



UNIVERSITE DE VERSAILLES SAINT
QUENTAIN EN YVELINES

RAPPORT DE PROJET

Outils pour la conception d'algorithmes
Mention : Informatique
Spécialité : IATIC

Par :

Sinda Chams El Ghazlen Gharsi
Manar Omezzine
Ameur Ben sghaier

Encadrant : GEORGE MANOUSSAKIS

Encadrante : JOHEN COHENE

DETECTION DES COMMUNAUTES DANS LES RESEAUX SOCIAUX

REMERCIEMENTS

Nous tenons à exprimer notre profonde gratitude à Monsieur George Manoussakis, notre professeur pour ses directives, ses conseils, et le temps qu'il nous a consacré pour répondre à nos différents lacunes.

Nos remerciements les plus cordiaux s'adressent aussi à Madame Johen Cohen pour sa disponibilité, son aide et ses critiques constructives.

Nous la remercions également pour ses qualités humaines et morales que nous avons toujours appréciées et respectées

TABLE DES MATIÈRES

Introduction générale	1
1 Présentation générale	2
Introduction	3
1.1 Présentation générale	3
1.1.1 Présentation de l'ISTY	3
1.2 Analyse de l'existant	3
1.2.1 Cadre du sujet	3
1.2.2 La finalité du projet	3
1.3 Travail demandé	4
1.3.1 But du sujet	4
1.3.1.1 Partie 1	4
1.3.1.2 Partie 1.1	4
1.3.1.3 Partie 2	4
1.3.1.4 Partie 3	5
Conclusion	5
2 Planification et capture des besoins	6
Introduction	7
2.1 Capture des besoins	7
2.1.1 Spécification des besoins	7
2.1.1.1 Besoins fonctionnels	7

2.1.1.2	Besoins non fonctionnels	7
2.1.2	Modélisation des besoins	8
2.1.3	Environnement matériel et logiciel	8
2.1.4	Planification des parties	9
	Conclusion.....	10
3	Étude et réalisation du projet	11
3.1	Technologies et outils utilisés :.....	12
	Technologies et outils utilisés :.....	12
3.2	Choix de structures.....	13
3.3	Première Partie	13
3.3.1	Génération des graphes aléatoires	13
3.3.1.1	Implémentation.....	14
3.3.1.2	Test et vérification.....	15
3.3.2	Génération des grands graphes de Stanford.....	17
3.3.2.1	Implémentation.....	18
3.3.2.2	Test et vérification.....	18
3.4	Deuxième Partie	21
3.4.1	Algorithme de Bron-Kerbosch.....	21
3.4.1.1	Version avec Pivot	21
3.4.1.2	Description de l'implémentation :	22
3.4.1.3	Test et vérification :.....	23
3.5	Troisième Partie	23
3.5.1	Étude de l'algorithme	23
3.6	Comparaison de graphes par rapport au temps.....	26
	Conclusion générale	27
	Références bibliographiques	28

TABLE DES FIGURES

2.1	Diagramme de cas d'utilisation globale	8
2.2	Planification des parties	9
2.3	table de tâches sur Trello	10
3.1	Spyder	12
3.2	Python	12
3.3	Networkx.....	13
3.4	Matplotlib.....	13
3.5	Github	13
3.6	Génération d'un graphe aléatoire avec 9 sommets.....	15
3.7	Degré maximum dans un graphe	16
3.8	Graphe aléatoire	16
3.9	Histogramme du graphe	16
3.10	Graphe aléatoire	17
3.11	Matrice d'adjacence au carré	17
3.12	Génération du graphe	18
3.13	Histogramme du graphe	19
3.14	Génération du graphe	19
3.15	Histogramme du graphe	20
3.16	Génération du graphe	20
3.17	Histogramme du graphe	21

TABLE DES FIGURES

3.18 Génération du graphe	23
3.19 Génération des cliques	23
3.20 Graphe généré.....	24
3.21 Liste dégénérescence du graphe	24
3.22 Graphe généré.....	25
3.23 les sous graphes gi du graphe.....	25
3.24 Comparaison de graphes par rapport au temps	26

LISTE DES TABLEAUX

2.1	Description de l'environnement logiciel et matériel de travail	9
-----	--	---

INTRODUCTION GÉNÉRALE

Dans le but de compléter les connaissances acquises et mettre en application les algorithmes vus durant les cours, il nous a été demandé d'implémenter, en utilisant le langage de programmation Python, certains algorithmes de graphes issus de la littérature scientifique récente.

L'analyse des réseaux sociaux est avant tout une boîte à outil permettant de visualiser et modéliser les relations sociales comme des nœuds (les individus, les organisations...) et des liens (relations entre ces nœuds). De ce fait, l'analyse des réseaux sociaux repose sur des visualisations graphiques issues d'algorithmes permettant de calculer des degrés de force ou de densité entre les différents acteurs d'un réseau.

Ce rapport expliquera les différentes étapes de notre projet; dans la première partie nous allons expliquer donner pour les graphes aléatoires générés et les graphes de Stanford, Le degré maximum du graphe, Un graphique donnant pour chaque degré du graphe le nombre de sommets ayant ce degré et Le nombre de chemins induits dans le graphe de longueur 2. Par la suite, nous allons présenter L'algorithme de Bron Kerbosch (version avec pivot) qui énumère les cliques maximales d'un graphe et de mettre un exemple d'exécution sur un graphe aléatoire de la première partie et un autre sur un graphe aussi grand que possible en termes de sommets. Finalement, nous allons implémenter divers algorithmes liés à l'énumération des cliques et de bicliques maximales, qui se trouvent dans le papier.

Ce travail a été réalisé en trinôme, dans le but de mieux gérer notre temps ainsi que les différentes versions de notre code, nous avons utilisé GitHub comme outil de travail collaboratif.

CHAPITRE 1

PRÉSENTATION GÉNÉRALE

Introduction	3
1.1 Présentation générale.....	3
1.1.1 Présentation de l'ISTY	3
1.2 Analyse de l'existant.....	3
1.2.1 Cadre du sujet	3
1.2.2 La finalité du projet	3
1.3 Travail demandé	4
1.3.1 But du sujet	4
Conclusion	5

Introduction

Dans le cadre de la deuxième année du cycle ingénieur à l'ISTY, il nous est proposé un projet de 6 semaines. Ce projet nous permettra de mettre en pratique nos connaissances, nos compétences professionnelles et surtout de développer l'esprit d'équipe à travers un cahier de charges ayant pour finalité le développement d'un outil qui vise à détecter les communautés dans les réseaux sociaux, c'est-à-dire des ensembles de nœuds très densément connectés entre eux.

1.1 Présentation générale

1.1.1 Présentation de l'ISTY

L'institut des sciences et techniques des Yvelines (ISTY) est l'une des 204 écoles d'ingénieurs françaises accréditées au 1er septembre 2020 à délivrer un diplôme d'ingénieur.

C'est l'école d'ingénieurs publique de l'Université de Versailles-Saint-Quentin-en-Yvelines (UVSQ). [1]

1.2 Analyse de l'existant

1.2.1 Cadre du sujet

Dans cette section nous allons dresser la finalité du projet suivie du travail demandé et enfin, nous allons clore par une conclusion.

1.2.2 La finalité du projet

Dans le cadre de la formation de L'IATIC, l'ISTY accompagne ses étudiants au niveau des contenus pédagogiques afin que les acquis puissent recouvrir les connaissances nécessaires aux futurs métiers ciblés par l'école en tenant compte des évolutions technologiques propres aux domaines des technologies de l'information et de la communication.

1.3 Travail demandé

1.3.1 But du sujet

Le but du projet est d'implémenter des algorithmes d'énumération de bibliques maximales dans des graphes générés aléatoirement. Le projet s'articule en trois parties.

1.3.1.1 Partie 1

✱ Générer des graphes aléatoires

La première partie consiste à générer des graphes aléatoires. Étant donné un nombre fixe de sommets, chaque arête apparaît avec probabilité p choisie aléatoirement telle que $0 < p < 1$.

✱ Stockage des graphes

stocker les graphes à l'aide de listes d'adjacences. L'objectif est de créer des graphes les plus grands possibles en termes de sommets.

✱ Utilisation de la base de données de Stanford

stocker les graphes à l'aide de listes d'adjacences. L'objectif est de créer des graphes les plus grands possibles en termes de sommets.

✱ Stockage des graphes

utiliser la base de données des grands graphes de Stanford et stocker en mémoire les graphes des réseaux sociaux suivants : egoFacebook, feather-lastfm-social et email-Eu-core.

1.3.1.2 Partie 1.1

✱ Degré maximum du graphe

Calculer le degré maximum pour les graphes aléatoires et Stanford.

✱ Générer un graphique

Générer un graphique donnant pour chaque degré du graphe le nombre de sommets ayant ce degré.

1.3.1.3 Partie 2

✱ Algorithme Bron Kerbosch

Écrire un algorithme de complexité exponentielle pour l'énumération des cliques d'un graphe. Soit un graphe $G=(V,E)$ avec V l'ensemble des sommets et E l'ensemble des arêtes

de G . Une clique de G est un ensemble K de sommets du graphe toutes connectées deux à deux. La clique K est maximale si elle n'est incluse dans aucune autre clique de G , c'est-à-dire qu'il n'existe pas de sommet dans $V-K$ connecté à tous les sommets de K . L'algorithme de Bron Kerbosch est un des algorithmes les plus connus énumérant les cliques maximales d'un graphe. Il en existe de nombreuses variantes. On est demandé d'implémenter la version avec pivot de l'algorithme.

1.3.1.4 Partie 3

✱ Énumération des cliques et bicliques maximales

Une clique est un graphe complet. Elle est maximale s'il n'est pas possible d'ajouter d'autres sommets du graphe et d'obtenir une clique de taille plus grande. Une biclique est un graphe biparti complet. C'est à dire que toutes les arêtes sont présentes entre les deux parties du graphe. Elle est maximale s'il n'est pas possible de rajouter des sommets à la biclique et d'obtenir une biclique de taille (en termes de sommets) plus grande. On est demandé d'implémenter les algorithmes donné dans le papier du projet.

Conclusion

Ce chapitre a été consacré à la présentation générale de l'ISTY . Ensuite nous avons exposé les différentes parties du projet, ainsi que le travail demandé.

Ce chapitre nous a permis également d'exposer l'approche de développement que nous allons adopter tout au long de notre projet.

Ce chapitre nous permettra donc de passer au deuxième chapitre qui est la planification et capture des besoins.

CHAPITRE 2

PLANIFICATION ET CAPTURE DES BESOINS

Introduction	7
2.1 Capture des besoins.....	7
2.1.1 Spécification des besoins	7
2.1.2 Modélisation des besoins	8
2.1.3 Environnement matériel et logiciel.....	8
2.1.4 Planification des parties.....	9
Conclusion	10

Introduction

Dans ce chapitre , nous allons se concentrer sur la première partie du projet qui est la spécification des besoins et la planification . Par conséquent, nous allons commencer par identifier les acteurs du projet , élaborer la démarche rationnelle qui part de l'identification et de l'analyse du problème, de la fixation des buts et des objectifs atteignables.

2.1 Capture des besoins

L'analyse des besoins est une étape principale qui a pour but de satisfaire les attentes des utilisateurs. Dans cette partie, nous commençons en premier lieu par identifier les acteurs. Nous définissons ensuite les besoins fonctionnels de notre projet.

2.1.1 Spécification des besoins

Dans cette section nous allons identifier les besoins fonctionnels et non fonctionnels relatifs à notre projet.

2.1.1.1 Besoins fonctionnels

Suite aux réunions en binôme, nous identifions l'ensemble des besoins évoqués :

- ✱ **Générer des automates fini non-déterministe.**
- ✱ **Générer des graphes aléatoires.**
- ✱ **Stockage des graphes.**
- ✱ **Utilisation de la base de données de Stanford**
- ✱ **Calculer le degré maximum du graphe.**
- ✱ **Générer un graphique**
- ✱ **Énumération des cliques avec Bron Kerbosch.**
- ✱ **Énumération des cliques et bicliques maximales.**

2.1.1.2 Besoins non fonctionnels

Dans ce contexte, les besoins non fonctionnels doivent répondre aux critères suivants :

- ✱ **Intégration** : Le projet doit être facile à implémenter et déployer sur n'importe quel autre machine.

- ✱ **La performance** : Il est nécessaire que le temps de réponse soit en temps réel. En outre, le temps de déploiement doit être réduit d'une manière remarquable.
- ✱ **Ergonomie** : Les principales fonctionnalités doivent prendre en considération tout type d'utilisateur.

2.1.2 Modélisation des besoins

A fin d'éclaircir le comportement de notre projet, nous proposons un diagramme de cas d'utilisation global dans le but de donner une vision plus claire. Ce diagramme comporte les fonctionnalités de bases que doit le programme .

La figure 3.1 présente le diagramme de cas d'utilisation global de notre projet.

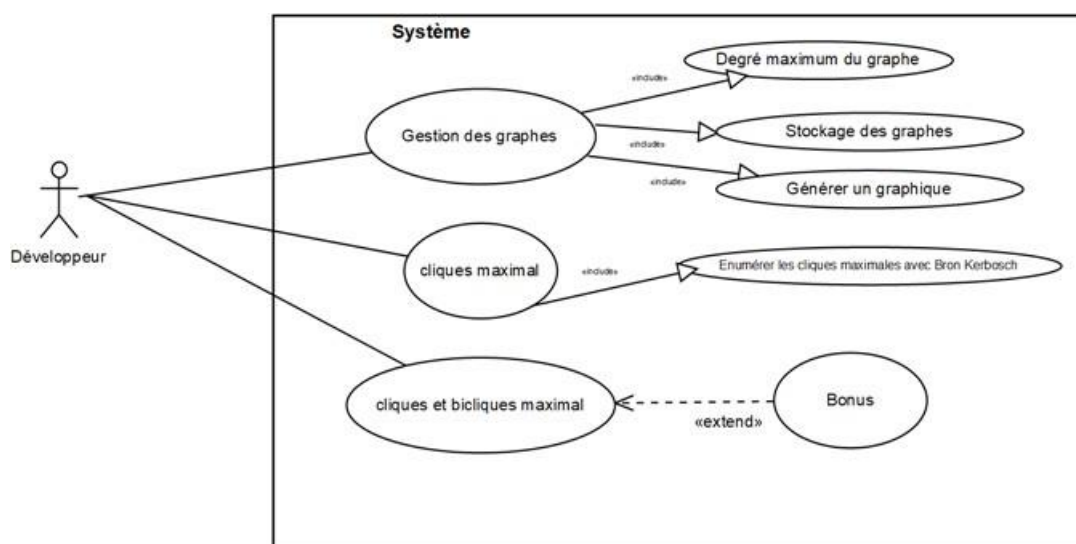


FIGURE 2.1 – Diagramme de cas d'utilisation globale

2.1.3 Environnement matériel et logiciel

Pour la réalisation de ce projet, l'environnement matériel a son importance dans le but de donner une vue globale sur les différentes tâches proposées. Comme tout projet informatique, il s'agit d'un ensemble d'outils et des plateformes utilisés pour répondre aux exigences fixés lors de la phase d'analyse des besoins . Dans le cadre de ce projet ,nous avons adopté des diverses technologie comme indique le tableau suivant :

TABLE 2.1 – Description de l’environnement logiciel et matériel de travail

machine de déploiement	→ PC portable azus, GR3 15 , Processeur : 10th Gen Intel® Core™ i7-8750HQ Quad Core Processor, RAM : 20 GB , Disque dur : 1 To HDD, 128 Gb SSD
Outil de distribution	→ Spyder
service d’hébergement du code	→ Github
Langage de scripts	→ Python

2.1.4 Planification des parties

la réunion de planification est une étape essentielle dans le projet. Lors de cette réunion, l’équipe choisira la durée nécessaire pour chaque partie en Dépendant de la complexité de ce dernier, et aussi en tenant également compte de l’objectif du projet. Dans notre Dans ce cas, nous avons décidé de diviser les taches par thèmes, comme le montre la figure 2.2



FIGURE 2.2 – Planification des parties

Afin de pallier des problèmes de planifications internes et de management du projet, nous avons opté pour l'outil collaboratif Trello. Cet outil nous permet de mieux gérer et planifier notre travail grâce à la création des tables de tâches. Dans notre cas, nous avons utilisé l'outil Trello au cours des réunions quotidiennes comme le montre la figure



FIGURE 2.3 – table de tâches sur Trello

Pour organiser notre travail, nous avons les listes de tâches suivantes :

- ✧ Backlog : contient l'ensemble des tâches pour tous les membres de l'équipe.
- ✧ To do : contient la liste des tâches à réaliser, elle doit être pleine au début du projet et vide à la fin.
- ✧ In progress : contient les tâches qui sont entrain de se réaliser, elle doit être vide à la fin du projet.
- ✧ Done : contient les tâches qui ont été réalisées, elle doit être vide au début du projet et pleine à la fin.
- ✧ Blocked : contient les tâches dont les membres de l'équipe trouvent des lacunes lors de l'implémentation.

Conclusion

Dans ce chapitre, nous avons présenté la partie théorique du projet. En effet, nous avons présente dans ce chapitre les exigences fonctionnelles et la planification du projet.

CHAPITRE 3

ÉTUDE ET RÉALISATION DU PROJET

3.1 Technologies et outils utilisés :	12
Technologies et outils utilisés :	12
3.2 Choix de structures	13
3.3 Première Partie	13
3.3.1 Génération des graphes aléatoires	13
3.3.2 Génération des grands graphes de Stanford	17
3.4 Deuxième Partie	21
3.4.1 Algorithme de Bron-Kerbosch	21
3.5 Troisième Partie	23
3.5.1 Étude de l'algorithme	23
3.6 Comparaison de graphes par rapport au temps	26

3.1 Technologies et outils utilisés :

Le développement de ce projet sera réalisé à l'aide des outils suivants :

✱ **Spyder**

est un environnement de développement pour Python. Libre (Licence MIT) et multiplateforme (Windows, Mac OS, GNU/Linux), il intègre de nombreuses bibliothèques d'usage scientifique : Matplotlib, NumPy, SciPy et IPython.

Créé et développé par Pierre Raybaut en 2008, Spyder est maintenu, depuis 2012, par une communauté de développeurs qui ont pour point commun d'appartenir à la communauté Python scientifique.



FIGURE 3.1 – Spyder

✱ **Python**

Python est un langage de programmation informatique généraliste. Contrairement à HTML, CSS ou JavaScript, son usage n'est donc pas limité au développement web. Il peut être utilisé pour tout type de programmation et de développement logiciel.

On s'en sert notamment pour le développement back end d'applications web ou mobile, et pour le développement de logiciels et d'applications pour PC. Il permet également d'écrire des scripts système, afin de créer des instructions pour un système informatique.



FIGURE 3.2 – Python

✱ **Networkx**

NetworkX est une bibliothèque Python qui permet d'instancier des graphes composés de nœuds et de ponts, ou liens. Grâce à cette bibliothèque, la manipulation de ces graphes est simplifiée.



FIGURE 3.3 – Networkx

✧ **Matplotlib**

Matplotlib est une bibliothèque destinée à tracer et visualiser des données sous formes de graphiques.



FIGURE 3.4 – Matplotlib

✧ **Github**

GitHub est un service d'hébergement Open-Source, permettant aux programmeurs et aux développeurs de partager le code informatique de leurs projets afin de travailler dessus de façon collaborative. On peut le considérer comme un Cloud dédié au code informatique.



FIGURE 3.5 – Github

3.2 Choix de structures

Pour pouvoir modéliser le graphe demandé; nous avons décidé de représenter un graphe en tant qu'objet auquel est associés de multiples fonctions. Un graphe est algorithmiquement une liste d'adjacence qui est représentée dans le langage Python, sous forme de dictionnaire qui est un type de collection associant une clé à une valeur.

Les clés sont dans notre cas les sommets et les valeurs associées sont les voisins de ces sommets. Toutes les fonctions liées aux graphes ont été développées dans la classe correspondante.

3.3 Première Partie

3.3.1 Génération des graphes aléatoires

Dans cette partie, nous allons implémenter des graphes aléatoires.

Étant donné un nombre fixe de sommets, chaque arête apparaît avec probabilité p choisie aléatoirement telle que $0 < p < 1$.

3.3.1.1 Implémentation

✱ Structure utilisée

Afin de présenter l'objet graphe, nous avons choisi de le modéliser par une classe graphe :

- ✱ Sommet : de type `int` qui présente le nombre des sommets du graphe généré.
- ✱ Dictionnaire : de type `dict` qui présente la liste d'adjacence du graphe et à qui sont associés des clés et des valeurs.

Dans notre cas; les clés sont les sommets du graphe et les valeurs sont les voisins de ces sommets.

✱ Fonction Générer graphe

Cette fonction prend en paramètre le graphe de type `Graphe` et retourne un dictionnaire qui a été rempli aléatoirement à partir d'une probabilité p entre 0 et 1.

`gauss()` est une méthode intégrée dans le module `Random`, elle a pour rôle de renvoyer un nombre aléatoire à virgule flottante avec une distribution gaussienne.

✱ Fonction dessiner graphe

Cette fonction prend en paramètre l'objet `Graphe`, en se référant à la bibliothèque `NetworkX` et en utilisant les fonctions prédéfinies de cette bibliothèque, on arrive à représenter la liste d'adjacence où les sommets qui sont les clés et les voisins qui sont les valeurs.

✱ Fonction degré maximum dans un graphe

La fonction `degre-max` qui retourne le degré maximal du graphe procède comme suit :
Itérer sur tous les sommets du graphe, Pour chaque sommet i , récupérer la longueur max de la liste de ses voisins et comparer le degré du sommet avec `max` et finalement `max` reçoit la valeur maximal entre `max` et le degré du sommet.

✱ Fonction histogramme graphe

Cette fonction prend en paramètre le dessin du graphe et retourne un graphique donnant pour chaque degré du graphe le nombre de sommets ayant ce degré en utilisant les fonctions prédéfinies des bibliothèques `matplotlib` et `pyplot`.

✱ Fonction nombre-chemin-longeur-2

Afin de déterminer le nombre de chemins de la longueur 2 nous avons opté pour la méthode de la matrice au carré qui retourne une matrice contenant le nombre des chemins

de longueur 2 entre deux sommets, Comme notre matrice est symétrique et que nous avons des chemins à 3 sommets tels qu'il n'y ait pas d'arêtes entre le premier et le dernier sommet du chemin. nous avons alors éliminé à partir de la matrice au carré toutes les valeurs qui ont en même temps une valeur > 0 dans la matrice d'adjacence du graphe, ainsi, on a calculé la somme par la formule suivante :

Somme matrice au carré :

\sum valeurs de la matrice au carré - \sum valeurs des éléments qui ont une valeur égale à 1 dans la matrice d'adjacence.

Somme des chemins de la longueur 2 :

$(\sum \text{valeurs de la matrice} - \sum \text{éléments de la diagonale de la matrice}) / 2$

3.3.1.2 Test et vérification

✱ Génération d'un graphe aléatoire :

Cet exemple illustre le fonctionnement de notre algorithme qui permet la génération du graphe aléatoire, en effet après avoir exécuté le programme avec les paramètres $n = 9$ on obtient la liste suivante :

0 : ['3', '9'], 1 : ['0', '2'], 2 : ['8'], 3 : ['5', '8'], 4 : ['1', '6', '8', '9'], 5 : ['0', '2', '8'], 6 : ['1', '5', '7'], 7 : ['8', '9'], 8 : ['0', '1', '5', '9'], 9 : []

ainsi que le graphe ci-dessous.

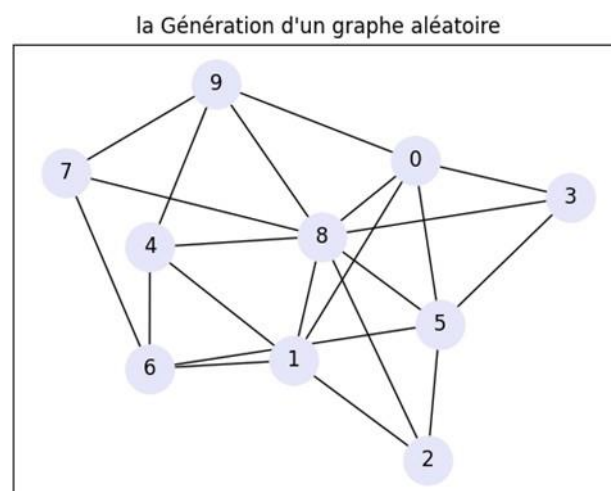


FIGURE 3.6 – Génération d'un graphe aléatoire avec 9 sommets

✱ Affichage des degrés maximales du graphe :

Cette fonction prend en paramètre l'Objet graphe et retourne le degré maximum, on prend la longueur max de la liste des voisins dans chaque sommet :

```
namespaces are one honking great idea -- let's do more of those!  
{0: ['4'], 1: ['0'], 2: [], 3: ['0'], 4: [], 5: ['6'], 6: ['0'], 7: ['0']}  
Le degré maximum du graphe est : 1
```

FIGURE 3.7 – Degré maximum dans un graphe

✱ Histogramme des graphes :

Lorsqu'on exécute l'algorithme pour le graphe ci-dessous, on obtient le résultat suivant :

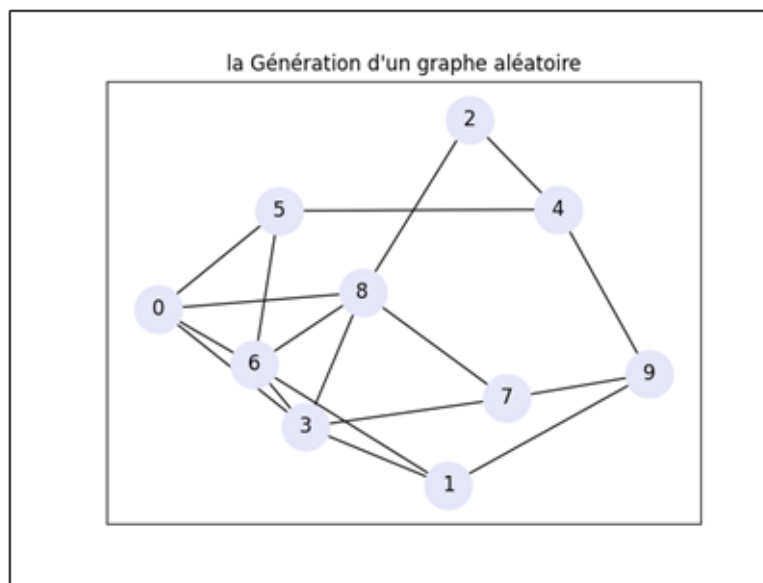


FIGURE 3.8 – Graphe aléatoire

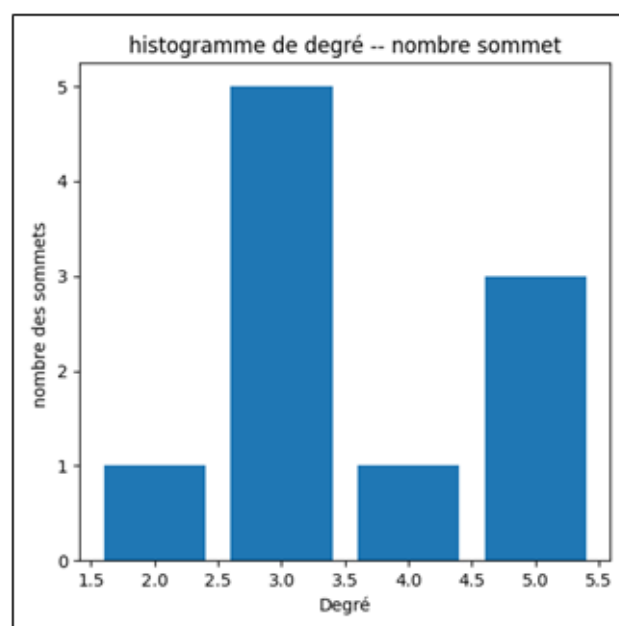


FIGURE 3.9 – Histogramme du graphe

✱ Nombre de chemins de longueur 2 :

Le résultat de la matrice se présente comme ceci :

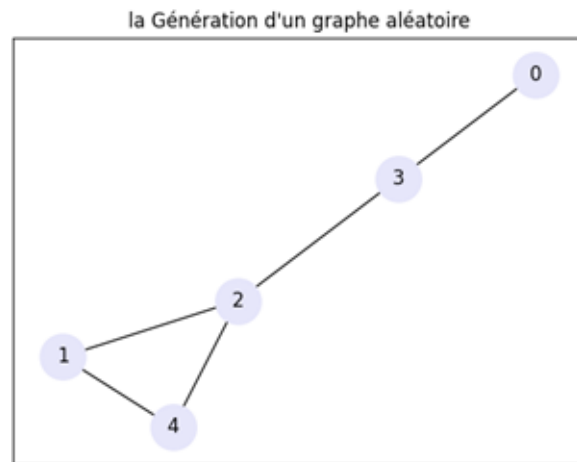


FIGURE 3.10 – Graphe aléatoire

```
(base) PS C:\Users\manar\Documents\projet_algo> py main_partie_1_aleatoire.py
{0: [], 1: [], 2: ['1', '4'], 3: ['0', '2'], 4: ['1', '2']}
matrice d'adjacence du graphe :
[[0. 0. 0. 1. 0.]
 [0. 0. 1. 0. 1.]
 [0. 1. 0. 1. 1.]
 [1. 0. 1. 0. 0.]
 [0. 1. 1. 0. 0.]]
matrice d'adjacence au carré du graphe :
[[1. 0. 1. 0. 0.]
 [0. 2. 1. 1. 1.]
 [1. 1. 3. 0. 1.]
 [0. 1. 0. 2. 1.]
 [0. 1. 1. 1. 2.]]
diagonale du matrice :
[[1. 2. 3. 2. 2.]]
nombre des chemins du longueur 2 est : 3.0
(base) PS C:\Users\manar\Documents\projet_algo>
```

FIGURE 3.11 – Matrice d'adjacence au carré

3.3.2 Génération des grands graphes de Stanford

Cette étude sur les graphes explique le processus de génération d'un graphique de réseau social pour montrer comment les amis sur sont connectés. Ce type de graphique de réseau peut être utilisé pour identifier les communautés ainsi que chaque communauté du réseau.

3.3.2.1 Implémentation

Durant cette partie, nous travaillons avec la base de données de Stanford qui existe sur l'internet.

Les bases des données sont sous la forme des fichiers txt où chaque ligne représente une liaison entre 2 sommets et arrêtes.

Nous utiliserons les mêmes fonctions expliquées précédemment.

3.3.2.2 Test et vérification

✱ Exemple d'exécution de base de données egp-Facebook :

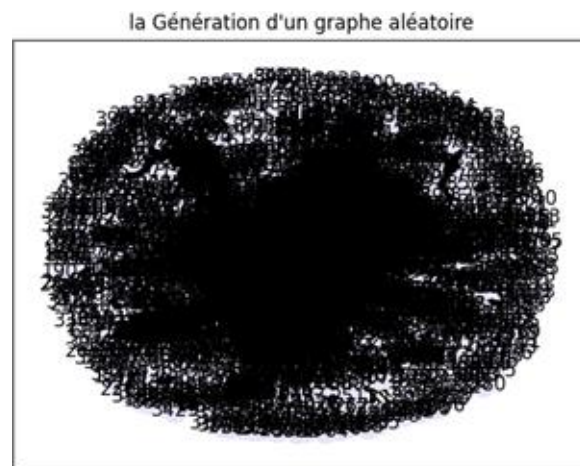


FIGURE 3.12 – Génération du graphe

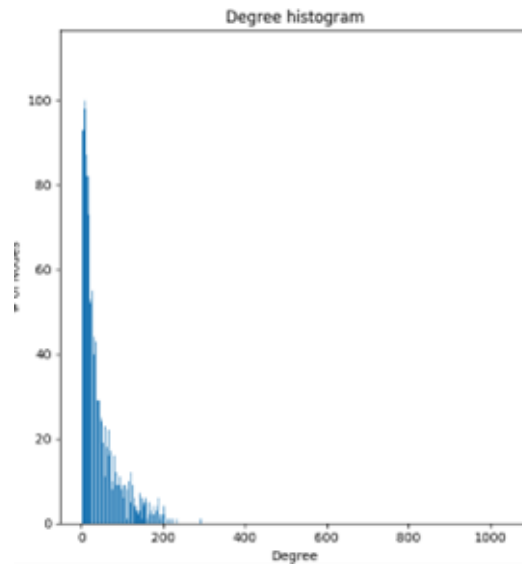


FIGURE 3.13 – Histogramme du graphe

✱ Exemple d'exécution de base de données de email-Eu-core. :

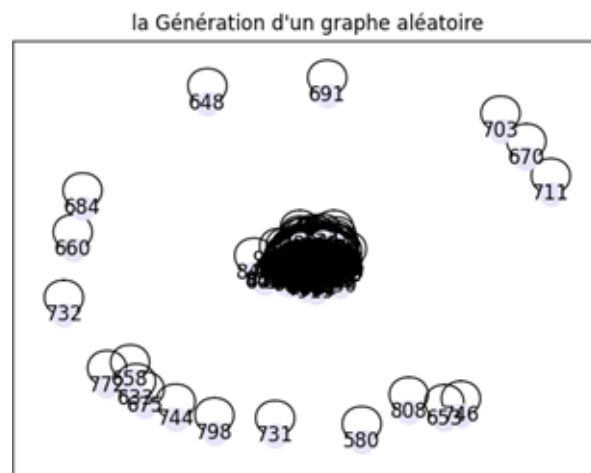


FIGURE 3.14 – Génération du graphe

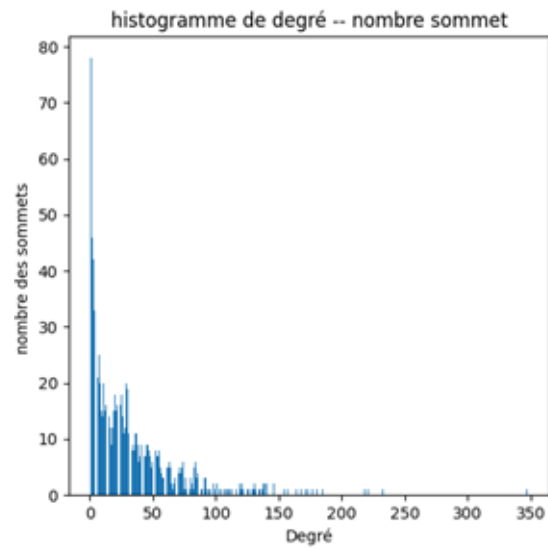


FIGURE 3.15 – Histogramme du graphe

✱ Exemple d'exécution de base de données de feather-lastfm-social :

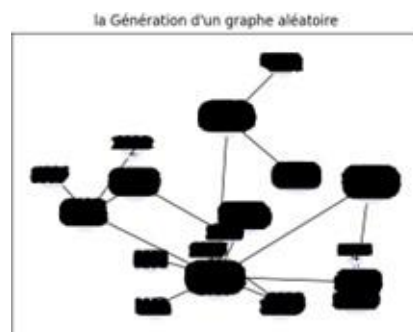


FIGURE 3.16 – Génération du graphe

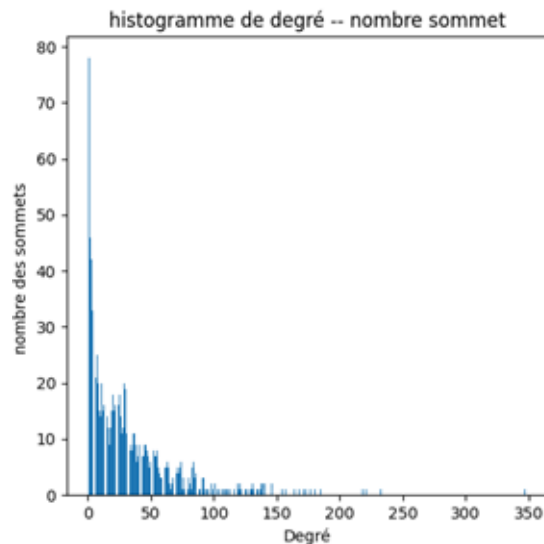


FIGURE 3.17 – Histogramme du graphe

3.4 Deuxième Partie

Dans cette partie, on est demandé d'écrire un algorithme de complexité exponentielle pour l'énumération des cliques d'un graphe.

3.4.1 Algorithme de Bron-Kerbosch

Pour cette partie on a utilisé l'algorithme du site se trouvant dans la bibliographie. L'algorithme de Bron-Kerbosch est un algorithme d'énumération de retour sur trace récursif utilisé pour trouver toutes les cliques maximales d'un graphe non orienté.

il est connu qu'un graphe à n sommets possède au maximum 3 cliques maximales. Ceci donne donc une $3^{n/3}$ borne inférieure en temps de $O(1.44^n)$ dans le pire des cas. Dans cette partie, nous expliquons l'idée de base de cet algorithme. et nous présentons la version avec pivot.

3.4.1.1 Version avec Pivot

La version avec pivot de l'algorithme est beaucoup plus efficace pour les graphes ayant un grand nombre de cliques maximales, cet algorithme se base sur le choix d'un élément pivot pour réduire le nombre d'appels récursifs et ainsi permettre un retour en arrière plus rapide.[2]

En effet, l'algorithme met en évidence que pour tout sommet $[u \in P \cup X]$, le pivot soit le sommet u lui-même, soit l'un de ses non-voisins $[P \cup x \setminus N(u)]$ doit être contenu dans toutes cliques contenant R .

Dans le cas où u ou aucun de ses non-voisins ne sont inclus dans une clique maximale contenant R , cette clique ne peut pas être maximale, puisque $u \in P \cup X$ peut être ajouté à cette clique (seuls les voisins de u ont été ajoutés à la clique originale R). Cela permet de ne pas avoir à parcourir à nouveau tout le sous-arbre et diminue donc le nombre d'appels récursifs dans la boucle 'for' donnant les cliques non maximales. La complexité de l'algorithme Pivot-tomita a prouvé que dans le cas où u est choisi dans $u \in P \cup X$ de manière que u ait le maximum de voisins, alors le temps d'exécution pour un graphe $G = (V, E)$ est en $O(3^{|V|/3})$.

3.4.1.2 Description de l'implémentation :

Pour implémenter l'algorithme Bron-Kerbosch avec pivot, nous allons devoir utiliser l'algorithme qui permet de choisir un pivot [Pivot-tomita].[2]

En effet, cet algorithme prend en entrée deux ensembles (P et X) et retourne un sommet pivot u où $P \cap N(u)$ maximale.

Au début on initialise un sommet u au premier élément de la liste $P \cup x$ puis on initialise une variable degré-max qui correspond à la taille de la liste $P \cap N(u)$. puis, on parcourt tous les sommets de la liste $P \cup X$ privé de u pour vérifier que la taille de la liste $P \cap N(v)$ [v sommet en cours] est supérieure à la valeur initiale de la variable degré-max. Si elle est supérieure, on réaffecte à v et degré-max à $P \cap N(v)$. Enfin, on retourne un pivot u avec $|P \cap N(u)|$ maximale.

On aura finalement l'algorithme qui donne un pivot u , nous allons l'utiliser pour implémenter la version avec pivot.

Au début, on vérifie si la clique R donnée est maximale, si $|P \cup X| = 0$, alors, il n'y a aucun sommet qui peut être ajouté à la clique. Donc, la clique est maximale et peut-être ajoutée à la solution. Sinon, la clique n'est pas maximale et on va créer une variable pivot lui affecte le sommet retourné par l'algorithme pivot-tomita.

Pour chaque sommet dans la liste $P \setminus N(\text{pivot})$ on va réaliser un appel vers l'algorithme Bron-Kerbosch avec pivot. le mot-clé "yield from" en Python nous permet d'enregistrer à chaque itération les cliques maximales retournées par l'algorithme appelé puis à retourner toutes ces données une fois l'algorithme terminé.

3.4.1.3 Test et vérification :

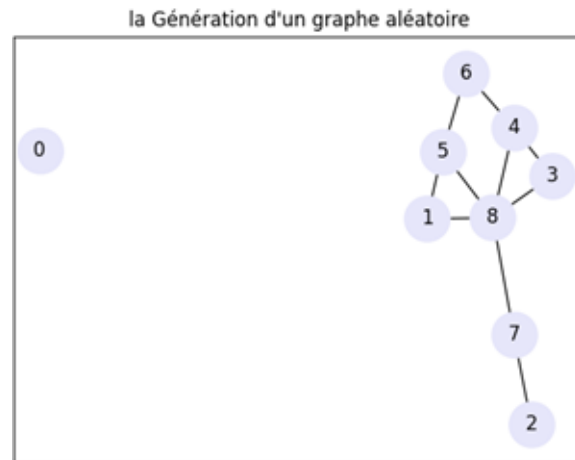


FIGURE 3.18 – Génération du graphe

```
la liste des cliques maximales est :  
[0]  
[2, 7]  
[6, 4]  
[6, 5]  
[8, 1, 5]  
[8, 3, 4]  
[8, 7]
```

FIGURE 3.19 – Génération des cliques

3.5 Troisième Partie

Dans cette partie, on est demandé d'implémenter divers algorithmes liés à l'énumération des cliques et de bicliques maximales.

Pour construire un graphe G_i . Nous avons suivi la définition qui se trouve dans le papier :

Let $G = (V, E)$ be a graph and let v_1, \dots, v_n be an ordering of its vertices. Graph $G_i, i \in [n]$, is the graph with vertex set $v_i \cup N_i(v_i) \cup N_{2i}(v_i)$ and edges :

- $xy \in G_i$, if $x \in N_i(v_i), y \in N_i(v_i)$ and $xy \in E$
- $xy \in G_i$, if $x \in N_{2i}(v_i), y \in N_{2i}(v_i)$ and $xy \in E$
- $xy \in G_i$, if $x \in N_i(v_i), y \in N_{2i}(v_i)$ and $xy \notin E$

3.5.1 Étude de l'algorithme

Pour construire un graphe G_i , il faut donc suivre cette définition. Il faut d'abord sélectionner le sommet i et ajouter ce sommet à l'ensemble des sommets de G_i . Ensuite, il faut ajouter à l'ensemble des sommets de G_i les sommets de $N_i(v_i)$, c'est-à-dire l'ensemble des sommets qui sont à une distance de 1 de i et qui ont un numéro d'ordre supérieur à celui de i . Il faut également ajouter à l'ensemble des sommets de G_i les sommets de $N_{2i}(v_i)$, c'est-à-dire l'ensemble des sommets qui sont à une distance de 2 de i et qui ont un numéro d'ordre supérieur à celui de i . Pour construire les arêtes du graphe G_i , il faut ajouter une arête entre deux sommets x et y si x et y appartiennent à $N_i(v_i)$ et si xy appartient à E , c'est-à-dire si x et y sont connectés dans le graphe G . Il faut également ajouter une arête entre deux sommets x et y s'ils appartiennent à $N_{2i}(v_i)$ et si xy appartient à E . Enfin, il faut ajouter une arête entre deux sommets x et y s'ils appartiennent respectivement à $N_i(v_i)$ et $N_{2i}(v_i)$ et si xy n'appartient pas à E .

Ligne 1 du l'algorithme :

'Consider any ordering of the vertices of G .'

D'après la définition dans le papier : l'ordre Un graphe a une dégénérescence k , ou est k -dégénéré, s'il existe un ordre v_1, \dots, v_n de ses sommets tels que pour tout $i \in [n]$.

L'ordre de dégénérescence peut être calculé en temps $O(m)$

. L'idée est essentiellement de supprimer itérativement les sommets de degré minimum jusqu'à ce que le graphe soit vide. L'ordre dans lequel les sommets sont supprimés donne l'ordre de dégénérescence.

Pour pouvoir implémenter cette ligne de l'algorithme, on a créé une fonction qui prend en entrée notre graphe et retourne en sortie la liste de dégénérescence qui satisfait la définition précédente.

En effet, la fonction a pour rôle principale de prendre un graphe G , crée deux listes : une liste D de la taille degré maximum du graphe qui prend chaque sommet du graphe et l'ajoute dans la case correspondante à son degré : `Liste[degr(sommet)].add(sommet s)` et l'autre liste, c'est la liste de dégénérescence qu'on va retourner.

Ainsi, on a créé une boucle qui parcouru la liste de degré D on commençait par le sommet ayant le degré minimum, le supprimer de liste D , l'ajouter dans la liste de dégénérescence et décaler tous ces voisins d'une case -1 dans la liste L puisque les arêtes des sommets seront supprimer d'où chaque voisin aura un degré -1, comme le montre les figures ci-dessous :

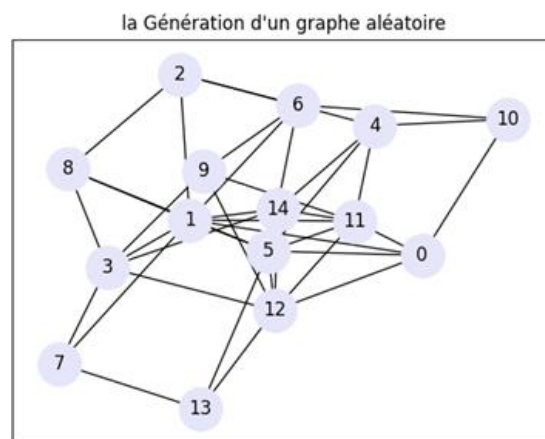


FIGURE 3.20 – Graphe généré

```
Le degré maximum du graphe est : 9
Liste degre-sommet :
[[], [], [], [7, 10, 13], [2, 8, 9], [0, 4, 6], [3, 14], [11, 12], [], [1, 5]]
La liste du degeneressnece du graphe :
[7, 10, 9, 2, 0, 3, 12, 11, 1, 5, 6, 13, 8, 14, 4]
```

FIGURE 3.21 – Liste dégénérescence du graphe

2- Construct the graphs G_i , $i \in [n]$. Pour pouvoir implémenter cette ligne de l'algorithme, on a créé une fonction qui prend en entrée notre graphe et la liste de dégénérescence retournée par la fonction précédente et on retourne en sortie tous les sous-graphes.

Pour chaque sommet i du graphe, on crée un graphe temporaire qui contient i et les sommets qui sont à une distance de 1 ou de 2 de i et qui ont un numéro d'ordre supérieur à celui de i . Ensuite, on parcourt tous les sommets du graphe temporaire et ajoute une arête entre deux sommets x et y si x et y sont connectés dans le graphe original et si x et y appartiennent soit à l'ensemble des sommets à une distance de 1 de i , soit à l'ensemble des sommets à une distance de 2 de i . Si x et y ne sont pas connectés dans le graphe original, on ajoute une arête entre eux si x appartient à l'ensemble des sommets à une distance de 1 de i et y appartient à l'ensemble des sommets à une distance de 2 de i , ou vice versa. Enfin, on ajoute le graphe temporaire à la liste de graphes et passe au sommet suivant.

Il est effectivement possible d'améliorer la complexité de l'algorithme pour construire les graphes i . Voici quelques pistes qui pourraient aider à optimiser l'implémentation :

1. Précalculer et stocker les voisins de chaque sommet et les voisins de deuxième degré de chaque sommet avant de commencer à construire les graphes i . Cela va permettre de ne pas avoir à recalculer ces informations à chaque étape de la boucle.
2. Utiliser une structure de donnée efficace pour stocker les graphes i . Par exemple, utiliser une matrice d'adjacence plutôt qu'une liste d'adjacence si les graphes i sont assez denses.
3. Utiliser des algorithmes de parcours de graphe efficaces pour parcourir les sommets et les arêtes du graphe principal et des graphes i . Par exemple, utiliser un parcours en profondeur plutôt qu'un parcours en largeur si le graphe est assez dense.

Il est possible que ces optimisations ne suffisent pas à atteindre la complexité en $O(n^3)$ ou en $O(m\Delta)$ mentionnée dans l'article, mais elles devraient quand même permettre de réduire considérablement la complexité de votre algorithme.

3.6 Comparaison de graphes par rapport au temps

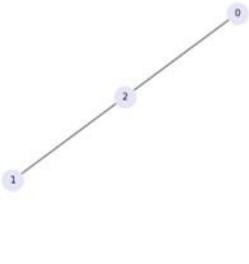
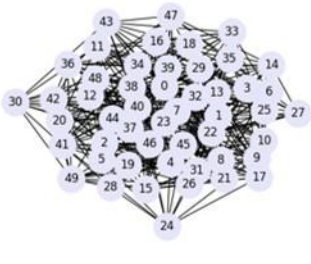

Numéro de la partie	Nombre des sommets		
	3 sommets	50 sommets	100 sommets
Dessin du graphe			
Partie 1	0.3259730339050293 secondes	0.40134644508361816 secondes	0.4696190357208252 secondes
Partie 2	0.18199920654296875 secondes	0.2059917449951172 secondes	0.23758769035339355 secondes
Partie 3	0.19400572776794434 secondes	0.227752685546875 secondes	0.3147776126861572 secondes

FIGURE 3.24 – Comparaison de graphes par rapport au temps

3.7 Comparaison de graphes par rapport à l'espace

Numéro de la partie	G1	G2	G3
Partie 1	$O(n+m)$	$O(n+m)$	$O(n+m)$
Partie 2	$O(3^n * n)$	$O(3^n * n)$	$O(3^n * n)$
Partie 3	$O(n^2)$	$O(n^2)$	$O(n^2)$

CONCLUSION GÉNÉRALE

C E projet nous a été aussi une excellente opportunité pour appréhender le travail dans un cadre académique.

En effet, il nous a permis de s'intégrer facilement dans le travail collaboratif et de faire face aux difficultés comme la répartition des tâches et la gestion du temps et des efforts.

Nous avons atteint notre objectif principal qui est la lecture et la compréhension des articles scientifiques fournis avec le sujet. Néanmoins, nous avons rencontré plusieurs difficultés concernant l'étude des différents algorithmes. Mis à part la compréhension de l'algorithme de Bron-Kerbosch, nous avons rencontré des soucis qui nous ont permis d'agir autrement dans la réalisation du projet.

par exemple, au niveau de complexité en temps lors de déploiement des grands graphes de Stanford.

L'autre souci majeur que nous avons rencontré, c'était au niveau de l'énumération des cliques et bicliques maximales de la troisième partie qui dépendaient principalement de la dégénérescence du graphe.

Pour conclure, nous avons approfondi nos connaissances en théorie de graphe.

Le sujet est très intéressant et compte également des perspectives importantes ainsi que des pistes de développement et de recherche qui pourraient être utilisées.

RÉFÉRENCES BIBLIOGRAPHIQUES

- [1] ISTY. Présentation de l'isty. <https://fr.wikipedia.org/wiki/Institut_des_sciences_et_techniques_des_Yvelines>, 2017. [En ligne; accès le 02-Novembre-2022].
- [2] Pivot-tomita. Résolution de problèmes de cliques dans les grands graphes. <https://hal.archives-ouvertes.fr/hal-01886724/document>, 2019. [En ligne; accès le 10-décembre-2022].