Meryem BELASSEL
Thomas NICOLLE
Groupe c

Rapport du projet

Introduction:

L'algorithme de Bron-Kerbosch représente un outil efficace pour identifier de manière exhaustive les cliques maximales dans un graphe.

L'objectif de ce projet est de mettre en place une version modifiée de l'algorithme de Bron-Kerbosch afin de repérer la plus grande k-quasi-clique dans un graphe spécifique. Une k-quasi-clique est un ensemble de sommets formant une clique après un minimum de k arêtes supprimées.

Afin de mener à bien ce projet, nous avons abordé plusieurs aspects. Dans un premier temps, nous avons élaboré les fonctions indispensables pour générer un graphe et consigner les éléments requis pour mettre en œuvre l'algorithme de Bron-Kerbosch. Par la suite, nous avons ajusté cette méthode afin de déterminer les k-quasi-cliques.

Principe de l'algorithme de Bron-Kerbosch :

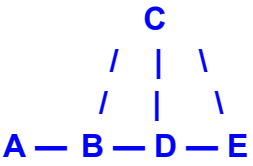
L'algorithme de Bron-Kerbosch consiste à énumérer toutes les cliques maximales de graphe non orienté. Autrement dit, il recense tous les sous-ensembles de sommets où chaque paire de sommets est reliée par un lien (c'est une clique), et aucun des ensembles de sommets recensés ne peut avoir de sommets supplémentaires ajoutés tout en conservant sa connectivité totale.
L'algorithme de Bron-Kerbosch, dans sa forme basique, est un algorithme de retour sur trace récursif qui cherche toutes les cliques maximales dans un graphe G. Plus précisément, avec trois ensembles de sommets disjoints R, P et X, il trouve les cliques maximales qui comprennent tous les sommets de R, certains des sommets de P et aucun des sommets de X.
Voici les étapes de l'algorithme en pseudo-code:

BronKerbosch(R, P, X) est
si P et X sont tous les deux vides alors
rapporter R comme une clique maximale
pour chaque sommet v dans P faire
BronKerbosch1(R ∪ {v}, P ∩ N(v), X ∩ N(v))
P := P \\ {v}

$X := X \cup \{v\}$

Où:
R est un sous-ensemble des sommets de la potentielle clique.
P est l'ensemble des sommets candidats pour être ajoutés à la potentielle clique.
X contient des sommets déjà traités ou appartenant déjà à une clique maximale.
Lorsque P et X sont tous les deux vides, il n'y a pas d'autres éléments qui peuvent être ajoutés à R, donc R est une clique maximale et l'algorithme retourne R.

Considérons le graphe suivant avec 5 sommets (A, B, C, D, E) et 6 arêtes (AB, BC, BD, CE, CD, ED) comme suit :



Appliquons maintenant l'algorithme de Bron-Kerbosch sur ce graphe. Tout d'abord, nous commençons par initialiser R, P et X. R et X seront vides, tandis que P regroupera tous les sommets potentiels (A, B, C, D, E). Par la suite, nous nous retrouvons dans le cycle de l'algorithme. Un sommet est sélectionné dans P, et nous appelons l'algorithme récursivement avec les nouveaux ensembles R, P et X. Nous allons d'abord traiter le sommet A par exemple :

BronKerbosch(R ∪ {A}, P ∩ N(A), X ∩ N(A))

Ce qui nous donne comme éléments de nos ensembles :

- R = {A}
- P = {B, C}
- X = {}

P représente les voisins du sommet traité. Le traitement des éléments par l'algorithme se poursuit jusqu'à ce que P et X soient tous les deux vides, ce qui signifie que nous avons atteint une clique maximum. Dans cet exemple, l'algorithme trouvera trois cliques maximales: {A, B, C}, {B, D, E} et {C, E}.

La fonction Bron-Kerbosch :

Pour coder ce projet, nous avons choisi d'utiliser le langage C++. Nous avons réalisé plusieurs fonctions pour nous aider à simplifier la tâche, tels que :

- **genererGraphe(int p) :** cette fonction permet de générer un graphe aléatoire avec une probabilité *p* pour chaque couple de sommets d'être lié par une arête. Le graphe est représenté par une matrice d'adjacence G. Une telle matrice est une représentation matricielle d'un graphe fini. Elle est utilisée pour représenter les arêtes entre les sommets d'un graphe. Les lignes et colonnes de cette matrice d'adjacence correspondent aux sommets du

graphe. Dans le cas où il y a une arête entre 2 sommets, alors la cellule à l'intersection de la ligne et de la colonne contient 1, sinon elle contient 0.

- **remplirVoisins() :** elle remplit le tableau initialisé auparavant *Voisins*, qui est de type *vector<vector<int>>*, une structure de données, vecteur de vecteurs. Dans ce cas, *Voisins* est un vecteur dont chaque élément est lui-même un vecteur d'entiers. Chaque élément de cette structure est un sommet du graphe. Pour un sommet donné, son vecteur correspondant dans *Voisins* contient la liste de ses voisins, ceux qui y sont directement liés par une arête.
- **afficherVoisins() :** Pour vérifier si le graphe a été bien généré, nous avons implémenté cette fonction pour afficher les voisins de chaque sommet du graphe.
- **intersection(const vector<int>& ensemble1, const vector<int>& ensemble2) :** cette fonction permet de faire l'intersection entre deux ensembles. Pour ce faire, elle crée **set1** de type *set* à partir d'ensemble1, puis parcourt ensemble2 et ajoute chaque élément qui se trouve également dans *set1* au *resultat* (qui est de type la structure vecteurr d'entiers).
- **difference(const vector<int>& ensemble1, const vector<int>& ensemble2) :** cette méthode permet de calculer la différence de deux ensembles, ça veut dire qu'elle récupère les éléments qui sont dans *ensemble1* mais pas dans *ensemble2*. Elle crée un *set* à partir de *ensemble2,* puis parcours *ensemble1* et ajoute chaque élément qui ne se trouve pas dans *set2* au resultat.
- **unionEnsembles(const vector<int>& ensemble1, const vector<int>& ensemble2) :** cette fonction réalise l'union entre deux ensembles, et donc les éléments se trouvant dans *ensemble1* ou *ensemble2* ou les deux. Tout d'abord, elle copie *ensemble1* dans *resultat,* puis elle ajoute tous les éléments de *ensemble2* à *resultat*. Ensuite, elle effectue le tri de *resultat* et utilise la fonction **unique**. La fonction **unique** permet de supprimer les doublons en déplaçant les doublons vers la fin du vecteur et l'itérateur pointe vers le premier doublon qui est par la suite utilisé par *erase* pour effacer tous les doublons.
- **BronKerbosch(vector<int> R, vector<int> P, vector<int> X) :** cette fonction repose sur une approche de backtracking pour naviguer toutes les combinaisons possibles de sommets qui forment des cliques. Expliquons le fonctionnement de cette dernière :

➔ **Les ensembles de sommets utilisés dans ce programme**:

- **R** : L'ensemble des sommets déjà choisis pour former une clique partielle.
- **P** : L'ensemble des sommets candidats qui peuvent être ajoutés à la clique partielle actuelle. Ce sont les voisins des sommets déjà sélectionnés dans **R**.
- **X** : L'ensemble des sommets déjà retirés de la clique partielle. Ce sont les sommets qui ne peuvent pas être ajoutés car ils sont déjà voisins des sommets dans **R**.

- ➔ **Cas de base** : L'algorithme s'arrête lorsque **P** et **X** sont vides, ce qui signifie qu'aucun sommet n'est plus candidat pour être ajouté à la clique courante.
- ➔ **Partie récursive** : Pour chaque sommet **v** dans **P**, l'algorithme essaie de l'ajouter à la clique partielle en formant un nouvel ensemble **R_ajout**. Ensuite, il met à jour les ensembles **P** et **X** en fonction des voisins de **v**.
    - ◆ **P_voisins** : Les sommets de **P** qui sont voisins de **v**. Ce sont les candidats potentiels pour être ajoutés à la clique partielle après l'ajout de **v**.
    - ◆ **X_voisins** : Les sommets de **X** qui sont voisins de **v**. Ces sommets sont exclus de la clique partielle car ils ont déjà été inclus ou sont adjacents à des sommets dans la clique.
- ➔ **Backtracking** : L'algorithme récursif continue d'appeler la fonction BronKerbosch avec les ensembles mis à jour jusqu'à ce que tous les sommets soient explorés. À chaque étape, il explore toutes les possibilités en ajoutant un sommet à la fois à la clique partielle.
- ➔ **Validation et stockage des cliques maximales** : Lorsqu'une clique maximale est identifiée et qu'il y a plus de sommets candidats pour former une plus grande clique, elle est stockée dans un ensemble de cliques trouvées (le *set* créé auparavant) pour éviter les doublons. Ensuite, les cliques maximales sont affichées dans un fichier" *cliques.txt*", placé dans le dossier *fichiers*.