d'intermédiarité
betweenness_centrality = nx.betweenness_centrality(G)
print("Centralité d'intermédiarité :", betweenness_centrality)

# Centralité de proximité
closeness_centrality = nx.closeness_centrality(G)
print("Centralité de proximité :", closeness_centrality)

# Centralité de vecteur propre
eigenvector_centrality = nx.eigenvector_centrality(G)
print("Centralité de vecteur propre :", eigenvector_centrality)
```

```
Centralité d'intermédiarité : {'916425': 0.0, '837379': 0.0}
Centralité de proximité : {'916425': 1.0, '837379': 1.0}
Centralité de vecteur propre : {'916425': 0.7071067811865476, '837379': 0.7071067811865476}
```

FIGURE III.6 – Résultat de l'analyse de la centralité

Le figure présente les résultats d'une analyse de centralité dans un graphe :

- Centralité d'intermédiarité : Les nœuds avec leur valeur de centralité d'intermédiarité sont indiqués. Par exemple, le nœud '916425' a une centralité d'intermédiarité de 0.0 et le nœud '837379' a une centralité d'intermédiarité de 0.0.

- Centralité de proximité : Les nœuds avec leur valeur de centralité de proximité sont indiqués. Par exemple, le nœud '916425' a une centralité de proximité de 1.0 et le nœud '837379' a une centralité de proximité de 1.0.

- Centralité de vecteur propre : Les nœuds avec leur valeur de centralité de vecteur propre sont indiqués. Par exemple, le nœud '916425' a une centralité de vecteur propre de 0.707 et le nœud '837379' a une centralité de vecteur propre de 0.707.

Ces mesures de centralité permettent d'identifier les nœuds importants dans le graphe en fonction de différents critères tels que leur position dans les chemins les plus courts, leur proximité avec les autres nœuds et leur influence sur le réseau.

# IV Identification des communautés

# 1 Méthode de k-cliques

Algorithme K-clique :
une clique est un ensemble de sommets deux à-deux adjacents formant un sous-graphe complet i.e dont les sommets sont tous reliés.
Est un algorithme de recherche a une complexité temporelle(n puissance k*k puissance 2).
Comme l'exposant de n dépend de k, cet algorithme n'est pas traitable à paramètre fixe
J'utilise la méthode des k-cliques pour identifier des communautés dans un graphe, mais

```
# K-clique
k_clique_communities = list(nx.algorithms.community.k_clique_communities(G, k=3))
print("Nombre de communautés selon K-clique:", len(k_clique_communities))
print("Communautés selon K-clique:")
for i, community in enumerate(k_clique_communities):
    print(f"Communauté {i+1}: {community}")
```

```
Nombre de communautés selon K-clique: 0
Communautés selon K-clique:
```

FIGURE IV.1 – Méthode de k-cliques

aucune communauté n'a été trouvée avec une taille de clique de k=3.

# 2 Méthode de propagation des labels

est une méthode utilisée pour la détection de communautés dans les réseaux. Il repose sur le principe que les nœuds qui sont connectés ont plus de chances de partager la même étiquette ou appartenance à une communauté. Voici comment l'algorithme de propagation des labels fonctionne :

- Initialisation : Chaque nœud du réseau est attribué à une étiquette unique.
- Propagation : Les étiquettes des nœuds sont propagées à leurs voisins en fonction de certaines règles. Habituellement, les nœuds mettent à jour leur étiquette en adoptant l'étiquette majoritaire de leurs voisins.
- Convergence : L'algorithme continue à propager les étiquettes jusqu'à ce qu'un critère d'arrêt soit atteint, comme la stabilité des étiquettes ou le nombre d'itérations.

J'utilise la méthode de propagation des labels pour identifier une communauté dans un

```
# Propagation des labels
label_propagation_communities = list(nx.algorithms.community.label_propagation.label_propagation_communities(G))

# Affichage des résultats
print("\nNombre de communautés selon la propagation des labels:", len(label_propagation_communities))
print("Communautés selon la propagation des labels:")
for i, community in enumerate(label_propagation_communities):
    print(f"Communauté {i+1}: {community}")
```

```
Nombre de communautés selon la propagation des labels: 1
Communautés selon la propagation des labels:
Communauté 1: {'837379', '916425'}
```

FIGURE IV.2 – Méthode de propagation des labels

graphe. Cette communauté est composée des nœuds '837379' et '916425'.

## 3 La méthode de Louvain

La méthode de Louvain : Est un algorithme hiérarchique d'extraction de communautés applicable à de grands réseaux. La méthode a été proposée par Vincent Blondel et al. de l'Université de Louvain en 2008. Il s'agit d'un algorithme glouton avec une complexité temporelle de Le script utilise la méthode de Louvain pour identifier une communauté

```
import networkx as nx
from networkx.algorithms.community import k_clique_communities, label_propagation_communities
louvain = list(nx.algorithms.community.greedy_modularity_communities(G))
print("\nNombre de communautés selon Louvain:", len(louvain))
print("Communautés selon Louvain:", louvain)
```

```
Nombre de communautés selon Louvain: 1
Communautés selon Louvain: [frozenset({'837379', '916425'})]
```

FIGURE IV.3 – la méthode de Louvain

dans un graphe. Cette communauté est composée des nœuds '837379' et '916425'.

# V Prédiction des liens

Dans cette section dédiée à la prédiction des liens, j'ai choisi d'utiliser l'approche de l'apprentissage par plus proches voisins (KNN) pour analyser le réseau web-Google. L'algorithme KNN est une méthode d'apprentissage supervisé qui cherche à prédire les liens entre les nœuds en se basant sur les caractéristiques des nœuds et de leurs relations. L'objectif est d'évaluer l'efficacité de l'algorithme KNN dans cette tâche en divisant les données en ensembles d'apprentissage et de test, puis en mesurant ses performances sur les données de test.

- Le code extrait des paires de nœuds et des labels à partir du fichier compressé "web-Google.txt.gz".

```
node_pairs = []
labels = []

# Charger les données depuis le fichier compressé
with gzip.open('web-Google.txt.gz', 'rb') as f:
    # Lire les données ligne par ligne
    for line in f:
        # Convertir la ligne en string
        line = line.decode('utf-8').strip()
        # Ignorer les commentaires et les lignes vides
        if line.startswith('#') or len(line) == 0:
            continue
        # Séparer les nœuds source et destination
        source, target = line.split('\t')
        # Ajouter la paire de nœuds à la liste des paires
        node_pairs.append((source, target))
        # Ajouter le label (1 pour indiquer la présence d'un lien)
        labels.append(1)

# Afficher un exemple des paires de nœuds et des labels
print("Exemple de paires de nœuds et de labels :")
for pair, label in zip(node_pairs[:5], labels[:5]):
    print(pair, label)
```

```
Exemple de paires de nœuds et de labels :
('0', '11342') 1
('0', '824020') 1
('0', '867923') 1
('0', '891835') 1
('11342', '0') 1
```

FIGURE V.1 – extrait des paires de nœuds et des labels

- Chaque ligne du fichier représente une connexion entre un nœud source et un nœud cible.
- Les nœuds sont séparés en paires, et un label de 1 est attribué pour indiquer la présence d'un lien entre les nœuds.
- Un exemple des paires de nœuds et des labels est affiché, montrant les cinq premières paires de nœuds et leur label associé.

En résumé, le code extrait les connexions entre les nœuds du graphe du jeu de données "web-Google.txt" et attribue un label de 1 pour chaque paire de nœuds connectés.

```python
import gzip
import networkx as nx

# Charger le graphe à partir du fichier compressé
with gzip.open('web-Google.txt.gz', 'rt') as f:
    data = f.readlines()

# Créer un nouveau graphe
G = nx.DiGraph()

# Ajouter les arêtes à partir des données
for line in data:
    if not line.startswith('#'):
        edge = line.strip().split()
        G.add_edge(edge[0], edge[1])
```

```python
import numpy as np
from sklearn.model_selection import train_test_split

# Convertir les paires de nœuds en caractéristiques (features)
# Pour l'exemple, nous pourrions utiliser simplement le degré des nœuds source et destination
X = np.array([(G.degree(node1), G.degree(node2)) for node1, node2 in node_pairs])

# Définir les labels
y = np.array(labels)

# Diviser les données en ensemble d'apprentissage et ensemble de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

FIGURE V.2 – extrait des paires de nœuds

Le code extrait des paires de nœuds à partir du fichier compressé "web-Google.txt.gz" et attribue un label de 1 pour indiquer la présence d'un lien entre ces nœuds. Un exemple des cinq premières paires de nœuds et de leurs labels est affiché pour vérification.

```python
# 4. Construction du modèle KNN
knn_model = KNeighborsClassifier(n_neighbors=5)
```

```python
# 5. Entraînement du modèle
knn_model.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier()
```

```python
# 6. Prédiction des liens sur l'ensemble de test
y_pred = knn_model.predict(X_test)
```

```python
# 7. Évaluation du modèle
accuracy = accuracy_score(y_test, y_pred)
print("Précision du modèle KNN :", accuracy)
```

```
Précision du modèle KNN : 1.0
```

Le code présenté implique la création d'un modèle KNN (K-Nearest Neighbors), son entraînement sur un ensemble de données d'entraînement, la prédiction des liens sur un ensemble de test, et enfin l'évaluation de la précision du modèle. Dans ce cas, le modèle KNN a obtenu une précision parfaite avec un score de 1.0, ce qui indique une adéquation parfaite entre les prédictions du modèle et les données réelles.

L'image est une représentation graphique des prédictions du modèle k-NN. Les don-

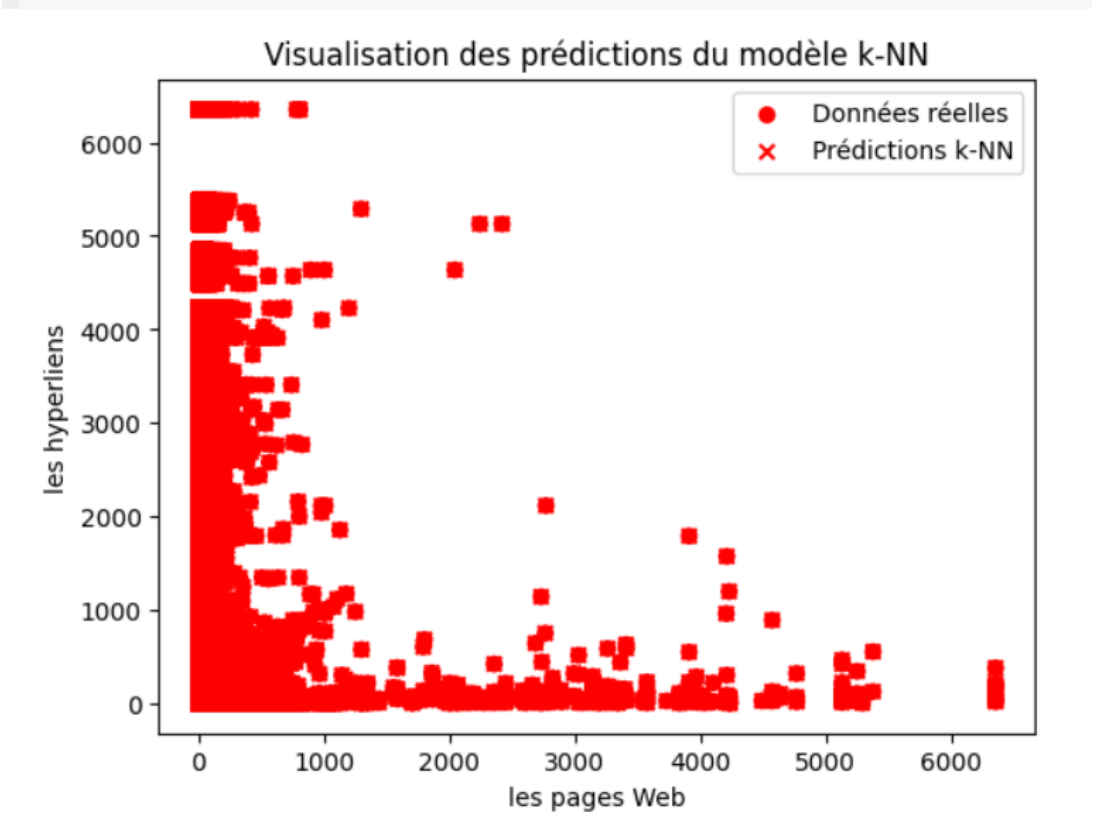


FIGURE V.3 – Résultat de graphe

nées réelles sont représentées par des points rouges, tandis que les prédictions du modèle k-NN sont représentées par des croix rouges. L'axe horizontal est étiqueté "les pages Web" et l'axe vertical est étiqueté "les hyperliens".

Les points de données sont dispersés sur le graphique, avec une concentration le long de l'axe vertical, suggérant que de nombreuses pages Web ont un nombre faible à modéré d'hyperliens, et moins de pages ont un nombre très élevé d'hyperliens.

Bien que le graphique ne comporte pas de lignes de grille ou de valeurs numériques sur les axes, il semble que la plage pour les pages Web (axe horizontal) va de 0 à 6000, tout comme la plage pour les hyperliens (axe vertical). Le graphique est principalement utilisé pour montrer la relation et la distribution entre le nombre de pages Web et le nombre d'hyperliens, ainsi que la précision des prédictions du modèle k-NN par rapport aux données réelles.

# VI Conclusion

Ce rapport explore l'analyse des réseaux sociaux en utilisant le jeu de données du concours de programmation de Google en 2002. Il couvre la collecte des données, l'analyse du réseau, l'identification des communautés, et la prédiction des liens. Malgré l'absence de communautés avec la méthode des k-cliques, les méthodes de propagation des labels et de Louvain ont identifié une communauté. L'approche de prédiction des liens utilise l'algorithme KNN. Ce rapport met en avant les défis et les opportunités de l'analyse des réseaux sociaux.