Rapport d'Analyse de Graphes avec SNAP

Amayes Djermoune

January 5, 2024

1 Introduction

Ce rapport présente une analyse approfondie de graphes à l'aide de la bibliothèque SNAP. L'analyse couvre à la fois les graphes orientés et non orientés, incluant des aspects tels que le nombre de sommets, le nombre d'arêtes, le degré maximal, les composantes connexes, les cycles avec 3 sommets, la dégénérescence, et la distribution des degrés.

2 Analyse de Graphe

2.1 Analyse de Graphe Orienté

Nous avons développé un algorithme pour analyser les graphes non orientés. L'algorithme utilise la bibliothèque SNAP pour effectuer les calculs suivants :

- Nombre de sommets
- Nombre d'arêtes
- Degré sortant maximal
- Nombre de composantes fortement connexes
- Nombre de cycles avec 3 sommets
- Dégénérescence

L'algorithme est basé sur la suppression itérative des nœuds de degré sortant minimum pour calculer la dégénérescence :

```
#include "Snap.h"

// Structure pour stocker les résultats d'analyse d'un graphe
struct GraphResult {
    TStr GraphName;
    int NumNodes;
    int NumEdges;
```

```
    int MaxDegree;
    int NumConnComp;
    int NumCycles3;
    int MaxDegeneracy;  // Ajout du champ pour stocker la dégénérescence
};

void AnalyzeDirectedGraph(const PNGraph& Graph, const TStr& GraphName, GraphResult& Result)
    TIntV NIdV;
    TCnComV CnComV;

    // 1. Type de graphe: Orienté
    //printf("1. Type de graphe: Orienté\n");

    // 2. Nombre de sommets
    const int NumNodes = Graph->GetNodes();
    //printf("2. Nombre de sommets: %d\n", NumNodes);

    // 3. Nombre d'arêtes
    const int NumEdges = Graph->GetEdges();
    //printf("3. Nombre d'arêtes: %d\n", NumEdges);

    // 4. Degré sortant maximal
    int MaxOutDegree = 0;
    for (TNGraph::TNodeI NI = Graph->BegNI(); NI < Graph->EndNI(); NI++) {
        MaxOutDegree = TMath::Mx(MaxOutDegree, NI.GetOutDeg());
    }
    //printf("4. Degré sortant maximum: %d\n", MaxOutDegree);

    // 5. Nombre de composantes fortement connexes
    TSnap::GetSccs(Graph, CnComV);
    const int NumStrongConnComp = CnComV.Len();
    //printf("5. Nombre de composantes fortement connexes: %d\n", NumStrongConnComp);

    // 6. Nombre de cycles avec 3 sommets
    const int NumCycles3 = TSnap::GetTriads(Graph);
    //printf("6. Nombre de cycles avec 3 sommets: %d\n", NumCycles3);

    // 7. Dégénérescence
    TIntV DegSeq;
    int MaxDegeneracy = 0;  // Variable to store the maximum degeneracy

    while (!Graph->Empty()) {
        int MinOutDegree = TInt::Mx;
        int MinOutNId = -1;

        for (TNGraph::TNodeI NI = Graph->BegNI(); NI < Graph->EndNI(); NI++) {
```

```
            if (NI.GetOutDeg() < MinOutDegree) {
                MinOutDegree = NI.GetOutDeg();
                MinOutNId = NI.GetId();
            }
        }

        if (MinOutNId != -1) {
            DegSeq.Add(MinOutDegree);
            Graph->DelNode(MinOutNId);  // Remove node with minimum out-degree

            // Update the maximum degeneracy
            MaxDegeneracy = TMath::Mx(MaxDegeneracy, MinOutDegree);

            // Print intermediate steps
        }
    }

    // Assignez les résultats à la structure GraphResult
    Result.GraphName = GraphName;
    Result.NumNodes = NumNodes;
    Result.NumEdges = NumEdges;
    Result.MaxDegree = MaxOutDegree;
    Result.NumConnComp = NumStrongConnComp;
    Result.NumCycles3 = NumCycles3;
    Result.MaxDegeneracy = MaxDegeneracy;  // Ajoutez cette ligne pour stocker la dégénéresc

    // Print the maximum degeneracy
    //printf("7. Dégénérescence : %d\n", MaxDegeneracy);
}

int main(int argc, char* argv[]) {
    Env = TEnv(argc, argv, TNotify::StdNotify);
    Env.PrepArgs(TStr::Fmt("Analyse de Graphe Orienté. Build: %s, %s. Heure: %s", __TIME__,

    // Spécifiez ici les noms des fichiers de graphes que vous souhaitez analyser
    TStrV GraphFiles;
    GraphFiles.Add("CA-GrQc.txt");
    GraphFiles.Add("twitchDE.txt");
    // ... ajoutez autant de fichiers que nécessaire

    // Créez un vecteur pour stocker les résultats de chaque graphe
    TVec<GraphResult> Results;

    // Analysez chaque graphe et stockez les résultats dans le vecteur
    for (int i = 0; i < GraphFiles.Len(); i++) {
        const TStr& InFNm = GraphFiles[i];
```

```
        PNGraph Graph = TSnap::LoadEdgeList<PNGraph>(InFNm);
        GraphResult Result;

        // Analysez le graphe
        AnalyzeDirectedGraph(Graph, InFNm.GetFMid(), Result);

        // Stockez les résultats dans le vecteur
        Results.Add(Result);
    }

    // Imprimez les résultats sous forme de tableau
    printf("| Nom du Graphe | Nombre de Sommets | Nombre d'Arêtes | Degré Sortant Maximal |
    printf("|---------------|-------------------|-----------------|---------------|-----

    for (int i = 0; i < Results.Len(); i++) {
        const GraphResult& Result = Results[i];
        printf("| %-13s| %-20d| %-15d| %-14d| %-29d| %-31d| %-15d|\n",
               Result.GraphName.CStr(), Result.NumNodes, Result.NumEdges, Result.MaxDegree,
               Result.NumConnComp, Result.NumCycles3, Result.MaxDegeneracy);
    }

    return 0;
}
```

2.2 Description de l'Algorithme

1. **Type de graphe:** On commence par déterminer si le graphe est orienté.
2. **Nombre de sommets:** On calcule le nombre de sommets dans le graphe.
3. **Nombre d'arêtes:** On calcule le nombre d'arêtes dans le graphe.
4. **Degré sortant maximal:** Trouver le degré sortant maximal en parcourant tous les nœuds du graphe.
5. **Nombre de composantes fortement connexes:** Recherche des composantes fortement connexes.
6. **Nombre de cycles avec 3 sommets:** Utilisation de l'algorithme de recherche des triades pour trouver les cycles à 3 sommets.
7. **Dégénérescence:** Calcule la dégénérescence du graphe en supprimant itérativement les nœuds de degré minimum.

2.3 Analyse de Graphe Non Orienté

Nous avons développé un algorithme pour analyser les graphes non orientés. L'algorithme utilise la bibliothèque SNAP pour effectuer les calculs suivants :

- Nombre de sommets
- Nombre d'arêtes
- Degré maximal
- Nombre de composantes connexes
- Nombre de cycles avec 3 sommets
- Dégénérescence

L'algorithme est basé sur la suppression itérative des nœuds de degré minimum pour calculer la dégénérescence :

```
#include "Snap.h"

// Structure pour stocker les résultats d'analyse d'un graphe
struct GraphResult {
    TStr GraphName;
    int NumNodes;
    int NumEdges;
    int MaxDegree;
    int NumConnComp;
    int NumCycles3;
    int MaxDegeneracy;  // Ajout du champ pour stocker la dégénérescence
};

void AnalyzeGraph(const PUNGraph& Graph, const TStr& GraphName, GraphResult& Result) {
    TIntV NIdV;
    TCnComV CnComV;

    // 1. Type de graphe (orienté ou non orienté)
    //printf("1. Type de graphe: Non orienté\n");

    // 2. Nombre de sommets
    const int NumNodes = Graph->GetNodes();
    //printf("2. Nombre de sommets: %d\n", NumNodes);

    // 3. Nombre d'arêtes
    const int NumEdges = Graph->GetEdges();
    //printf("3. Nombre d'arêtes: %d\n", NumEdges);

    // 4. Degré maximal
```

```
int MaxDegree = 0;

for (TUNGraph::TNodeI NI = Graph->BegNI(); NI < Graph->EndNI(); NI++) {
    MaxDegree = TMath::Mx(MaxDegree, NI.GetDeg());
}

//printf("4. Degré maximal: %d\n", MaxDegree);

// 5. Nombre de composantes connexes
TSnap::GetWccs(Graph, CnComV);
const int NumConnComp = CnComV.Len();
//printf("5. Nombre de composantes connexes: %d\n", NumConnComp);

// 6. Nombre de cycles avec 3 sommets
const int NumCycles3 = TSnap::GetTriads(Graph);
//printf("6. Nombre de cycles avec 3 sommets: %d\n", NumCycles3);

// 7. Dégénérescence
int MaxDegeneracy = 0;
TIntV DegSeq;

while (!Graph->Empty()) {
    int MinDeg = TInt::Mx;
    int MinNId = -1;

    // Trouver le nœud de degré minimum
    for (TUNGraph::TNodeI NI = Graph->BegNI(); NI < Graph->EndNI(); NI++) {
        if (NI.GetDeg() < MinDeg) {
            MinDeg = NI.GetDeg();
            MinNId = NI.GetId();
        }
    }

    if (MinNId != -1) {
        // Sauvegarder le degré du nœud supprimé à chaque itération
        DegSeq.Add(MinDeg);

        // Mettre à jour la dégénérescence à chaque itération
        MaxDegeneracy = TMath::Mx(MaxDegeneracy, MinDeg);

        // Supprimer le nœud de degré minimum
        Graph->DelNode(MinNId);
    }
}

// Afficher la dégénérescence maximale à la fin
```

```
    //printf("7. Dégénérescence: %d\n", MaxDegeneracy);

    // Assignez les résultats à la structure GraphResult
    Result.GraphName = GraphName;
    Result.NumNodes = NumNodes;
    Result.NumEdges = NumEdges;
    Result.MaxDegree = MaxDegree;
    Result.NumConnComp = NumConnComp;
    Result.NumCycles3 = NumCycles3;
    Result.MaxDegeneracy = MaxDegeneracy;  // Ajoutez cette ligne pour stocker la dégénéresc

    // Ajout du résultat supplémentaire (vous pouvez définir cette valeur en fonction de vot
}

int main(int argc, char* argv[]) {
    Env = TEnv(argc, argv, TNotify::StdNotify);
    Env.PrepArgs(TStr::Fmt("Analyse de Graphe Non Orienté. Build: %s, %s. Heure: %s", __TIME

    // Spécifiez ici les noms des fichiers de graphes que vous souhaitez analyser
    TStrV GraphFiles;
    GraphFiles.Add("FacebookSites.txt");
    GraphFiles.Add("GitHub.txt");
    GraphFiles.Add("Wikipedia1.txt");
    // ... ajoutez autant de fichiers que nécessaire

    // Créez un vecteur pour stocker les résultats de chaque graphe
    TVec<GraphResult> Results;

    // Analysez chaque graphe et stockez les résultats dans le vecteur
    for (int i = 0; i < GraphFiles.Len(); i++) {
        const TStr& InFNm = GraphFiles[i];
        PUNGraph Graph = TSnap::LoadEdgeList<PUNGraph>(InFNm);
        GraphResult Result;

        // Analysez le graphe
        AnalyzeGraph(Graph, InFNm.GetFMid(), Result);

        // Stockez les résultats dans le vecteur
        Results.Add(Result);
    }

    // Imprimez les résultats sous forme de tableau
    printf("| Nom du Graphe | Nombre de Sommets | Nombre d'Arêtes | Degré Maximal | Nombre d
    printf("|---------------|-------------------|-----------------|---------------|------

    for (int i = 0; i < Results.Len(); i++) {
```

```
        const GraphResult& Result = Results[i];
        printf("| %-13s| %-20d| %-15d| %-13d| %-23d| %-31d| %-15d|\n",
               Result.GraphName.CStr(), Result.NumNodes, Result.NumEdges, Result.MaxDegree,
               Result.NumConnComp, Result.NumCycles3, Result.MaxDegeneracy);
    }

    return 0;
}
```

2.4 Description de l'Algorithme

1. **Type de graphe:** On commence par déterminer si le graphe est orienté ou non orienté.
2. **Nombre de sommets:** On calcule le nombre de sommets dans le graphe.
3. **Nombre d'arêtes:** On calcule le nombre d'arêtes dans le graphe.
4. **Degré sortant maximal:** Trouver le degré maximal en parcourant tous les nœuds du graphe.
5. **Nombre de composantes connexes:** Recherche des composantes connexes.
6. **Nombre de cycles avec 3 sommets:** Utilisation de l'algorithme de recherche des triades pour trouver les cycles à 3 sommets.
7. **Dégénérescence:** Calcule la dégénérescence du graphe en supprimant itérativement les nœuds de degré minimum.

3 Distribution de Graphes

3.1 Distribution de Graphe Orienté

Le code inclut une fonction pour calculer la distribution des degrés de sortie des graphes orientés. Cette distribution est ensuite visualisée à l'aide de Gnuplot, générant des fichiers d'histogrammes :

```
#include "Snap.h"
#include "gnuplot.h"

// Fonction pour calculer et écrire la distribution des degrés dans un fichier
void WriteDegreeDistribution(const PNGraph& Graph, const TStr& GraphName) {
    TIntH DegreeFreq;

    // Calculer la fréquence des degrés de sortie
```

```
    for (TNGraph::TNodeI NI = Graph->BegNI(); NI < Graph->EndNI(); NI++) {
        int OutDeg = NI.GetOutDeg();
        DegreeFreq.AddDat(OutDeg)++;
    }

    // Écrire les données dans un fichier
    FILE* OutFile = fopen((GraphName + "_degree_distribution.dat").CStr(), "w");
    fprintf(OutFile, "# Degré Fréquence\n");

    // Utiliser les itérateurs pour parcourir la collection
    for (TIntH::TIter it = DegreeFreq.BegI(); it < DegreeFreq.EndI(); it++) {
        fprintf(OutFile, "%d %d\n", it.GetKey().Val, it.GetDat().Val);
    }
    fclose(OutFile);
}

// Fonction pour tracer la distribution des degrés à l'aide de Gnuplot
void PlotDegreeDistributionGnuplot(const TStr& GraphName) {
    FILE* GnuplotPipe = popen("gnuplot", "w");
    fprintf(GnuplotPipe, "set terminal png\n");
    fprintf(GnuplotPipe, "set output '%s_degree_distribution.png'\n", GraphName.CStr());
    fprintf(GnuplotPipe, "set title 'Distribution des Degrés - %s'\n", GraphName.CStr());
    fprintf(GnuplotPipe, "set xlabel 'Degré'\n");
    fprintf(GnuplotPipe, "set ylabel 'Fréquence'\n");
    fprintf(GnuplotPipe, "plot '%s_degree_distribution.dat' with boxes\n", GraphName.CStr()
    fclose(GnuplotPipe);
}

int main(int argc, char* argv[]) {
    // Initialiser l'environnement Snap
    Env = TEnv(argc, argv, TNotify::StdNotify);
    Env.PrepArgs(TStr::Fmt("Analyse de Graphe Orienté. Build: %s, %s. Heure: %s", __TIME__,

    // Spécifier ici les noms des fichiers de graphes que vous souhaitez analyser
    TStrV GraphFiles;
    GraphFiles.Add("CA-GrQc.txt");
    GraphFiles.Add("twitchDE.txt");

    // ... ajouter autant de fichiers que nécessaire

    // Analyser chaque graphe et écrire la distribution des degrés
    for (int i = 0; i < GraphFiles.Len(); i++) {
        const TStr& InFNm = GraphFiles[i];
        PNGraph Graph = TSnap::LoadEdgeList<PNGraph>(InFNm);

        // Écrire la distribution des degrés dans un fichier
```

```
        WriteDegreeDistribution(Graph, InFNm.GetFMid());

        // Utiliser Gnuplot pour tracer l'histogramme
        PlotDegreeDistributionGnuplot(InFNm.GetFMid());
    }

    return 0;
}
```

3.2 Description de l'Algorithme

1. **Calcul de la fréquence des degrés sortants :** Calcule la fréquence des degrés de sortie pour chaque nœud du graphe orienté.

2. **Écriture des résultats :** Les résultats sont écrits dans un fichier avec l'extension "$_d egree_d istribution.dat$"$.Chaque ligne du fichier contient une paire (degré, fréquence).$**Génération du diagra**

3.3 Distribution de Graphe Non Orienté

Un autre algorithme a été développé pour calculer la distribution des degrés de sortie des graphes non orientés. La bibliothèque SNAP est utilisée pour calculer la fréquence des degrés et générer un fichier de distribution :

3. ```
   #include "Snap.h"
   #include "gnuplot.h"

   // Fonction pour calculer et écrire la distribution des degrés dans un fichier
   void WriteDegreeDistribution(const PUNGraph& Graph, const TStr& GraphName) {
       TIntH DegreeFreq;

       // Calculer la fréquence des degrés
       for (TUNGraph::TNodeI NI = Graph->BegNI(); NI < Graph->EndNI(); NI++) {
           int Deg = NI.GetOutDeg();
           DegreeFreq.AddDat(Deg)++;
       }

       // Écrire les données dans un fichier
       FILE* OutFile = fopen((GraphName + "_degree_distribution.dat").CStr(), "w");
       fprintf(OutFile, "# Degré Fréquence\n");

       // Utiliser les itérateurs pour parcourir la collection
       for (TIntH::TIter it = DegreeFreq.BegI(); it < DegreeFreq.EndI(); it++) {
           fprintf(OutFile, "%d %d\n", it.GetKey().Val, it.GetDat().Val);
       }
       fclose(OutFile);
   }
   ```
   ```

```
// Fonction pour tracer la distribution des degrés à l'aide de Gnuplot
void PlotDegreeDistributionGnuplot(const TStr& GraphName) {
    FILE* GnuplotPipe = popen("gnuplot", "w");
    fprintf(GnuplotPipe, "set terminal png\n");
    fprintf(GnuplotPipe, "set output '%s_degree_distribution.png'\n", GraphName.CStr());
    fprintf(GnuplotPipe, "set title 'Distribution des Degrés - %s'\n", GraphName.CStr());
    fprintf(GnuplotPipe, "set xlabel 'Degré'\n");
    fprintf(GnuplotPipe, "set ylabel 'Fréquence'\n");
    fprintf(GnuplotPipe, "plot '%s_degree_distribution.dat' with boxes\n", GraphName.CStr()
    fclose(GnuplotPipe);
}

int main(int argc, char* argv[]) {
    // Initialiser l'environnement Snap
    Env = TEnv(argc, argv, TNotify::StdNotify);
    Env.PrepArgs(TStr::Fmt("Analyse de Graphe Non Orienté. Build: %s, %s. Heure: %s", __TIME

    // Spécifier ici les noms des fichiers de graphes que vous souhaitez analyser
    TStrV GraphFiles;
    GraphFiles.Add("FacebookSites.txt");
    GraphFiles.Add("Wikipedia1.txt");
    GraphFiles.Add("GitHub.txt");

    // ... ajouter autant de fichiers que nécessaire

    // Analyser chaque graphe et écrire la distribution des degrés
    for (int i = 0; i < GraphFiles.Len(); i++) {
        const TStr& InFNm = GraphFiles[i];
        PUNGraph Graph = TSnap::LoadEdgeList<PUNGraph>(InFNm);

        // Écrire la distribution des degrés dans un fichier
        WriteDegreeDistribution(Graph, InFNm.GetFMid());

        // Utiliser Gnuplot pour tracer l'histogramme
        PlotDegreeDistributionGnuplot(InFNm.GetFMid());
    }

    return 0;
}
```

4 Résultats

Les résultats de l'analyse des graphes sont présentés dans le tableau ci-dessous.
La distribution des degrés est également tracée sous forme d'histogramme.

Graphe	Sommets	Arêtes	Degré Sortant Maximal	Comp. Fortement Connexes	Cycles
CA-GrQc	5242	28980	81	355	
TwitchDE	9417	153055	1256	2723	

Table 1: Résultats de l'Analyse des Graphes Orientés

Graphe	Sommets	Arêtes	Degré Maximal	Comp. Connexes	Cycles 3	Dégénéresc
FacebookSites	22012	170947	708	11	720374	52
GitHub	37002	288745	7472	5	476817	34
Wikipedia1	2277	32361	732	1	356442	63

Table 2: Résultats de l'Analyse des Graphes Non Orientés

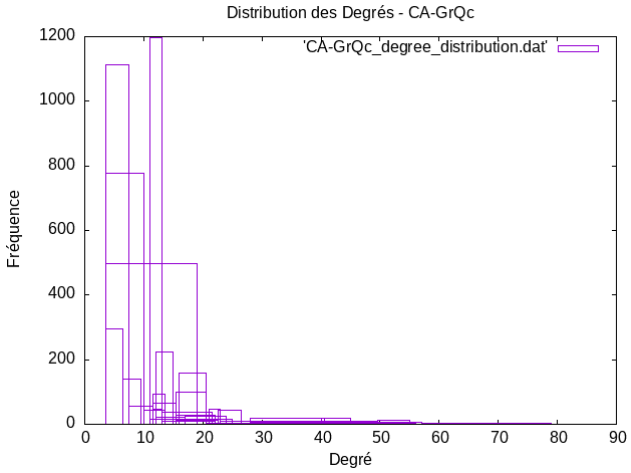

Figure 1: Distribution des Degrés - CA-GrQc

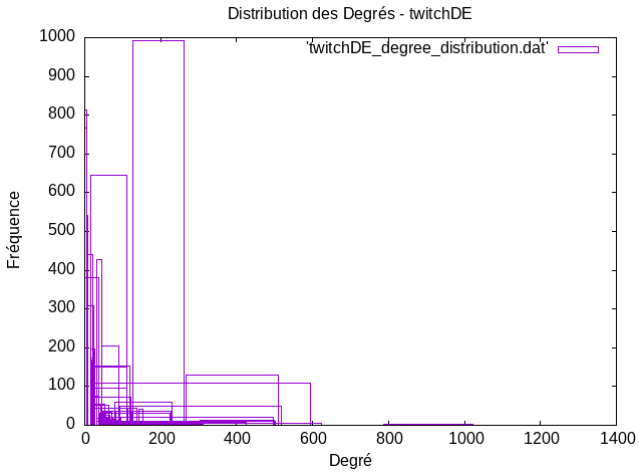

Figure 2: Distribution des Degrés - CA-GrQc

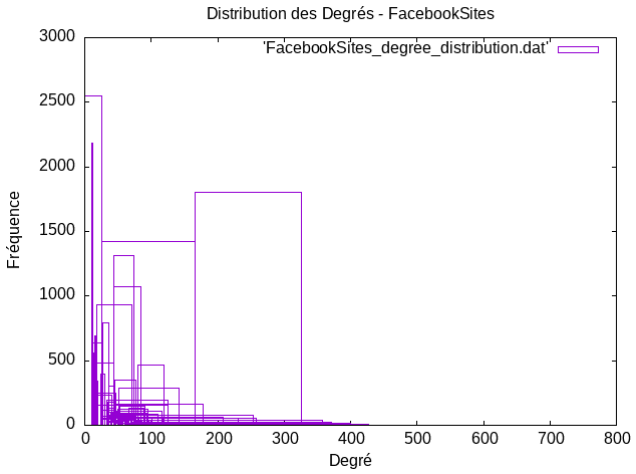

Figure 3: Distribution des Degrés - FacebookSites

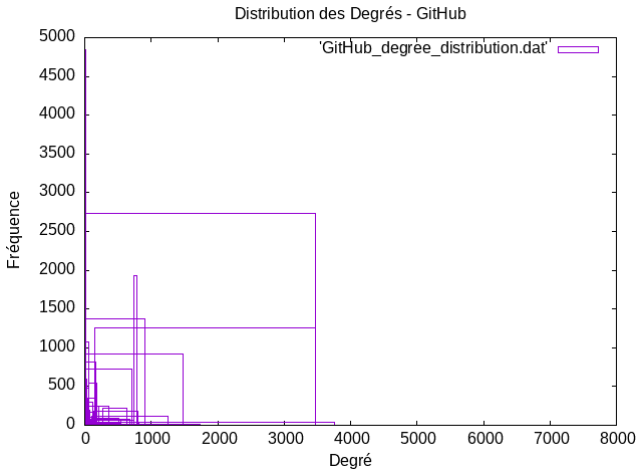

Figure 4: Distribution des Degrés - GitHub

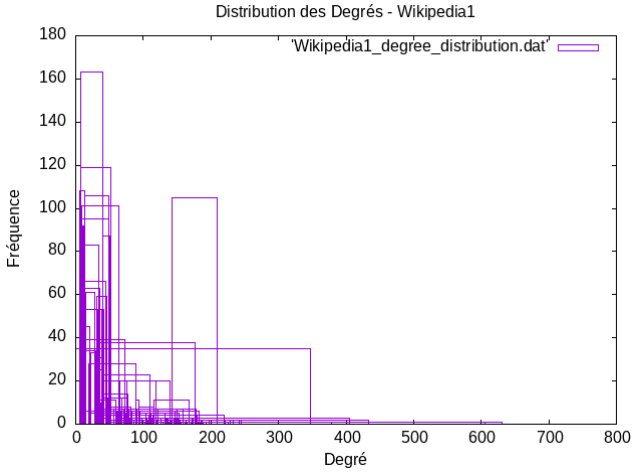

Figure 5: Distribution des Degrés - Wikipedia1

5 Complexité Algorithmique

5.1 Analyse de Graphe Orienté

L'analyse de graphes orientés repose sur plusieurs opérations fondamentales, chacune ayant sa propre complexité algorithmique.

1. **Nombre de Sommets et d'Arêtes :** La détermination du nombre de sommets et d'arêtes a une complexité linéaire par rapport à la taille du

graphe, car il suffit de parcourir l'ensemble des nœuds et des arêtes une fois.

2. **Degré Sortant Maximal :** Calculer le degré sortant maximal nécessite de parcourir tous les nœuds du graphe une fois, ce qui donne une complexité linéaire.

3. **Composantes Fortement Connexes :** La recherche des composantes fortement connexes utilise un algorithme basé sur la recherche en profondeur (Depth-First Search, DFS) et a une complexité linéaire.

4. **Cycles avec 3 Sommets :** L'algorithme de recherche de triades a une complexité qui dépend de la structure spécifique du graphe, mais en général, elle est linéaire.

5. **Dégénérescence :** Le calcul de la dégénérescence en supprimant itérativement les nœuds de degré minimum a une complexité quadratique dans le pire des cas, car cela peut nécessiter de parcourir tous les nœuds pour chaque nœud supprimé.

5.2 Analyse de Graphe Non Orienté

L'analyse de graphes non orientés partage certaines similitudes avec celle des graphes orientés, mais avec des variations dues à la nature non orientée des arêtes.

1. **Nombre de Sommets et d'Arêtes :** La complexité reste linéaire, car le processus de parcours est similaire à celui des graphes orientés.

2. **Degré Maximal :** Calculer le degré maximal implique également de parcourir tous les nœuds, ce qui donne une complexité linéaire.

3. **Composantes Connexes :** La recherche des composantes connexes utilise un algorithme basé sur la recherche en largeur (Breadth-First Search, BFS) et a une complexité linéaire.

4. **Cycles avec 3 Sommets :** L'algorithme de recherche de triades a une complexité qui dépend de la structure du graphe, mais elle est généralement linéaire.

5. **Dégénérescence :** Le calcul de la dégénérescence a une complexité quadratique dans le pire des cas, similaire à celui des graphes orientés.

5.3 Distribution des Degrés

Les algorithmes de distribution des degrés, tant pour les graphes orientés que non orientés, ont une complexité linéaire par rapport à la taille du graphe, car ils nécessitent de parcourir tous les nœuds et de compter la fréquence de chaque degré.

5.4 Optimisations Potentielles

Des optimisations spécifiques peuvent être envisagées en fonction des propriétés du graphe, telles que sa densité, sa connectivité, ou la distribution des degrés. L'utilisation de structures de données efficaces, comme celles fournies par la bibliothèque SNAP, contribue également à améliorer les performances.

6 Conclusion

Ce rapport offre une compréhension approfondie des propriétés des graphes, tant orientés que non orientés, grâce à une analyse détaillée et à la visualisation de leurs caractéristiques clés. Les algorithmes développés, combinés à la bibliothèque SNAP, fournissent des outils puissants pour explorer la structure et la connectivité des graphes.

Remerciements

Je tiens à exprimer mes sincères remerciements à Mme Cohenne Jeanne et Mr George Manoussakis pour les conseils et réponse aux questions tout au long de ce projet.

7 Références Bibliographiques

- SNAP - Stanford Network Analysis Project.
- Librairies SNAP : `snap-core` et `glib-core`.
- Documentation officielle SNAP. https://snap.stanford.edu/snappy/doc/reference/graphs.html.
- CS224W: Machine Learning with Graphs - Stanford University https://web.stanford.edu/class/cs224w/.
- "Modern Graph Theory" - Bollobas.
- "Network Flows" - Ahuja, Magnanti, and Orlin.
- "Analyzing Social Media Networks with NodeXL: Insights from a Connected World" - Hansen and Al .
- MIT OpenCourseWare - Graph Theory ++ math for computer science, lectures 6-10.
- Algorithms for specific algorithms of interest.