UVSQ université PARIS-SACLAY | ISTY Institut des Sciences et Techniques des Yvelines
CAMPUS DE MANTES EN YVELINES
CAMPUS DE SAINT-QUENTIN-EN-YVELINES

CALCUL HAUTE PERFORMANCE SIMULATION

ISTY UVSQ
Université Paris-Saclay

M1 ***C****alcul* ***H****aute* ***P****erformance,* ***S****imulation*
Projet de Programmation Numérique
Rapport de Projet

Recherche de Cliques Maximales dans des Graphes Creux

Réalisé par :

Mme. Rim ACHRRAB
Mme. Katia CHIKHI
Mme. Nedjma BOUAMARA

Encadré par :

M. manoussakis George

Promotion : 2022/2023

Remerciements

C'est avec un grand plaisir autant qu'un devoir de remercier nos professeurs du master 1 calcul haute performance,simulation, qu'ils ont été toujours à la hauteur afin de nous fournir une meilleure formation.

Nous réservons ces lignes en signe de gratitude et de profonde reconnaissance à l'égard de tous nos professeurs qui nous ont aidés à la réalisation de notre projet.

Nos gratitudes s'adressent à : Monsieur le professeur MANOUSSAKIS George qui nous a encadré durant notre réalisation de la première partie de ce travail.

Monsieur le professeur Hugo BOLLORÉ pour ses conseils prodigues en ce qui concerne le module de Projet de Programmation Numérique.

Nos responsables Messieurs les professeurs Pablo DE OLIEVEIRA et Thomas DUFAUD, pour ses efforts et ses intérêts envers ses étudiants.

Nous remercions les membres de jury d'avoir eu l'aimable volonté de juger et d'évaluer notre travail, de nous accorder une grande partie de leur temps précieux pour lire notre travail et l'enrichir par leurs remarques et questions pertinentes.

Résumé

Le problème de recherche de cliques maximales (MCE) est un problème très souvent présent dans l'analyse et l'exploitation des graphes non-orientés. C'est un problème NP-difficile dont la difficulté de trouver les cliques maximales augmente en fonction de la taille du graphe étudié.

Nous nous proposons dans ce petit travail l'algorithme de Bron Kerbosh pour résoudre le problème d'énumération de cliques maximales en utilisant la dégénérescence et l'ordre de dégénérescence du graphe, cette approche nous semble intéressante en ce qui concerne le temps de son éxecution et son espace en mémoire.

Mots clés : Graphe non orienté, MCE, Clique, Algorithme

Abstract

The maximum clique enumeration (MCE) problem is very often present in data graph analysis and mining. It is an NP-difficult problem, in fact, finding the maximum cliques becomes harder in the case of large graphs.

In the first part of our project, we propose a solution by using a new version of Bron Kerbosh algorithm based on the degeneracy and the degeneracy ordering of the undirected graph, this approach will be more interesting to us -as high-performance computing students- in term of computation time and memory space.

Keywords : Undirected graph, MCE, Clique, Algorithm

Table des matières

Table des figures

Liste des abréviations

MCE	Maximal Clique Enumeration
VM	Virtual Machine

Introduction générale

Les graphes non orientés peuvent être utilisés en tant que structures de données qui nous permettent la modélisation des objets ou des problèmes avec de nombreuses applications et utilisations, notamment dans les domaines des réseaux sociaux, des réseaux de transport ou encore les domaines de la biologie. Parmi ces applications se trouve la recherche de structures,On peut par exemple chercher des communautés dans les graphes issus des réseaux sociaux, etc.

Dans notre projet, nous nous intéresserons à la recherche de cliques maximales dans des graphes creux.

Une clique est un sous-graphe du graphe principal comportant des noeuds qui sont tous connectés entre eux.

Le problème d'énumération de cliques maximales ou MCE est parmi les grands problèmes qui existent dans la recherche de cliques, c'est un problème algorithmique qui s'intègre dans le domaine de la théorie des graphes et il consiste à lister toutes les cliques maximales au sein d'un graphe non orienté.

Le problème d'énumération de cliques maximales MCE est un problème NP-difficile, ce qui implique qu'il n'existe pas un algorithme plus efficace qui permet de trouver toutes les cliques maximales et résoudre le problème pour des graphes d'ordre assez grand. Les solutions donc vont être approchées et non pas exactes [13].

Chapitre 1

1 Contexte général du projet

1.1 Introduction

Dans ce chapitre, nous allons élaborer le cadre général du projet. Cette étape nous permet de formaliser et spécifier le contexte de notre travail dont on va lancer la problématique, les objectifs et les notions de bases de la théorie des graphes.

1.2 Recherche de cliques maximales

Pour rechercher les cliques maximales dans des structures de données, maints algorithmes existent pour résoudre le problème MCE. L'algorithme de Bron Kerbosh est un algorithme d'énumération qui a été créé en 1973 pour trouver toutes les cliques maximales dans un graphe non orienté.

Il peut être utilisé pour détecter les communautés étroites dans des réseaux sociaux comme Facebook en traitant chaque personne comme un nœud et chaque lien d'amitié comme un arc reliant les deux personnes dans le graphe.

Avant d'appliquer l'algorithme de Bron Kerbosh, il faut d'abord créer la représentation graphique qui convient le réseau social.Une fois que vous avez la représentation graphique, vous pouvez ensuite appliquer l'algorithme de Bron-Kerbosch pour trouver les cliques dans le graphique.

L'algorithme de Bron-Kerbosch est un moyen efficace pour détecter les communautés étroites dans les grands réseaux sociaux, car il a une complexité temporelle de $O(3^{(n/3)})$ et $O(n^2)$ en mémoire, où n est l'ordre du graphe ($n \in \mathbb{N}$). Cependant, il convient de noter que l'algorithme ne peut détecter que les cliques et peut ne pas être en mesure d'identifier des types plus généraux des communautés étroites.

1.3 Problématique

L'objectif de notre travail est l'implémentation d'un algorithme qui va être une résolution optimale pour le problème de clique MCE.

Ce projet propose des solutions aux problèmes suivants :

Problème 1 : Dans les grands graphes (les réseaux de transports ou les réseaux sociaux), il est plus difficile de trouver manuellement toutes ses cliques maximales.

Problème 2 : Les algorithmes existants pour trouver les cliques maximales ne sont pas optimisés (problèmes de complexité en temps et en mémoire), ce qui rend le problème beaucoup plus complexe dans des graphes de grande taille.

Ceci nécessite de mettre en présence et de faire sortir en monde réel un algorithme en python capable de résoudre le problème de cliques maximales dans un graphe donné, qui pourra être plus efficaces en terme de complexité.

1.4 Objectifs

Après avoir détecté les problèmes, voilà nos objectifs à atteindre :

— L'objectif principal de ce projet consiste à modéliser une solution informatique qui va mettre en présence un algorithme en python qui va détecter toutes les cliques maximales contenues dans un graphe non orienté.

— La mise en place d'une structure de données idoine pour la représentation graphique afin de faciliter l'accès aux informations désirées.

— Réduire l'intervalle de temps de l'exécution de l'algorithme pour qu'on puisse rapidement trouver les sous-graphes complets du graphe principal.

1.5 Concepts fondamentaux

1.5.1 La théorie des graphes

La théorie des graphes est un domaine polyvalent de base mathématique qui s'applique en plusieurs domaines informatiques,elle est dédiée à l'étude des graphes et leurs applications.Elle a été créée par le mathématicien suisse Leonhard Euler en 1774 afin de mettre des relations entre plusieurs éléments et faciliter les présentations des phénomènes en théorie [21].

1.5.2 Définition d'un graphe

Un graphe est une structure de données définie par un couple G = (S, A) tel que :

— S est un ensemble fini de sommets (ou nœuds), un sommet est tout simplement n'importe quelle entité représentée par les données (par exemple : un utilisateur de réseaux sociaux).

— A est un ensemble d'arêtes ou des relations reliant ces sommets, tel que :
$A \subseteq \{(x, y)|x, y \in Setx \neq y\}$.

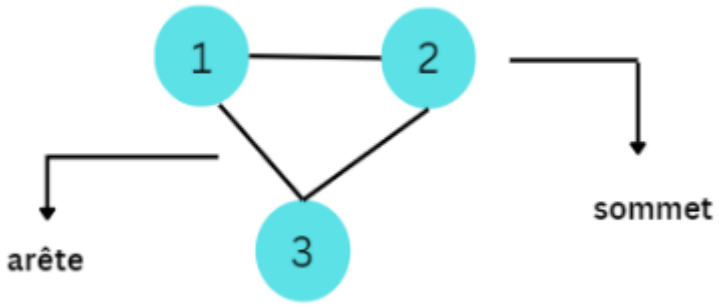

FIGURE 1 – Un graphe avec ses arrêts et ses sommets

1.5.3 Graphe non-orienté

Un graphe est dit non orienté si les arêtes n'ont pas de sens entre deux sommets, Cela signifie que si un sommet S_1 est relié à un autre sommet S_2 par un arc, ils sont également en relation suivant les deux sens (S_1 vers S_2 et S_2 vers S_1).

Un graphe non orienté représente une relation symétrique, en fait $(s_1, s_2) = (s_2, s_1)$.

Si l'ensemble des sommets S est un ensemble fini, on dit que G est un graphe non-orienté fini.

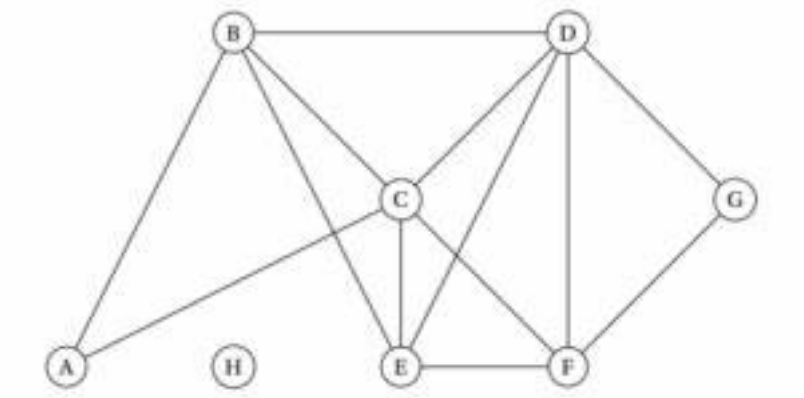

FIGURE 2 – Graphe non-orienté [10]

1.5.4 Graphe complet

un graphe G est dit complet s'il comporte une arête (s_i, s_j) pour toute paire de sommets différents $(s_i\ ,\ s_j) \in S$ (S est l'ensemble des sommets contenants dans le graphe G).

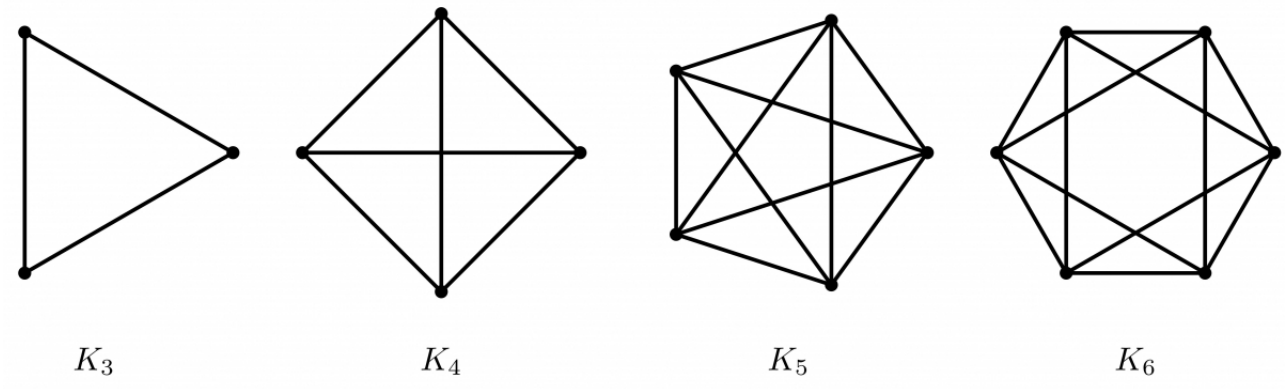

FIGURE 3 – Exemples de graphes complets [9]

1.5.5 Ordre d'un graphe

L'ordre d'un graphe est défini comme le nombre fini de sommets qu'il contient. Si un graphe possède six sommets alors il est d'ordre six.

L'ordre du graphe est important dans notre travail, en fait, il influence la complexité des algorithmes utilisés car certains algorithmes peuvent être plus efficaces pour des graphes de petite taille que pour des graphes de grande taille.

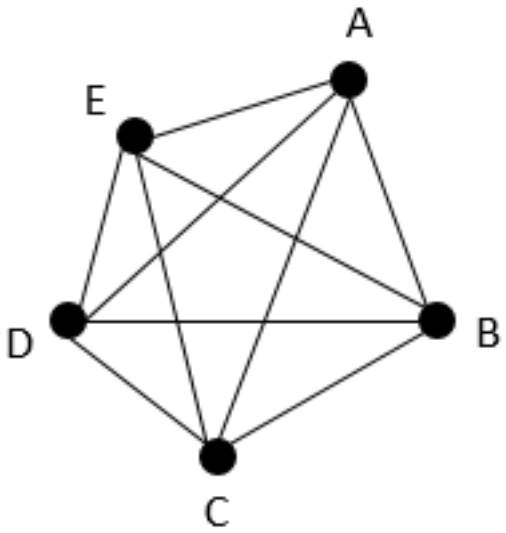

FIGURE 4 – Graphe d'ordre 5 [8]

1.5.6 Degré d'un sommet

1.5.6.1 Définition Soit G=(S,A) un graphe non-orienté fini. Le degré d'un sommet $s_i, i \in \{1, 2, ..., n\}$ avec $n \in \mathbb{N}$, que l'on note $d(s_i)$ est le nombre d'arêtes dont lesquels il est relié.
En utilisant un définition mathématique, le degré d'un sommet est défini comme suit :

$$d(s_j) = \{\{s_i, s_j\} \in A / s_i \in S\}$$

1.5.6.2 Exemple Dans le graphe ci-dessous, les degrés des sommets $\{0, 1, 2, 3\}$ sont les suivants :
d(0)=d(3)=3, d(1)=2, et d(2)=2.

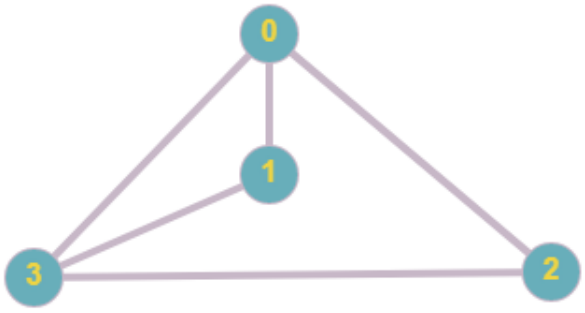

FIGURE 5 – Degré des sommets d'un graphe [1]

1.5.7 Voisinage d'un graphe

Dans un graphe G, si deux sommets s_i et s_j sont attachés à une arête, on dit que s_j est un voisin de s_i (on peut dire aussi que s_j et s_i sont adjacents).

L'ensemble de voisinage d'un sommet d'un graphe est noté N tel que :

— $N(s_i)$: désigne l'ensemble de voisinage ouvert du sommet s_i qui est composé des sommets adjacents à s_i dans le graphe G sans inclure le sommet s_i.

— $N[s_i]$: est l'ensemble $\{N(s_i)Us_i\}$ qui est défini comme le voisinage fermé de s_i.

On cosidère le graphe G ci-dessous :
Le voisinage ouvert de 1 est l'ensemble $\{2, 3, 5\}$
Le voisinage ouvert de 2 est l'ensemble $\{1, 3, 4\}$
Le voisinage ouvert de 3 est l'ensemble $\{1, 2\}$
Le voisinage ouvert de 4 est l'ensemble $\{2\}$
Le voisinage ouvert de 5 est l'ensemble $\{1\}$

En outre, le voisinage fermé de 1 est l'ensemble $\{2, 3, 5\} \cup \{1\} = \{1, 2, 3, 5\}$

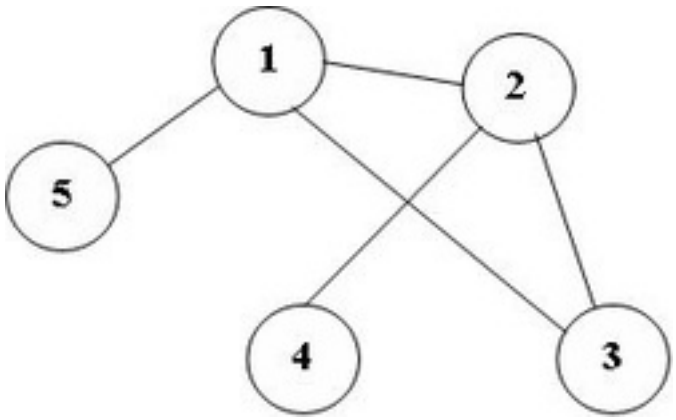

FIGURE 6 – Voisinage d'un graphe non orienté [2]

1.5.8 Sous-graphe

Soit G=(S,A) un graphe qui contient un ensemble fini S de sommets et A d'arrêts. On dit que $G' = (S', A'$ est un sous-graphe de G si et seulement si :

— S $\subseteq S'$

— A $\subseteq A'$

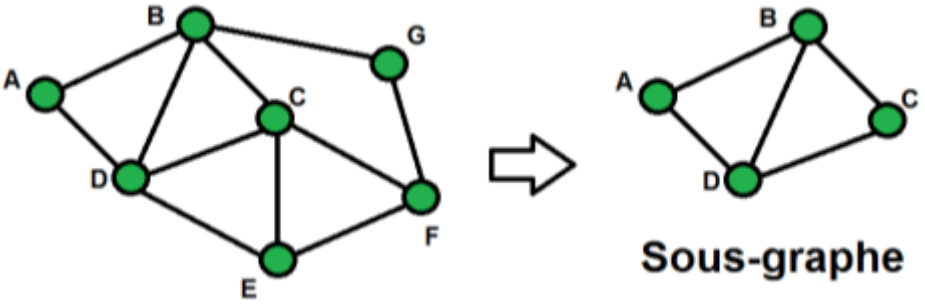

FIGURE 7 – Le graphe $\{A, B, C, D\}$ est un sous-graphe [15]

1.5.9 Graphe creux

Un graphe creux (sparse graph en anglais) est un graphe dont ses sommets ont des degrés faibles (identiquement équivalent les sommets ne sont pas connectés à plusieurs voisins).

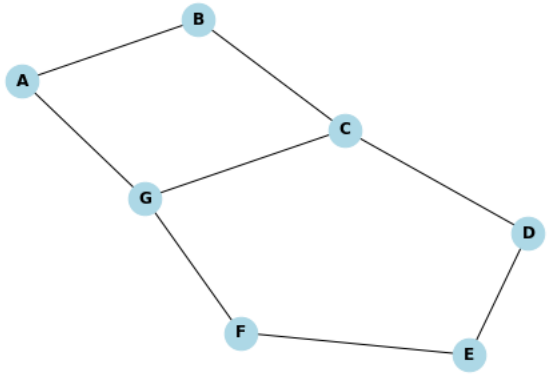

FIGURE 8 – Graphe creux [11]

1.5.10 Sous-graphe induit

Etant donné un graphe non orienté G=(S,A) d'ordre fini n. Si on supprime des sommets $s_i, i \in (0, 1, ..., n)$ en éliminant tous ses arcs qui partent ou qui arrivent de ces sommets, on obtient un sous-graphe induit de G.

Dans le graphe G=(S,A) ci-dessous, on a : $S = \{C, D, E, F\}$
et $A = \{(C, D), (C, E), (C, F), (D, E), (D, F), (E, F)\}$

- Le sous-graphe qui a pour sommets l'ensemble $S1 = \{C, D, E\} \subset S_1$ et pour arêtes l'ensemble $A_1 = \{(C, D), (C, E), (D, E)\} \subset A$ est un sous-graphe induit de G.
- Le sous-graphe qui contient les sommets C,D,E,G n'est pas un sous-graphe induit de G, car le sommet G n'est pas inclu dans l'ensemble S.

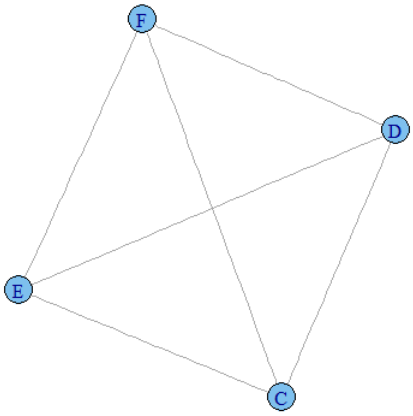

FIGURE 9 – Sous-graphe induit [23]

1.5.11 Clique d'un graphe

Une clique dans un graphe G est un sous-ensemble de sommets de G tel que chaque sommet est relié à tous les autres par des arêtes (ou arcs), une clique est donc un sous-graphe complet d'un graphe donné G.

Dans l'exemple ci-dessous, on remarque que l'ensemble $\{1, 2, 5\}$ forme une clique.

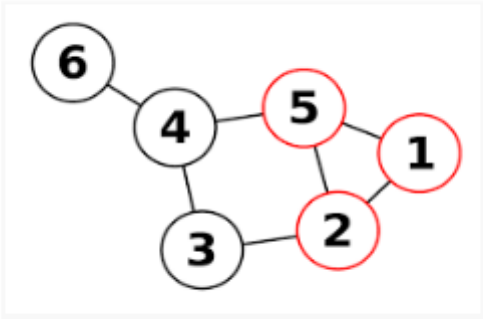

FIGURE 10 – Clique d'un graphe [25]

1.5.12 Clique maximale d'un graphe

Une clique est dite maximale si elle n'est pas incluse dans une clique de taille supérieure. c'est-à-dire qu'elle ne peut pas être étendue en ajoutant un sommet supplémentaire.

Dans l'exemple ci-dessous :

— $\{1, 2, 5\}$ est une clique maximale

— $\{2, 3, 4\}$ n'est pas une clique maximale car elle est incluse dans l'ensemble $\{2, 3, 4, 6\}$.

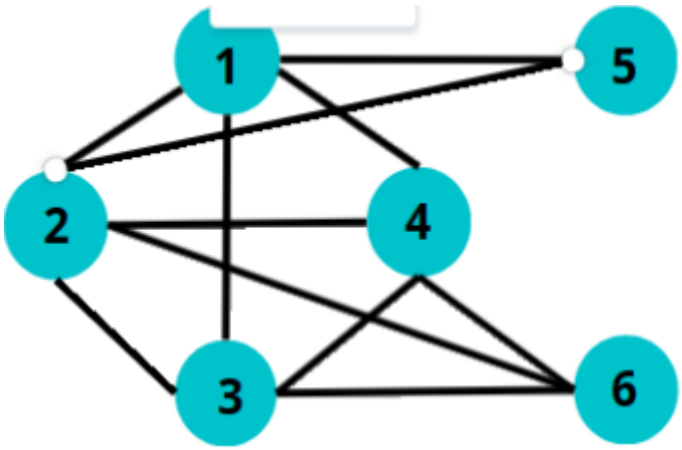

FIGURE 11 – Clique maximale dans un graphe

1.5.13 Dégénéréscence d'un graphe

La dégénérescence d'un graphe est une mesure qui est associée à un graphe non orienté, elle permet de mesurer à quel point un graphe est creux.
On dit qu'un graphe G est k-dégénéré si pour tout sous-graphe H de G, H contient au moins un noeud de degré inférieur ou égal à k, donc $\delta(H) \leq k$ (avec $\delta(H)$ le degré minimum des noeuds de H). La dégénérescence de G est le plus petit nombre entier k tel que G est k-dégénéré.

1.5.14 Ordre de dégénéréscence d'un graphe

L'ordre de dégénérescence, noté σ_G est un ordre sur les sommets qu'on obtient en supprimant à chaque fois le sommet qui a le plus petit degré dans le sous-graphe restant.

1.5.15 Exemple

Prenons ce graphe comme exemple :

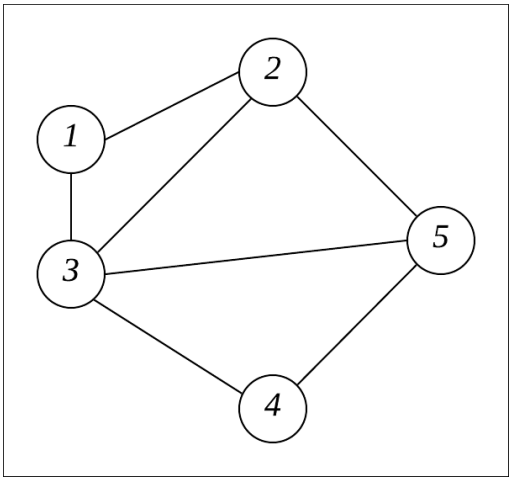

FIGURE 12 – Un graphe 2-dégénéré [7]

a. Calcul de la dégénérescence

Pour $k = 0$: On supprime les sommets de degré inférieur à 0.
Pour $k = 1$: On supprime les sommets de degré inférieur à 1.
Pour $k = 2$: On elimine tous les sommets de degré inférieur à 2 (dans cet exemple, on supprime les sommets 1 et 4).
On s'arrête à cette étape, car si k=3, le sous-graphe résultant est vide.
Donc, le plus petit entier k pour que le graphe G soit k-dégénéré est k=2.

b. Calcul de l'ordre de dégénérescence

L'ordre de dégénérescence du graphe est alors l'ensemble $\{1, 4, 2, 3, 5\}$ (En supprime à chaque fois le sommet qui a le plus petit degré dans l'ensemble du graphe jusqu'à ce qu'on termine l'ensemble des sommets du graphe donné).

1.5.16 Table de hashage

Une table de hachage est une structure de données qui permet une association clé–valeur. En utilisant une table de hachage, nous sommes en mesure de définir chaque sommet particulier comme une clé de hachage, et son appariement de valeur à un tableau de sommets adjacents.La raison de l'utilisation d'une table de hachage est la recherche rapide des liens entre les sommets dans les graphes qui fait (O(1)) en temps.

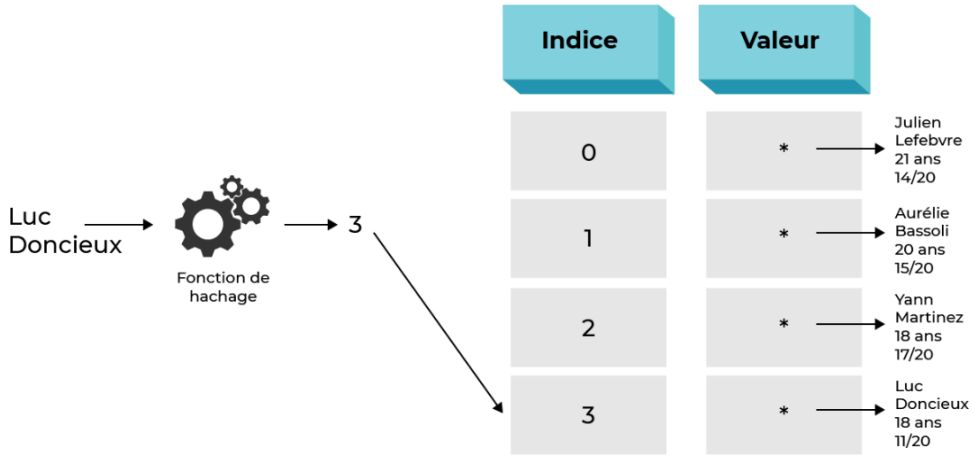

FIGURE 13 – Un exemple de table de hashage [19]

1.6 Conclusion

Dans ce chapitre, on a vu des notions de base de la théorie des graphes qui nous seront importantes pour comprendre le problème des clique maximales dans un graphe et savoir implémenter l'algorithme convenable tout en gardant l'optimalité des mesures de performance.

Chapitre 2

2 Implémentation

2.1 Introduction

Après avoir cité la méthode utilisé pour résoudre le problème de cliques, on va d'abord expliquer en détails l'algorithme de Bron Kerbosh pour pouvoir ensuite l'implémenter sous python.

Nous discuterons également les outils de développement, et le langage de programmation utilisés.

2.2 Algorithme Pour le problème MCE

Algorithm 1:

Data: A graph G.
Result: All the maximal cliques of G.

```
1  Compute k the degeneracy of G and σ_G.
2  Compute the degenerate adjacency lists of G.
3  Initialize T an empty generalized suffix tree.
4  for j = 1 to n do
5      Compute all maximal cliques of graph G_j.
6      for every maximal clique K of graph G_j do
7          Order the vertices of K following σ_G
8          Search for K in T.
9          if there is a match then
10             Reject it.
11         else
12             Insert the proper suffixes of K in T.
13             Output K.
```

FIGURE 14 – Algorithme 1 [17]

L'algorithme ci-dessus, issu de l'article [17] est la base de notre programme python.

Cet algorithme va avoir comme résultat toutes les cliques maximales d'un graphe donné sans duplication :

— **Ligne 1 :** dans un premier temps, on va implémenter toutes les propriétés nécessaires des graphes telles que : algorithme qui calcul le degrés des sommets du graphe, la dégénérescence k et l'ordre de dégénérescence σ_G d'un graphe non orienté.

— **Ligne 2 :** La ligne 2 est à supprimer.

— **Ligne 3 :** On va initialiser une table de hachage à la place de l'arbre des suffixes vu qu'elle est moins coûteuse en temps et en mémoire.
Pour stocker une paire clé/valeur, nous pouvons utiliser un tableau simple comme une structure de données où les clés peuvent être utilisées directement comme index pour stocker des valeurs. Mais dans le cas où les clés sont volumineuses et ne peuvent pas être directement utilisées comme index pour stocker la valeur, nous pouvons utiliser une technique de hachage. Dans le hachage, les grandes clés sont converties en petites, en utilisant des fonctions de hachage, puis les valeurs sont stockées dans des structures de données appelées tables de hachage [14].

— De la ligne 4 jusqu'à 13 :
D'abord on doit diviser le graphe G d'ordre n $n \in \mathbb{N}$ en sous-graphes notés G_j, $j \in [n]$ pour qu'on puisse simplifier la recherche des cliques maximales dans le graphe.
a. Graphe G_j : On considère un graphe non orienté G=(S,E) d'ordre n, comportant un ensemble de sommets $S = \{s_1, s_2, ..., s_n\}$. Le graphe G_j, $j \in \{1, ..., n\}$,est le sous-graphe induit de G tel que $G_j = G[N[s_j \cap S_j]$, avec $N[s_j] \cap S_j \subset S$ et $S_j = \{s_j, s_{j+1}, ..., s_n\}$.
Puis, On va introduire l'algorithme de Bron Kerbosh pour trouver les cliques maximales et les imprimer sans duplication- (ou les sous-graphes complets K $K \in \{G_1, ..., G_n\}$) dans l'ordre de dégénérescence.

L'algorithme de Bron Kerbosh avec pivot, qu'on va l'utiliser dans l'algorithme de Bron Kerbosh avec ordre de dégénérescence est le suivant :

```
algorithme BronKerbosch2(R, P, X)
    si P et X sont vides alors
        déclarer R clique maximale
    choisir un sommet pivot u dans P ⋃ X
    pour tout sommet v dans P \ N(u) faire
        BronKerbosch2(R ⋃ {v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

FIGURE 15 – Algorithme 1.2 [24]

— On initialise trois ensembles tels que :
P : l'esemble des sommets du graphe.
R : l'ensemble des cliques maximales du graphe.
X : l'ensemble des sommets déjà visités.

— Si les ensembles P et X sont vides alors R est une clique maximale du graphe.

— Si ce n'est pas le cas, on choisi un sommet pivot u $\in P \cup X$ contenant les sommets restants.

— En utilisant une boucle for, on cherche tous les sommets v qui ne sont pas adjacents à u dans P.

— L'algorithme utilise un appel récursif de la même fonction en utilisant les paramètres : la réunion de l'ensemble R et le sommet v, l'intersection de l'ensemble P et les sommets adjacents à v, et l'intersection de l'ensemble X et les sommets adjacents à v. [24]

— Par la suite, on supprime le sommet v de l'ensemble P et on l'ajoute dans l'ensemble X des sommets visités, et ainsi de suite jusqu'à l'obtention des cliques maximales du graphe.

L'algorithme ci-dessous est l'algorithme de Bron kerbosh avec ordonnancement des nœuds (c'est l'algorithme qu'on utilisera par la suite dans notre code).

```
algorithme BronKerbosch3(G)
    P = V(G)
    R = ∅
    X = ∅
    pour tout sommet v visités dans un ordre de dégénérescence de G faire
        BronKerbosch2({v}, P ⋂ N(v), X ⋂ N(v))
        P := P \ {v}
        X := X ⋃ {v}
```

FIGURE 16 – Algorithme 2.2 [24]

On a utilisé cet algorithme parce qu'il est plus rapide que l'algorithme de Bron Kerbosh de référence (il utilise dégénérescence du graphe pour trouver les cliques maximales).

— On parcourt tous les sommets du graphe suivant son ordre de dégénérescence en utilisant une boucle for. On rappelle que l'ordre de dégénéréscence d'un graphe est un ordre sur ses sommets qui hiérarchise les sommets de degré faible. Cet algorithme peut rapidement supprimer les sommets qui ne sont pas inclus dans les cliques maximales du graphe.

— Pour chaque sommet v visité, l'algorithme appelle récursivement la fonction BronKerbosch2 avec les paramètres suivants : un ensemble contenant seulement le sommet v, l'intersection de P et l'ensemble des sommets adjacents à v, l'intersection de X et l'ensemble des voisins de v.

— On enlève le sommet v visité de l'ensemble P des sommets restants, et on l'ajoute X, et ainsi de suite.

2.3 Environnement de travail

2.3.1 Environnement matériel (Hardware)

Nous avons utilisé pour ce projet trois ordinateurs ayant les caractéristiques suivantes :
Ordinateur portable : HP EliteBook 840 G5

- Système d'exploitation : Windows 10 Professionnel 64 bits
- Processeur : intel(R) Core(TM) i5-7300U CPU @ 2.60 GHZ
- Mémoire RAM : 8GB
- Carte mère : HP 83B2 KBC version 23.53.00

Ordinateur portable : HP laptop 15s-fq2xxx 11th Gen

- Système d'exploitation : Windows 11 Professionnel 64 bits
- Processeur : Intel(R) Core(TM) i3-1125G4 @ 2.00GHz
- Mémoire RAM : 8GB
- Espace disque : 256 GB SSD

Ordinateur portable : Hewelett-Packard HP 250 G2 Notebook PC

- Système d'exploitation : Ubuntu 22.04.1 LTS 64 bits
- Processeur : Intel(R) Core(TM) i3-3110M CPU @ 2.40GHz x4
- Mémoire RAM : 6GB
- Carte graphique : Mesa Intel(R) HD Graphics 4000 (IVB GT2)
- Espace disque : 120 GB

2.3.2 Environnement logiciel (Software)

Machine Virtuelle

Une machine virtuelle (VM) est un système informatique simulé qui tourne sur une machine physique (appareil ordinateur). Cette VM dispose de son propre système d'exploitation et de matériel virtuel comme un processeur, de la mémoire vive, un disque dur et une carte réseau, tout comme un ordinateur physique [20].

FIGURE 17 – Logo d'une VM [12]

Linux

Linux est un système d'exploitation libre et open-source, basé sur Unix. Il a été créé par Linus Torvalds en 1991.Il est conçu pour être utilisé sur une grande variété de périphériques, et il est largement utilisé pour les serveurs, les superordinateurs, les ordinateurs de bureau, les smartphones et les dispositifs embarqués [26]

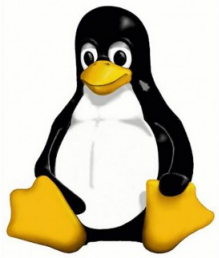

FIGURE 18 – Logo Linux [22]

Visual Studio Code

Lancé en 2015 par le géant Microsoft, Visual Studio Code est très vite devenu un des logiciels populaires auprès des programmeurs, gratuit et open source ce qui est rare chez Microsoft. VS Code n'est pas seulement facile à utiliser ou qu'il prend en charge plusieurs langages, mais aussi il donne accès à des extensions téléchargeables, des thèmes adaptés à l'utilisateur, les raccourcis clavier, coloration des mots-clés du code, le Debugging etc.

FIGURE 19 – Logo Visual Studio Code [3]

LaTeX

LaTeX est un système de traitement de texte et préparation de documents pour une composition de haute qualité. Il est le plus souvent utilisé pour des documents techniques ou scientifiques de différentes tailles.

FIGURE 20 – Logo Latex Project [4]

Python

Python est un langage de programmation interprété de haut niveau, créé par Guido van Rossum en 1991. Il est largement utilisé par la communauté des développeurs en raison de la productivité accrue qu'il offre . Il prend en charge plusieurs paradigmes de programmation, y compris la programmation structurée, orientée objet et fonctionnelle. Python est utilisé dans tous les domaines, de l'apprentissage automatique à la création de sites Web, en passant par le test de logiciels. Il est multiplateforme et ses programmes peuvent s'exécuter sous Windows,Linux et macOS.

FIGURE 21 – Logo Python [16]

2.4 Le code réalisé

Notre code entier est disponible dans le dépôt Github : `https://github.com/KatiaCHIKHI/PPN.git`

Le code est composé des fonctions suivantes :

Fonction de création de la table de hashage

En python, on représente les tables de hashage avec des dictionnaires qu'on peut les créer soit en utilisant des accolades, soit la fonction *dict*().

Fonction d'ajout d'un sommet

Dans la table de hashage, les sommets sont considérés comme des clés.

```
0   def addSommet(self, somm):
1           if somm not in self.gdict:
2               self.gdict[somm] = []
```

Fonction d'ajout d'un arc

```
0   def addArc(self,arc):
1           arc = set(arc)
2           (somm1, somm2) = tuple(arc)
3           if somm1 in self.gdict:
4               self.gdict[somm1].add(somm2)
5           else:
6               self.gdict[somm1] = set(somm2)
7           if somm2 in self.gdict:
8               self.gdict[somm2].add(somm1)
9           else:
10              self.gdict[somm2] = set(somm1)
```

La fonction getArcs nous permet de récupérer les arcs d'un graphe donné :

```
0    def getArcs(self):
1            arcs = []
2            for somm in self.gdict:
3                for nxtsomm in self.gdict[somm]:
4                    if {nxtsomm, somm} not in arcs:
5                     arcs.append({somm, nxtsomm})
6            return arcs
```

Le code suivant calcul les degré de chaque sommet du graphe en utilisant les sommets adjacents (neighbor) présents dans le dictionnaire 'self.gdict[somm] :

```
0   def calculate_degrees(self):
1            degrees = defaultdict(int)
2            for somm in self.gdict:
3                for neighbor in self.gdict[somm]:
4                    degrees[somm] += 1
5
6            return degrees
```

Fonction de calcul de l'ordre de dégénérescence

```
0   def degeneracy_ordering(self):
1            ordering = []
2            degrees = self.calculate_degrees()
3            sorted_vertices = sorted(degrees, key=lambda x: degrees[x], reverse=False)
4
5
6            while len(sorted_vertices) > 0:
7
8                somm = sorted_vertices.pop(0)   # Enlever le premier noeud de la queue(ce
9
10
11               ordering.append(somm)           # Ajouter ce noeud a  ordering
12
13               for u in self.gdict:
14
15                   degrees[u] -= 1
16
17
18           return ordering
19
20
```

Fonction de calcul de la dégénérescence

```
0  def degeneracy(self):
1
2          degrees = self.calculate_degrees()
3          sorted_vertices = sorted(degrees, key=lambda x: degrees[x], reverse=False)
4          k = 0
5
6
7          for vertex in sorted_vertices:
8              if degrees[vertex] > k:
9                  k += 1
10             else:
11                 break
12         print("the degeneracy of the graph is : ",k)
```

Fonction pour trouver les graphes G_j de G

```
0  def find_Gj(self,j):
1         gj= Graphe()
2         list_voisinage = []
3         Vi = []
4         vertex_order = -1
5         # calcul N[vi]
6         list_voisinage.append(self.getSommets()[j])
7         for s in self.gdict[self.getSommets()[j]]:
8          list_voisinage.append(s)
9         #calcul Vi
10        ordre = self.degeneracy_ordering()
11        for i in range(len(self.degeneracy_ordering())):
12         if( ordre[i] == self.getSommets()[j]):
13             vertex_order = i
14         if(vertex_order != -1):
15             Vi.append(ordre[i] )
16
17         #N[vi] inter Vi
18         for v1 in list_voisinage:
19             for v2 in Vi:
20                 if v1 == v2 :
21                     gj.addSommet(v1)
22         #ajout des arcs reliants
23         for som in gj.getSommets():
24             for som_voisin in self.gdict[som]:
25                 if (som_voisin in gj.getSommets()) and
26                 (som_voisin not in gj.gdict[som]):
27                     gj.gdict[som].append(som_voisin)
28
29
30        return gj
```

Fonction de calcul de cliques maximales Dans ce code, on a utilisé l'algorithme de Bron Kerbosh pour trouver les cliques maximale d'un graphe non orienté donné. On a utilisé Bron Kerbosh avec pivot et en utilisant l'ordre de dégénérescence du graphe pour trouver rapidement l'ensemble des cliques maximales [18] [5] [6].

```
0   def find_cliques(graph):
1     p = set(graph.gdict.keys())
2     r = set()
3     x = set()
4     cliques = []
5     for v in graph.degeneracy_ordering():
6       neighs = graph.gdict[v]
7       find_cliques_pivot(graph.gdict, r.union([v]),
8       p.intersection(neighs), x.intersection(neighs), cliques)
9       p.remove(v)
10      x.add(v)
11    return sorted(cliques, key= lambda x: len(x))
12
13
14  def find_cliques_pivot(graph, r, p, x, cliques):
15    if len(p) == 0 and len(x) == 0:
16      cliques.append(r)
17    else:
18      u = iter(p.union(x)).__next__()
19      for v in p.difference(graph[u]):
20        neighs = graph[v]
21        find_cliques_pivot(graph, r.union([v]),
22        p.intersection(neighs), x.intersection(neighs), cliques)
23        p.remove(v)
24        x.add(v)
```

Application de l'algorithme sur un graphe de petite taille

```
0  from Graphe import Graphe
1  from Graphe import sort_items
2  from Graphe import filter_lists
3  from Bron_kerbosch_function import find_cliques
4
5
6  graph_elements = {
7     "1" : ["2","3"],
8     "2" : ["1","3"],
9     "3" : ["1","2","4"],
10    "4" : ["3"]
11 }
12
13 # Instanciation
14 g = Graphe(graph_elements)
15
16 #Initialiser une liste vide T
17 T =[]
18
19 #Implémentation de l'algorithme
20 for j in range(0, len(g.getSommets())):
21    Gj = g.find_Gj(j)
22    cliques = find_cliques(Gj)
23    for k in cliques:
24       k = sort_items(k,g.degeneracy_ordering())
25       for n in k:
26          T.append(k)
27          T= filter_lists(T)
28
29 print("les cliques maximales du graphe sont:")
30 for k in T:
31    print(k)
32
```

3 Conclusion générale

Le problème d'énumération de cliques maximales dans les graphes non orientés est un problème complexe et difficile surtout pour le cas de traitement des graphes de grande taille. Pour résoudre ces problèmes il existe de nombreux algorithmes, plus ou moins adaptés aux grands graphes. Dans notre travail nous avons proposé et évalué une méthode pour résoudre ce problème en utilisant à la fois l'algorithme de Bron Kerbosh et l'ordre de dégénérescence du graphe tout en optimisant la complexité en temps et en espace.

Ce projet nous a donné l'occasion pour développer notre esprit d'équipe et le travail collectif. Pendant ce travail, nous avons rencontré des difficultés telles que l'utilisation des nouvelles techniques, langages, outils de développement ainsi que la contrainte du temps.

Cependant, nous chercherons d'ajouter d'autres tâches telles que l'ajout de quelques mesures de performance du programme, l'ajout des plots dans le code pour mieux visualiser les cliques maximales du graphes,... .

Références

[1] URL : https://math.stackexchange.com/questions/3379012/algorithm-to-check-if-an-equivalent-node-exists-on-a-different-graph-with-differ. (accessed : 26.11.2022).

[2] URL : http://atestat-grafuri.weebly.com/tipuri.html. (accessed : 10.12.2022).

[3] URL : https://github.com/Microsoft/vscode/issues/36877. (accessed : 06.01.2023).

[4] URL : https://ext.boulgour.com/lifl/beaufils/logos/. (accessed : 06.01.2023).

[5] URL : https://github.com/GayathriBalakumar/Community-detection-on-social-media/blob/master/Report%20-%20Detecting%20Tight%20Communities%20and%20Friend%20Recommendation%20on%20Facebook.pdf. (accessed : 06.12.2022).

[6] URL : https://github.com/GayathriBalakumar/Community-detection-on-social-media/blob/master/Source%20Code.py. (accessed : 26.12.2022).

[7] AéroCode. *Théorie des graphes(1)*. URL : https://aerocode.net/272. (accessed : 01.01.2023).

[8] alloprof. *Les graphes*. URL : https://www.alloprof.qc.ca/fr/eleves/bv/mathematiques/graphes-m1413. (accessed : 20.11.2022).

[9] Maxime Bourrigan. *Bourrigan : Géométrie d'incidence*. URL : https://culturemath.ens.fr/thematiques/geometrie/geometrie-d-incidence. (accessed : 20.11.2022).

[10] *Graphes*. URL : https://www.annales2maths.com/terminale-cours-graphes/. (accessed : 01.11.2022).

[11] *Graphes abstraits*. URL : https://www.schoolmouv.fr/cours/graphes-abstraits/fiche-de-cours. (accessed : 20.12.2022).

[12] *Ibis Paint X pour ordinateur*. URL : https://android-emulator.club/ibis-paint-x/. (accessed : 06.01.2023).

[13] Hamida Seba Jocelyn Bernard. *Résolution de problèmes de cliques dans les grands graphes*. URL : https://hal.science/hal-01284640/file/resolution-de-problemes-de-cliques.pdf. (accessed : 16.11.2022).

[14] Godfry Justo. *Hash tables*. URL : https://eng.libretexts.org/Bookshelves/Computer_Science/Programming_and_Computation_Fundamentals/Book%3A_Algorithm_Design_and_Analysis_(Justo)/04%3A_Hash_Tables_Graphs_and_Graph_Algorithms/4.01%3A_Activity_1_-_Hash_Tables. (accessed : 28.12.2022).

[15] *LA THÉORIE DES GRAPHES*. URL : https://www.methodemaths.fr/theorie_des_graphes/. (accessed : 17.12.2022).

[16] *Logos Download*. URL : https://logos-download.com/9988-python-logo-download.html. (accessed : 06.01.2023).

[17] George Manoussakis. "A new decomposition technique for maximal clique enumeration for sparse graphs". In : *ELSEVIER* 770.10 (24 May 2019), p. 25-33. DOI : https://doi.org/10.1016/j.tcs.2018.10.014.

[18] *Maximal cliques*. URL : https://github.com/SudharakaP/MaximalCliques/tree/master/src. (accessed : 01.12.2022).

[19] OpenClassrooms. *Stockez et retrouvez des données grâce aux tables de hachage.* URL : https://openclassrooms.com/fr/courses/19980-apprenez-a-programmer-en-c/19978-stockez-et-retrouvez-des-donnees-grace-aux-tables-de-hachage. (accessed : 04.01.2023).

[20] Oracle. *Qu'est-ce qu'une machine virtuelle (VM) ?* URL : https://www.oracle.com/fr/cloud/definition-machine-virtuelle-vm/. (accessed : 06.01.2023).

[21] *Qu'est-ce que la théorie des graphes ?* URL : https://www.jedha.co/blog/theorie-des-graphes#:~:text=La%20th%C3%A9orie%20des%20graphes%20est%20une%20discipline%20math%C3%A9matique,de%20travailler%20sur%20les%20relations%20entre%20les%20donn%C3%A9es.. (accessed : 31.12.2022).

[22] Martin Schindler. *Schindler : Linux 4.16 behebt weitere Probleme bei Spectre und Meltdown.* URL : https://www.silicon.de/41667667/linux-4-16-behebt-weitere-probleme-bei-spectre-und-meltdown. (accessed : 01.01.2023).

[23] *Stack Overflow : Identifying cliques in R.* URL : https://stackoverflow.com/questions/26222659/identifying-cliques-in-r. (accessed : 01.01.2023).

[24] Wikipédia. *Algorithme de Bron-Kerbosch.* URL : https://fr.wikipedia.org/wiki/Algorithme_de_Bron-Kerbosch. (accessed : 20.11.2022).

[25] Wikipédia. *Clique (théorie des graphes).* URL : https://fr.wikipedia.org/wiki/Clique_(th%C3%A9orie_des_graphes). (accessed : 03.01.2023).

[26] Wikipédia. *Linux.* URL : https://fr.wikipedia.org/wiki/Linux. (accessed : 06.01.2023).