Université of Science et Technologies HOUARI BOUMEDIENE
Département d'informatique

TP

COMPELXITE

« PROJET COMPLEXITE »

« PROBLEME DE CLIQUE »

NOM	PRENOM	MATRICULLE	GROUPE
BELIMI	Ibrahim Sabri	161631074255	A2 TD3 TP4
ARAB	MAHER	171731045353	A2 TD3 TP4
ZAIT	Fouad	181831072145	A2 TD3 TP4
Khellaf	Abdelhamid	161631061928	A2 TD3 TP4

Dans Ce travail, on est demande de faire l'étude de 4 Algorithmes

L'étude se compose de :

1) Etude théorique du problème :
    A. Historique et présentation du problème.
    B. Définition formelle du problème.
    C. Présenter la modélisation de la solution (Structure de données de la solution).
    D. Présenter l'algorithme de résolution + Complexité théorique.
    E. Présenter l'algorithme de vérification avec pseudo-code et calcul détaillé de sa complexité théorique.
    F. Présenter une instance du problème avec sa solution (un exemple).

2) Etude Expérimentale : (Variation de la taille du problème)
    A. Simuler la complexité temporelle et spatiale **théorique** de l'algorithme de résolution.
    B. Simuler la complexité temporelle et spatiale **théorique** de l'algorithme de vérification.
    C. Calcul de la complexité temporelle et spatiale **expérimentale** de l'algorithme de vérification.
    D. Analyse des résultats.
    E. Conclusion
    F. Références
    G. Annexe : code source

(1)
Etude
Théorique

1)-A Historique et présentation du problème:

Présentation

En informatique, le problème de la clique est un problème qui consiste à trouver des cliques (sous-ensembles de sommets d'un graphe tous adjacents les uns aux autres, également appelés sous-graphes complets) dans un graphe. Un graphe complet est un graphe dont lequel chaque sommet est lié avec tous les autres (n-1) sommets (la liaison veut dire que les 2 sommets sont compatibles, et la définition de la compatibilité change d'un problème à un autre). Ce problème a plusieurs formulations différentes selon les cliques et les informations sur les cliques devant être trouvées. Les formulations courantes du problème de la clique incluent la recherche d'une clique maximum (une clique avec le plus grand nombre possible de sommets), la recherche d'une clique de poids maximal dans un graphe pondéré, la liste de toutes les cliques maximums et la résolution du problème de décision consistant à déterminer si un graphe contient une k-clique (k étant un entier donné). Nous ce qui nous intéresse c'est bien le problème de décision (si un graphe contient une k-clique ou non).

Historique

L'étude de sous-graphes complets en mathématiques est bien plus antérieure à la terminologie « clique ». Par exemple, les sous-graphes complets font une première apparition dans la littérature mathématique par Erdős et Szekeres (1935). Mais le terme « clique » et le problème de lister les cliques de manière algorithmique proviennent tous deux des sciences sociales, où des sous-graphes complets sont utilisés pour modéliser des cliques sociales (des groupes de personnes qui se connaissent toutes). En 1949, Luce et Perry ont utilisé des graphes pour modéliser les réseaux sociaux et ont adapté la terminologie des sciences sociales à la théorie des graphes. Ils ont été les premiers à appeler les sous-graphes complets « cliques ». Le premier algorithme pour résoudre le problème de la clique est celui de Harary et Ross (1957), qui étaient motivés par les applications sociologiques toujours.
Depuis les travaux de Harary et Ross, de nombreux autres ont conçu des algorithmes pour différentes versions du problème de la clique. En

1977, Tarjan et Trojanowski ont publié un premier travail sur la complexité du pire cas du problème de la clique maximum. Toujours dans les années 1970, Cook et Karp ont commencé à utiliser la théorie de la NP-complétude et notamment des résultats d'insolvabilité pour fournir une explication mathématique de la difficulté du problème de clique. Et en 1991, une série d'articles commence à se publiée dans le New York Times. Il a été montré que (en supposant $P \neq NP$), il n'est même pas possible d'approcher le problème avec précision et efficacité.
Les algorithmes de résolution du problème de clique ont été utilisés dans plusieurs domaines. Comme en chimie, en bio-informatique, en mathématiques …

1)-B Définition formelle du problème

Définitions :

1- Graphe :

Un graphe est une représentation graphique d'un ensemble d'objets où certaines paires d'objets sont reliées par des liens. Les objets interconnectés sont représentés par des points appelés sommets, et les liens qui relient les sommets sont appelés arêtes.
Formellement, un graphe est une paire d'ensembles (V, E), où V est l'ensemble des sommets et E est l'ensemble des arêtes, reliant les paires de sommets.

2- Graphe complète :

Un graphe avec exactement une arête entre chaque paire de sommets distincts est appelée graphe complet. Un graphe complet à n sommets est généralement noté K_n.

3- Clique :

Une clique est un sous-ensemble de sommets d'un graphe non orienté G tel que tous les deux sommets distincts de la clique sont adjacents ; c'est-à-dire que son sous-graphe induit est complet. Les cliques sont l'un des concepts de base de la théorie des graphes et sont utilisées dans de nombreux autres problèmes mathématiques et constructions sur les graphes.
La tâche de trouver s'il y a une clique d'une taille donnée dans un graphe (le problème de clique) est NP-complet, mais malgré ce résultat de dureté, de nombreux algorithmes pour trouver des cliques ont été étudiés.

4- NP-complète :

Problème NP-complet, n'importe lequel d'une classe de problèmes de calcul pour lesquels aucun algorithme de solution efficace n'a été trouvé. De nombreux problèmes informatiques importants appartiennent à cette classe, par exemple le problème du voyageur de commerce, les problèmes de satisfiabilité et les problèmes de couverture de graphe.

5- Problème de k-clique :

Dans l'analyse des réseaux sociaux, une k-clique est une clique détendue, c'est-à-dire qu'une k-clique est un sous-graphe quasi-complet. Une k-clique dans un graphe est une clique de taille k, tel qu'entre chaque 2 somme de clique il y'a une arête.
Les graphes du monde réel, tels que les réseaux sociaux, les réseaux biologiques et les réseaux de communication, sont souvent constitués de structures de sous-graphes cohésives. L'extraction de sous-graphes cohésifs à partir d'un graphe est un problème fondamental en analyse de réseau qui a beaucoup attiré l'attention des communautés de bases de données et d'exploration de données [2, 3, 4, 5, 6, 7]. La sous-structure cohésive la plus élémentaire d'un graphe est peut-être la structure k-clique qui a été largement utilisée dans diverses applications d'analyse de réseau.
La détection de toutes les k-cliques dans un graphe est un problème fondamental d'exploration de graphe. Malheureusement, le problème de la liste des k-cliques est souvent considéré comme infaisable pour un grand k, car le nombre de k-cliques dans un graphe est exponentiel par rapport à la taille k. Les solutions de pointe pour le problème sont basées sur l'heuristique d'ordre sur les nœuds qui peut efficacement lister toutes les k-cliques dans de grands graphes du monde réel pour un petit k (par exemple, $k \leq 10$).
Malgré qu'une variété d'algorithmes heuristiques ont été proposés, il manque encore une comparaison approfondie pour couvrir tous les algorithmes de pointe et évaluer leurs performances à l'aide de divers graphiques du monde réel. Cela rend difficile pour un praticien de sélectionner l'algorithme à utiliser pour une application spécifique.
Les algorithmes de détection de k-clique peut être classifier en deux catégories :

- Les algorithmes exacts.
- Les algorithmes d'approximation.

Dans ce travaille en introduise l'algorithme de Chiba-Nishizeki[1] note Arbo qui est un algorithme un algorithme exact non ordonner.

Algorithme de résolution :

Dans ce travail on a introduit l'algorithme de chiba-Nishizeki [1], cet algorithme trie d'abord les sommets dans un ordre de degré décroissant (ligne 7 de l'algorithme 1). Ensuite, l'algorithme traite les sommets suivants dans cet ordre (ligne 8 de l'algorithme 1).

L'étape suivant permet de crée un sous-graphe G_v induit par les voisins de v puis exécute récursivement la même procédure sur un tel sous-graphe induit (lignes 9-10 de l'algorithme 1).

Apres la gestion d'un sous-graphe induit G_v, l'algorithme doit réorganiser les sommets dans G_v en fonction de leurs degrés tel qu'après avoir traité un sommet v, l'algorithme supprime v du graphe actuel pour éviter qu'une k-clique contenant v ne soit répertoriée à plusieurs reprises (ligne 11 de l'algorithme 1).

L'algorithme suivant montre le pseudocode de l'algorithme de Chiba-Nishizeki. La particularité de cet algorithme est que sa complexité temporelle est étroitement liée à l'arboricité du graphe. Plus précisément, l'algorithme répertorie toutes les k-cliques dans le temps $O(km\alpha^{k-2})$.

Note que :

K : nombre de sommet de clique.

M : le nombre d'arêtes.

α : l'arboricité de graphe G.

- Complexité spatiale :

Elle est O(n+m) tel que :

M : le nombre d'arêtes.

N : le nombre de sommets.

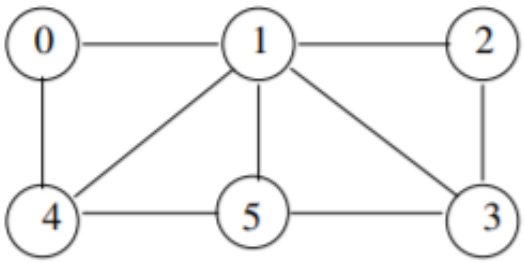
Procédure Arbo ;
Entrée : G : graphe, k : entier ;
Sortie : tous les k-cliques ;

```
Début
    Si k=2
        Alors
            Pour chaque arêtes (u, v) dans G
            Faire
                Return k-clique R union {(u, v)};
            Fait;
        Sinon
            Triée les sommets de G tel que dv1(G)>=.... >=dv|VG|(G) ;
            Pour i de 1 a |VG|
            Faire
                Crée un sous graphe Gvi de G induit par l'ensemble des voisins de vi ;
                Arbo (Gvi , R union {vi}, l-1) ;
                Supprimer vi de G ;
            Fait ;
    Fsi;
Fin;
```

1)-C Modélisation de la solution (Structure de données de la solution):

Nous avons modélisé le graphe non orienté G=(S,A) par une matrice d'adjacence M[n,n] où les indices i,j représentent les sommets du graphe . Cette matrice contiendra des 0 et des 1 M[i,j] =1 si i et j sont relié entre eux (il existe une arrête entre les sommets i et j) et M[i,j] = 0 si i et j ne sont pas relié entre eux (il n'existe pas une arrête entre les sommets i et j).

Considérons par exemple le graphe suivant :



La matrice d'adjacence de ce graphe est:

	0	1	2	3	4	5
0	0	1	0	0	1	0
1	1	0	1	1	1	1
2	0	1	0	1	0	0
3	0	1	1	0	0	1
4	1	1	0	0	0	1
5	0	1	0	1	1	0

On remarque qu'il est possible de ne mémoriser que la composante triangulaire supérieure de la matrice d'adjacence car la matrice est symétrique mais en divisant ainsi la taille mémoire par deux nous compliquons légèrement les traitements car il sera alors nécessaire de faire un test avant d'accéder à M[i][j].

On utilise un algorithme de recherche exhaustive (méthode qui consiste à vérifier toutes les solutions possible) Cet algorithme examine chaque sous graphe avec k-sommets et vérifie s'il forme une clique.

-engendrer une solution quelconque s= (s_1, s_2, s_3…, s_k) qui peut être correcte ou mauvaise. La structure de données adopté pour le stockage des sommets est un vecteur nommé s, s [0] =s_1, s[2] =s_2,s[3]=s_3 …..s[k]=s_k.

-Si s est une k-clique alors tous les sommets du sous graphe sont reliés entre eux, ce qui veut dire que :

s[1] est relié aux sommets s[2],s[3],...s[k]

s[2] est relié aux sommets s[3],s[4],...s[k]

s[k-1] est relié au sommet s[k]

On va tester pour un nombre k donné en entrée toutes les combinaisons de sommets possibles qui peuvent éventuellement former une clique de k sommets r.

1)-D Présentation de l'algorithme de vérification avec pseudo code et calcul détaillée de sa complexité théorique :

But de l'algorithme : lister pour un graphe (matrice générée par Random rempli de 1 si indice i est relié à j sinon rempli par 0) tous les k-cliques pour k donnée en entrée et les compter nous avons procéder ainsi :

1) Donner n (taille de la matrice n*n).
2) remplissage d'un tableau qui va contenir tous les sommets (on en aura besoin dans un fonction qui traitera les combinaisons entre sommets pour générer le vecteur s de sommets pour qu'on puisse tester s'il forme une k-clique).
3) remplissage automatique de la matrice par des 0 et des 1 (matrice symétrique).
4) affichage de la matrice d'adjacence qui représente le graphe.
5) Donner un nombre k entier (nombre auquel on veut savoir s'il existe une k clique et les lister).
6) Appeler un procédure Affichage_Cliques on lui donnera en entrée n (taille de la matrice) , k , la matrice et le vecteur de sommets , cette procédure nous permettra de connaitre le nombre de clique de taille k trouvé grâce à un appel à la procédure combinationUtil (qui est récursive) elle essayera toutes le combinaisons possibles de k et appelera la fonction verif_clique (retourne 1 si k est une clique sinon 0) pour verifier si c'est une clique pour afficher la k-clique et incrementer le compteur de k-cliques à la fin la procedure Affichage_Cliques va nous afficher le nombre de k-cliques compter dand la procédure précédente .

```
Algorithme Prog_Principale (){
*matrice: matrice d'entiers;
*Sommets: vecteurs d'entiers (vecteur qui contient les sommets)
i,j,k,nbr_comb: entiers;
Debut
Faire
        Ecrire("donner les dimensions de la matrice n*n");
        Ecrire("donner n");
        Lire(n);
Jusqu'à (n>0) ;
Allouer (matrice);//allouer de l'espace pour la matrice//
Allouer(sommets);//allouer de l'espace pour le vecteur de sommets//
//remplissage du vecteurs de sommets //
Pour i<--1 à n
  Faire
        Sommets[i]=i;
   Fait
//remplissage de la matrice d'adjacence//
Pour i<--1 à n
        Pour j<--1 à n
          Faire
                matrice[i,j]=random () mod 2 ;//remplir la matrice avec des valeurs aléatoires 1
                ou 0
           Fait
Pour i<--1 à n
        Pour j<--1 à n
```

Faire
Si (matrice[i,j]==1) alors
matrice[j,i]=1;// si matrice [i,j]==1 donc i et j sont reliés on doit donc remplir matrice[j,i] avec 1 pour que ca soit un graphe cohérent (matrice symétrique)
Fait
//affichage de la matrice//
Pour i<--1 à n
Pour j<--1 à n
Faire
Ecrire(matrice[i,j]);
Fait

Faire
Ecrire("donner un nombre k pour verifier s'il existe une k-clique ");
Lire(k);
Jusqu'à (k<=n);
Ecrire("Les cliques de taille k trouvé dans le graphe donné sont :");
//debut de calcul du temps d'execution //
Affichage_Cliques(sommets,matrice, n, k);//appeler la procedure Affichage_Cliques//
//fin du calcul du temps d'execution//
Fin;

}

```
//**********************PROGRAMME PRINCIPALE*****************************************//
using namespace std;
int main()
{
    int* matrice,*sommets;
    int *s;
    int i,j, n,k,nbr_comb;
    srand(time(NULL));
//**********************TAILLE DE GRAPH + DECLARATION DE MATRICE D'ADJACENCE**************************
    do
    {
        printf("Quelles sont les dimensions de la matrice n*n? (n > 0)\n");
        printf("Donner n \n");
        scanf("%d", &n);
    }while(n <= 0);
    matrice = (int*)malloc(n*n*sizeof(int));
    sommets = (int*)malloc(n * sizeof(int));
//*************************************************************************************************
//**********************REMPLISSAGE DE TABLEAU CONTENANT TOUS LES SOMMETS DE GRAPH*********************
    for(i=0;i<n;i++)
    {
        sommets[i] = i;
    }
//*************************************************************************************************
//**********************REMPLISSAGE DE MATRICE D'ADJACENCE QUI REPRESENTE LE GRAPH*********************
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            matrice[i*n + j]=rand()%2;
        }
    }

    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            if(matrice[i*n + j]==1)
            {
                matrice[j*n + i]=1;
            }}}
```

```
//**********************AFFICHAGE DE MATRICE D'ADJACENCE QUI REPRESENTE LE GRAPH*****************

    printf("\n La matrice d'adjacence choisi par random est : \n");
    for(i=0;i<n;i++)
    {
        for(j=0;j<n;j++)
        {
            printf("%d  ", matrice[i*n + j]);
        }
        printf("\n");
    }
//*************************************************************************************************

//**********************OBTENIR 'k' LE NOMBRE DE SOMMETS VOULU DE CLIQUE************************
    do
    {
        printf("Donner le nombre k pour verifier si il existe une k-clique (k < %d) \n",n);
        scanf("%d", &k);
    }while(k>n);
//*************************************************************************************************

//////////////////////CACUL TEMPS D'EXECUTION/////////////////////////////////////////////
    printf("\n Les cliques de taille %d trouve dans le graph donne sont:\n");
    auto start = chrono::steady_clock::now();
    Affichage_Cliques(sommets,matrice, n, k);
    auto end = chrono::steady_clock::now();
//////////////////////CACUL TEMPS D'EXECUTION/////////////////////////////////////////////
    system("pause");

//////////////////////TEMPS D'EXECUTION/////////////////////////////////////////////////////
    cout << "Temps D'execution en Nanosecondes: "
    << chrono::duration_cast<chrono::nanoseconds>(end - start).count()
    << " ns"
    << endl;
//////////////////////TEMPS D'EXECUTION/////////////////////////////////////////////////////
    system("pause");
    return 0;
```

Procédure Affichage_Cliques(){
Entrée:
Sommets[] :vecteur d'entiers contenant les sommets , matrice [][]:matrice d'adjacence , n : entier(taille de la matrice), k:entier ;
Sortie: affichage du nombre de k-clique aprés l'appel à combinationUtil.
data[k]: tableau d'entiers;
Cpt:entier;
Cpt=0;
combinationUtil(arr,matrice, n, k, 0, data, 0,cpt);//essayer toutes les combinaisons si une combinaison choisie est une clique on l'affiche et incremente cpt//
Ecrire("Nombre total de clique de taille k trouve dans le graph donne cpt");
}

```
void Affichage_Cliques(int arr[],int*matrice, int n, int k)
{
    int data[k];
    int* cpt = (int*) malloc(sizeof(int));
    *cpt = 0;
    combinationUtil(arr,matrice, n, k, 0, data, 0,cpt);
    printf("\n Nombre total de clique de taille %d trouve dans le graph donne est: %d\n",k,*cpt);
}
//*************************************************************************************************
```

Procedure combinationUtil(){
Entrée:
Sommets[] :vecteur d'entiers contenant les sommets , matrice [][]:matrice d'adjacence , n : entier(taille de la matrice), k:entier ,index:entier,data[]:vecteur d'entiers,i:entier,cpt :entier.

\

Sortie: afficher les k-cliques après verification des toutes les combinaisons.
Debut
Faire
  Si (index==k) alors
    Faire
Si(verif_clique(matrice,data,k,n) == 1) alors
Faire
Ecrire("La combinaison : ");
Pour j<--1 à k
                    Faire
                              Ecrire(data[j]);//afficher la clique
                    Fait
          Ecrire(“===> Une clique”);
       cpt=cpt+1;
     Fait
  Fsi
Jusqu'à(i >= n);
Data[index]=arr[i];
combinationUtil (arr,matrice, n, k, index+1, data, i+1,cpt);//appel recursive a combinaisonUtil avec paramètres index+1 et i+1//
combinationUtil(arr, matrice,n, k, index, data, i+1,cpt); //appel recursive a combinaisonUtil avec paramètres index+1 et i+1//
Fin;

```
void combinationUtil(int arr[], int*matrice, int n, int k, int index, int data[], int i, int*cpt)
{

    if (index == k)
    {
        if (verif_clique(matrice,data,k,n) == 1)
        {
            printf("La combinaison : ");
            for (int j=0; j<k; j++)
            {
                printf("%d ",data[j]);
            }
            printf("===> Une clique\n");
            *cpt = *cpt+1;
        }

        return;
    }

    if (i >= n)
        return;

    data[index] = arr[i];
    combinationUtil(arr,matrice, n, k, index+1, data, i+1,cpt);
    combinationUtil(arr, matrice,n, k, index, data, i+1,cpt);
}
```

Fonction verif_clique(){
Entrée: Sommets[] :vecteur d'entiers contenant les sommets , matrice [][]:matrice d'adjacence ,
n : entier(taille de la matrice), k:entier.
Sortie: clique :booleen retourne vrai si k est une clique sinon 0.
i,j:entiers.clique :booleen.
Debut
i,j:entiers.
Clique =vrai;
i=1;
Tantque(i<=k-1)et (clique)
Faire
Debut
j=i+1;
Tantque(j<=k) et clique faire
si (M[s[i],s[j]==0) alors clique =faux;
sinon j=j+1;
i=i+1;
Fin;
Si clique alors retourner vrai
Sinon retourner faux;
Fin;

```c
int verif_clique(int*matrice,int*s,int k,int n)
{
    _Bool clique=1;
    int i=0,j;
    while(i<k-1 && clique==1)
    {
        j=i+1;
        while(j<k && clique==1)
        {
            if(matrice[s[i]*n+s[j]]==0)
            {
                clique=0;
            }
            else
            {
                j=j+1;
            }
        }
        i=i+1;
    }
    if(clique==1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
```

Calcul de la complexité théorique temporelle :

Dans notre partie du projet on s'intéresse à la partie du problème Np-complet qui est trouvé toutes les k-clique dans un graphe.

Dans notre algorithme la fonction verifclique qui vérifie si pour un vecteur de sommets de taille k donnée ses sommets forment une k-clique cette fonction s'exécute en $O(k^2)$ dans le pire cas car dans le pire cas k est une clique(on parcours tout le tableau de taille k contenant les sommets la 1ere boucle va de l'indice 1 jusqu'à k-1 donc on parcours k-1 éléments dans le pire cas et la 2eme boucle va de l'indice i+1 jusqu'à k donc on parcours k-1 éléments dans le pire cas donc on aura en $(k-1)*(k-1) = k^2 - 2k+1$ donc on note $O(k^2)$ la complexité étant polynomiale le problème de la clique appartient à NP .Cette fonction s'exécute pour toutes les combinaisons possibles donc pour C(n,k) fois mais cette fonction est récursive (passe par deux phases la phase de descente et la phase de remontée. Dans la phase de descente, chaque appel récursif fait à son tour un appel récursif. En arrivant à la condition terminale , on commence la phase de remontée qui se poursuit jusqu'à ce que l'appel initial soit terminé, ce qui termine le processus récursif) chaque appel à la fonction engendre 2 appels récursives pour la fonction se produit n fois jusqu' à ce qu'à la condition d'arrêt on exécute C(n,k) au plus 2^n fois, dans le pire k k serait égale à n/2 donc k a besoin d'au moins n/2 étapes pour atteindre n donc le nombre d'appels double au moins 2n/2 fois Mais il y a beaucoup plus d'appels. La complexité en temps de cette fonction est $O(2^n)$ .lorsqu'on la multiplie par $O(k^2)$ car pour chaque combinaison on appelle la fonction verifclique la complexité devient $O(k^2 * 2^n)$ la complexité de la procédure affichecliques est égale à $O(k^2 * 2^n)$ en ignorant les déclarations de variables la procédure appelle uniquement la procédure combinationUtil

Complexité spatiale :

La complexité spatiale de notre fonction verifclique est O (1) car on ne crée pas d'objets supplémentaires, la procédure CombinaisonUtil prendra C(n,k)*k d'espace mémoire car pour chaque combinaisons on va créer un tableau de taille k dans lequel on va stocker les combinaisons de sommets qui peuvent ou pas former une k-clique donc la complexité spatiale de notre algorithme sera O(k*C(n,k)) ou C(n,k) =n!/k!(n-k)! Dans le pire cas on cherchera une clique de n/2 éléments donc k=n/2.

1)-F Présenter une instance du problème avec sa solution (un exemple)

Les donnes de L'algorithme sont :

ENTREES :

N : entier ; « taille de graph définie par l'utilisateur »

k : entier ; « nombre de sommet voulu dans une clique ''taille de clique'' »

SORTIE :

Affichage de cliques de taille k trouve:

On prend un exemple

N = 5 ; ***……taille de graph***

K = 3 ***; nombre de sommet voulu dans une clique***

Puisque N= 5, donc l'algorithme va générer **(automatiquement au hasard)** une matrice d'adjacence de taille (5 * 50 qui représente un graph de taille 5) **(5 sommets)**

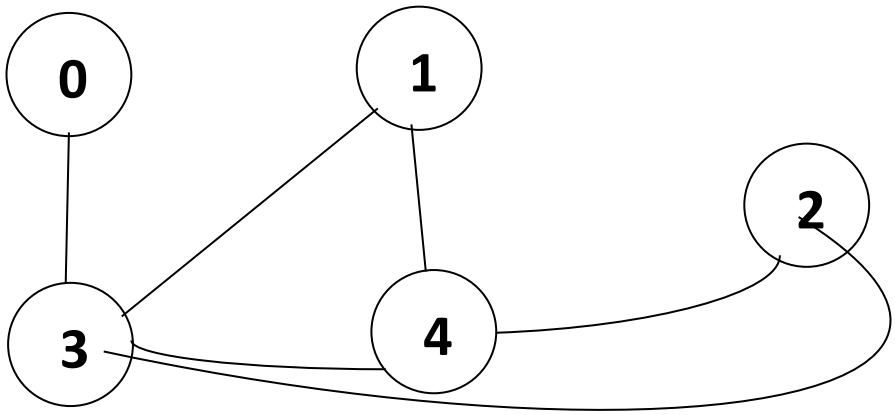
Ceci est la matrice d'adjacence généré par la Random ;

```
Quelles sont les dimensions de la matrice n*n? (n > 0)
Donner n
5

 La matrice d'adjacence choisi par random est :
0  0  0  1  0
0  0  0  1  1
0  0  0  1  1
1  1  1  1  1
0  1  1  1  1
```

Et la valeur de K ;

```
Donner le nombre k pour verifier si il existe une k-clique (k < 5)
3
```

Et donc le graph déduit à partir de la matrice d'adjacence est ;



Selon ce graph toutes les combinaisons possibles de 3 sommets sont :

$$5 \, C \, 3 = 10$$

Qui sont :

0 1 2 0 1 3 0 1 4 0 2 3 0 2 4 0 3 4
1 2 3 1 2 4 1 3 4
2 3 4

Et les clique de taille 3 sont ; 1 3 4 et 2 3 4

L'algorithme va tester chaque combinaison de 3 sommet et il vérifie si elle compose une clique ;

Puis il affiche les cliques de taille (k=3) trouver, nombre de clique et temps d'exécution

```
Donner le nombre k pour verifier si il existe une k-clique (k < 5)
3

 Les cliques de taille 3 trouve dans le graph donne sont:
La combinaison : 1 3 4 ===> Une clique
La combinaison : 2 3 4 ===> Une clique

 Nombre total de clique de taille 3 trouve dans le graph donne est: 2
```

```
Press any key to continue . . .
Temps D'execution en Nanosecondes: 255600 ns
Press any key to continue . . .
```

(2)
Etude Expérimentale

2)-B simuler la complexité temporelle et spatiale théorique de l'algorithme de resolution :

Dans un graphe complet de 500 sommets avec 124750 arêtes :

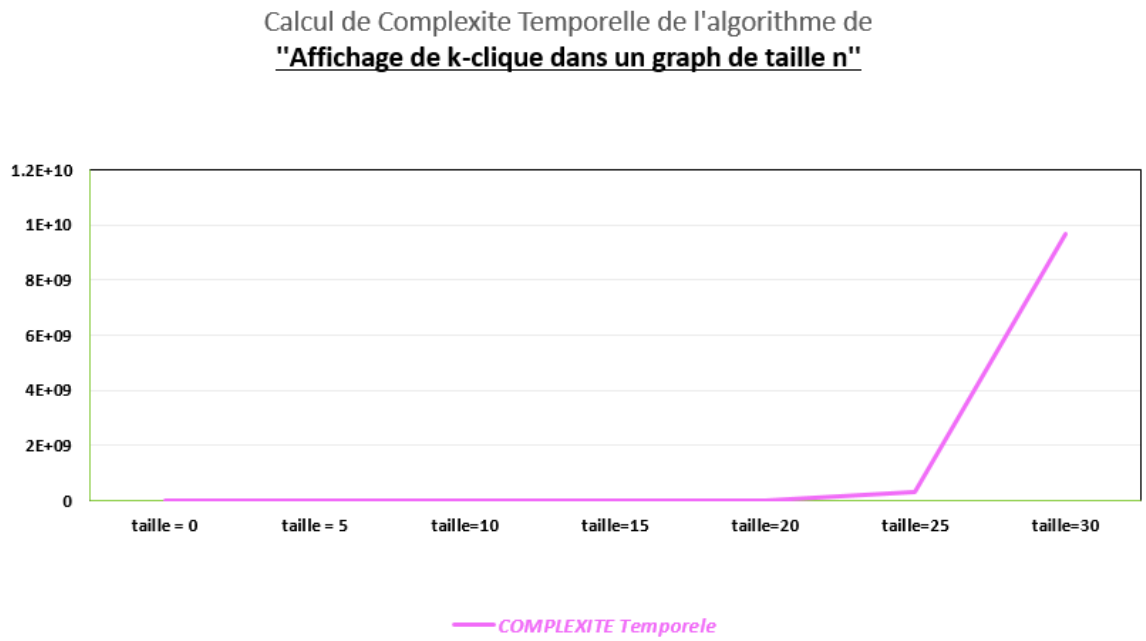
valeur	5	10	15	20	25	30
Complexité temporelle $O(\mathbf{km\alpha^{k-2}})$	**623750**α^3	**1247500**α^8	**1871250**α^{13}	**2495000**α^{18}	**3118750**α^{23}	**3742500**α^{28}
Complexité spatiale O(n+m)	**125250**	**125250**	**125250**	**125250**	**125250**	**125250**

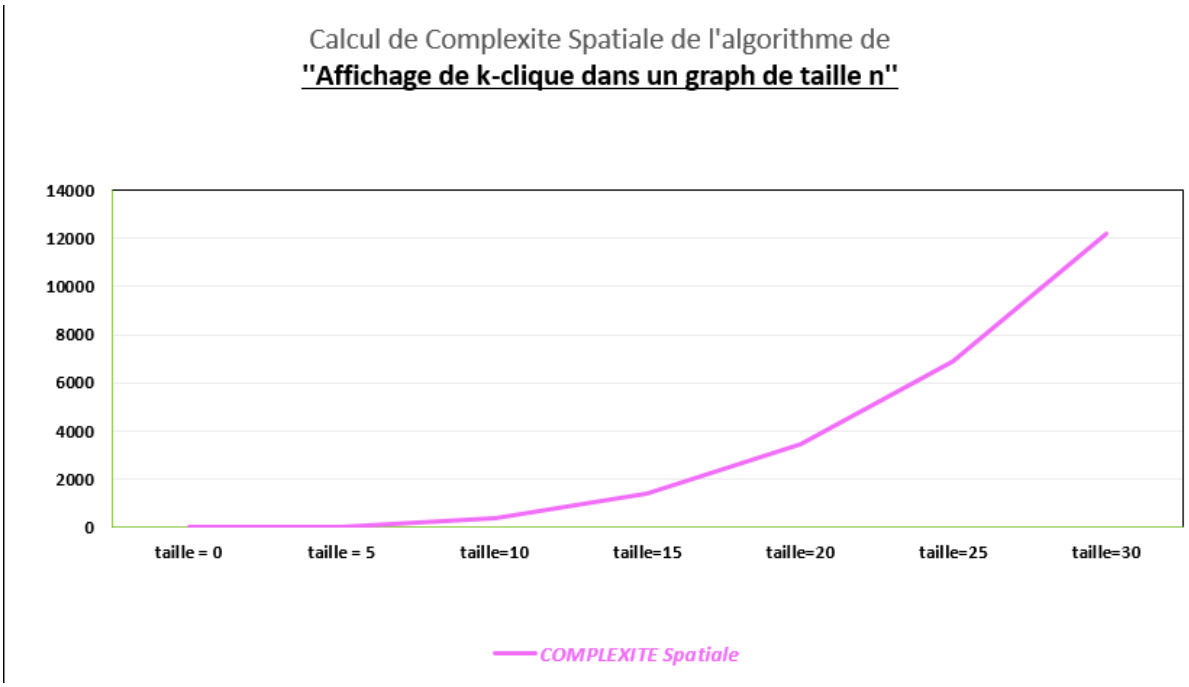
Tel que α est l'arboricité de graphe G.

2)-B simuler la complexité temporelle et spatiale théorique de l'algorithme de vérification :

On varie la taille du graphe (matrice) pour une clique de taille k =3 :

Valeurs	Taille=5	Taille=10	Taille=15	Taille=20	Taille=25	Taille=30
Complexité temporelle $O(k^2 * 2^n)$	288	9216	294912	9437184	301989888	9663676416
Complexité spatiale O(k*C(n,k))	30	360	1365	3420	6900	12180





2)-C Calcule de la complexité temporelle et spatiale expérimentale de l'algorithme de vérification.

Les Tests Du temps d'exécution en variant la taille du problème
(Taille de graph)
Dans ces tests on fixe le nombre k a 3 (k = 3)

Dans les tests il n'y a pas de ***(meilleur cas ,pire cas ,moyen cas)***
Car il faut tester chaque combinaison possible si elle est une clique

Pour un graph de taille = 5

(Meilleur cas + Moyen Cas + Pire Cas)

```
Quelles sont les dimensions de la matrice n*n? (n > 0)
Donner n
5

 La matrice d'adjacence choisi par random est :
1  1  1  0  1
1  1  1  0  0
1  1  1  1  1
0  0  1  0  1
1  0  1  1  0
Donner le nombre k pour verifier si il existe une k-clique (k < 5)
3

 Les cliques de taille 3 trouve dans le graph donne sont:
La combinaison : 0 1 2 ===> Une clique
La combinaison : 0 2 4 ===> Une clique
La combinaison : 2 3 4 ===> Une clique

 Nombre total de clique de taille 3 trouve dans le graph donne est: 3
Press any key to continue . . .
Temps D'execution en Nanosecondes: 792500 ns
Press any key to continue . . .
```

Pour un graph de taille = 10

(Meilleur cas + Moyen Cas + Pire Cas)

```
Quelles sont les dimensions de la matrice n*n? (n > 0)
Donner n
10

 La matrice d'adjacence choisi par random est :
0  1  1  1  1  1  1  1  1  0
1  0  1  1  0  1  0  0  0  1
1  1  1  0  1  1  1  1  1  1
1  1  0  0  1  1  1  1  1  0
1  0  1  1  1  1  1  0  1  1
1  1  1  1  1  0  0  1  1  1
1  0  1  1  1  0  0  1  1  1
1  0  1  1  0  1  1  1  1  1
1  0  1  1  1  1  1  1  1  0
0  1  1  0  1  1  1  1  0  0
Donner le nombre k pour verifier si il existe une k-clique (k < 10)
3

 Les cliques de taille 3 trouve dans le graph donne sont:
La combinaison : 0 1 2 ===> Une clique
```

```
La combinaison : 4 6 8 ===> Une clique
La combinaison : 4 6 9 ===> Une clique
La combinaison : 5 7 8 ===> Une clique
La combinaison : 5 7 9 ===> Une clique
La combinaison : 6 7 8 ===> Une clique
La combinaison : 6 7 9 ===> Une clique

 Nombre total de clique de taille 3 trouve dans le graph donne est: 53
Press any key to continue . . .
Temps D'execution en Nanosecondes: 20191500 ns
Press any key to continue . . .
```

Pour un graph de taille = 15

(Meilleur cas + Moyen Cas + Pire Cas)

```
Quelles sont les dimensions de la matrice n*n? (n > 0)
Donner n
15

 La matrice d'adjacence choisi par random est :
0  1  0  0  0  1  1  1  1  1  1  0  1  1  1
1  1  1  1  1  1  1  0  1  1  1  1  0  1  1
0  1  1  0  0  1  1  0  1  1  1  1  1  1  0
0  1  0  1  1  1  0  0  1  0  0  0  1  1  0
0  1  0  1  1  0  1  1  1  0  1  1  1  1  1
1  1  1  1  0  0  1  0  1  0  1  1  0  0  0
1  1  1  0  1  1  0  0  1  1  1  1  1  1  1
1  0  0  0  1  0  0  0  1  1  1  1  1  1  1
1  1  1  1  1  1  1  1  0  1  0  1  0  1  1
1  1  1  0  0  0  1  1  1  1  1  1  0  1  1
1  1  1  0  1  1  1  1  0  1  0  1  1  0  1
0  1  1  0  1  1  1  1  1  1  1  0  0  1  1
1  0  1  1  1  0  1  1  0  0  1  0  0  1  1
1  1  1  1  1  0  1  1  1  1  0  1  1  1  1
1  1  0  0  1  0  1  1  1  1  1  1  1  1  1
Donner le nombre k pour verifier si il existe une k-clique (k < 15)
3

 Les cliques de taille 3 trouve dans le graph donne sont:
La combinaison : 0 1 5 ===> Une clique
La combinaison : 0 1 6 ===> Une clique
```

```
La combinaison : 10 12 14 ===> Une clique
La combinaison : 11 13 14 ===> Une clique
La combinaison : 12 13 14 ===> Une clique

 Nombre total de clique de taille 3 trouve dans le graph donne est: 174
Press any key to continue . . .
Temps D'execution en Nanosecondes: 62501900 ns
Press any key to continue . . .
```

Pour un graph de taille = 20

(Meilleur cas + Moyen Cas + Pire Cas)

```
Quelles sont les dimensions de la matrice n*n? (n > 0)
Donner n
20

 La matrice d'adjacence choisi par random est :
1  0  0  1  1  0  0  1  1  1  1  1  0  1  1  1  1  1  1  1
0  0  0  1  1  1  1  1  1  1  1  1  1  0  1  1  1  1  0  1
0  0  0  0  1  0  0  1  0  1  0  0  1  1  1  0  1  0  1  0
1  1  0  0  1  0  0  1  1  1  1  1  1  1  1  1  1  0  1  1
1  1  1  1  0  1  1  0  1  1  1  1  1  0  0  0  1  1  1  1
0  1  0  0  1  0  1  1  1  1  0  1  1  1  0  1  0  1  0  1
0  1  0  0  1  1  0  0  1  1  1  1  1  0  0  1  1  0  1  1
1  1  1  1  0  1  0  0  1  0  1  0  1  1  1  1  0  0  0  1
1  1  0  1  1  1  1  1  0  1  1  1  1  1  1  1  0  1  0  0
1  1  1  1  1  1  1  0  1  0  1  1  1  1  1  1  1  1  0  1
1  1  0  1  1  0  1  1  1  1  1  0  0  1  1  1  0  0  1  1
1  1  0  1  1  1  1  0  1  1  0  1  1  1  1  1  1  1  0  1
0  1  1  1  1  1  1  1  1  1  0  1  1  0  0  1  1  1  1  1
1  0  1  1  0  1  0  1  1  1  1  1  0  1  1  1  0  0  1  1
1  1  1  1  0  0  0  1  1  1  1  1  0  1  1  1  1  1  0  1
1  1  0  1  0  1  1  1  1  1  1  1  1  1  1  0  1  0  1  1
1  1  1  1  1  0  1  0  0  1  0  1  1  0  1  1  1  1  1  1
1  1  0  0  1  1  0  0  1  1  0  1  1  0  1  0  1  1  0  1
1  0  1  1  1  0  1  0  0  0  1  0  1  1  0  1  1  0  1  1
1  1  0  1  1  1  1  1  0  1  1  1  1  1  1  1  1  1  1  0
Donner le nombre k pour verifier si il existe une k-clique (k < 20)
3

 Les cliques de taille 3 trouve dans le graph donne sont:
La combinaison : 0 3 4 ===> Une clique
```

```
La combinaison : 15 18 19 ===> Une clique
La combinaison : 16 17 19 ===> Une clique
La combinaison : 16 18 19 ===> Une clique

 Nombre total de clique de taille 3 trouve dans le graph donne est: 441
Press any key to continue . . .
Temps D'execution en Nanosecondes: 176762000 ns
Press any key to continue . . .
```

Pour un graph de taille = 25

(Meilleur cas + Moyen Cas + Pire Cas)

```
Quelles sont les dimensions de la matrice n*n? (n > 0)
Donner n
25

 La matrice d'adjacence choisi par random est :
0  0  1  1  1  0  1  1  0  1  1  1  0  0  1  1  1  1  1  0  1  1  0  1  0
0  0  0  0  1  1  1  1  0  0  1  1  1  1  1  0  1  1  1  0  1  1  1  1  1
1  0  1  1  1  1  1  1  0  1  1  1  0  1  1  1  1  1  1  1  1  1  0  1  1
1  0  1  1  1  1  1  1  1  0  0  1  1  0  1  1  0  1  1  1  0  1  1  0  1
1  1  1  1  1  1  1  0  1  1  1  1  1  0  1  1  1  0  1  0  1  1  1  1  0
0  1  1  1  1  1  0  0  1  1  1  1  1  1  0  1  1  1  1  0  1  1  0  1  1
1  1  1  1  1  0  1  0  0  1  0  0  0  1  1  1  1  1  0  1  1  1  1  1  0
1  1  1  1  0  0  0  0  1  1  1  0  1  1  0  1  1  1  1  0  1  0  1  1  1
0  0  0  1  1  1  0  1  0  1  1  0  0  1  1  1  1  0  0  1  0  0  0  1  0
1  0  1  0  1  1  1  1  1  0  1  1  1  0  1  0  1  1  1  1  0  1  0  1  1
1  1  1  0  1  1  0  1  1  1  0  1  0  1  1  0  0  0  1  1  1  1  1  0  1
1  1  1  1  1  1  0  0  0  1  1  0  1  0  0  1  1  1  1  1  1  1  0  1  1
0  1  0  1  1  1  0  1  0  1  0  1  0  1  1  1  1  1  1  0  0  1  1  1  1
0  1  1  0  0  1  1  1  1  0  1  0  1  0  0  1  1  1  1  1  1  1  1  1  0
1  1  1  1  1  0  1  0  1  1  1  0  1  0  0  1  0  1  1  1  1  1  0  0  1
1  0  1  1  1  1  1  1  1  0  0  1  1  1  1  0  1  0  1  1  1  0  1  1  1
1  1  1  0  1  1  1  1  1  1  0  1  1  1  0  1  1  1  1  1  1  1  1  0  1
1  1  1  1  0  1  1  1  0  1  0  1  1  1  1  0  1  0  1  0  1  1  1  1  1
1  1  1  1  1  1  0  1  0  1  1  1  1  1  1  1  1  1  0  1  1  1  1  1  1
0  0  1  1  0  0  1  0  1  1  1  1  0  1  1  1  1  0  1  1  0  1  0  1  1
1  1  1  0  1  1  1  1  0  0  1  1  0  1  1  1  1  1  1  0  0  1  1  1  1
1  1  1  1  1  1  1  0  0  1  1  1  1  1  1  0  1  1  1  1  1  1  1  0  1
0  1  0  1  1  0  1  1  0  0  1  0  1  1  0  1  1  1  1  0  1  1  1  1  1
1  1  1  0  1  1  1  1  1  1  0  1  1  1  0  1  0  1  1  1  1  0  1  1  1
0  1  1  1  0  1  0  1  0  1  1  1  1  0  1  1  1  1  1  1  1  1  1  1  1
Donner le nombre k pour verifier si il existe une k-clique (k < 25)
3

 Les cliques de taille 3 trouve dans le graph donne sont:
La combinaison : 0 2 3 ===> Une clique
La combinaison : 0 2 4 ===> Une clique
La combinaison : 0 2 6 ===> Une clique
```

```
La combinaison : 20 22 23 ===> Une clique
La combinaison : 20 22 24 ===> Une clique
La combinaison : 20 23 24 ===> Une clique
La combinaison : 21 22 24 ===> Une clique
La combinaison : 22 23 24 ===> Une clique

 Nombre total de clique de taille 3 trouve dans le graph donne est: 920
Press any key to continue . . .
Temps D'execution en Nanosecondes: 341295700 ns
Press any key to continue . . .
```

Pour un graph de taille = 30

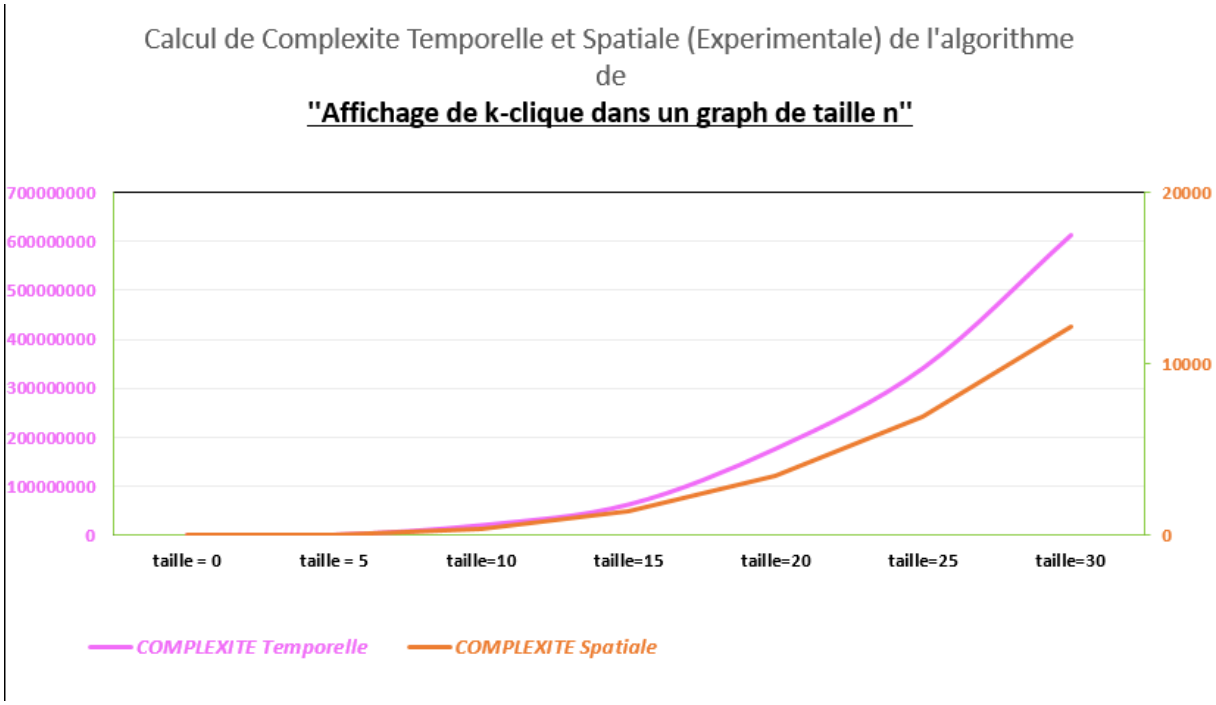
(Meilleur cas + Moyen Cas + Pire Cas)

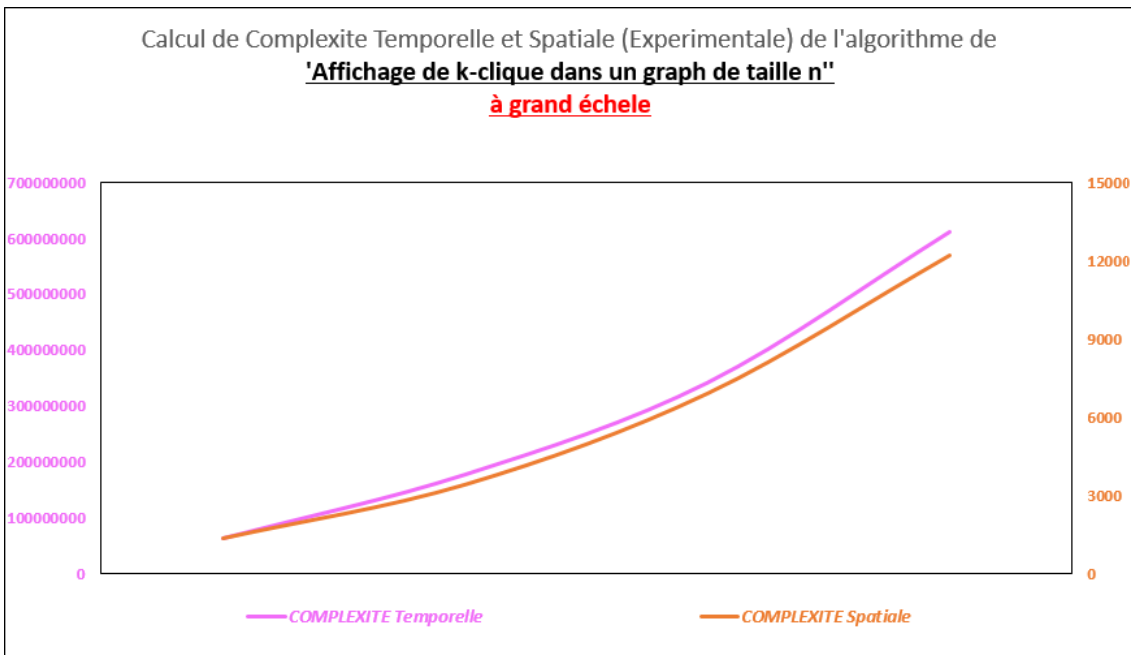
```
Quelles sont les dimensions de la matrice n*n? (n > 0)
Donner n
30

 La matrice d'adjacence choisi par random est :
0  1  1  1  1  1  0  1  1  1  0  0  1  1  1  0  0  1  1  1  0  1  1  1  1  1  1  1  0  0
1  0  1  0  1  1  0  0  1  0  0  0  0  1  1  1  0  1  1  1  1  1  1  1  0  1  1  1  1  1
1  1  1  1  1  0  1  1  0  1  0  1  1  1  1  1  1  1  1  1  1  1  1  1  0  1  1  1  0  1  0
1  0  1  0  1  1  1  1  1  1  1  0  1  1  0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
1  1  1  1  0  1  1  1  0  1  1  1  1  1  0  1  0  0  1  1  1  1  1  0  1  0  1  1  0  0
1  1  0  1  1  1  0  1  1  1  1  1  1  1  1  1  1  1  1  1  0  1  0  1  1  0  1  1  1  1
0  0  1  1  1  0  0  1  1  1  1  1  0  1  0  1  0  0  1  0  1  1  1  1  1  1  1  0  1
1  0  1  1  1  1  1  1  1  0  1  1  0  0  1  0  1  1  1  1  1  1  1  1  1  1  1  0  0  0
1  1  0  1  0  1  1  1  1  1  1  0  1  0  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0
1  0  1  1  1  1  1  0  1  1  1  1  0  0  1  1  0  1  1  1  0  1  1  1  0  1  1  1  1  0
0  0  0  1  1  1  1  1  1  1  0  1  0  1  1  0  0  0  1  1  1  1  1  1  1  1  0  1  0  1
0  0  1  0  1  1  1  1  0  1  1  1  1  0  1  1  1  0  1  1  1  1  1  1  1  1  1  0  1  1
1  0  1  1  1  1  0  0  1  0  0  1  0  1  0  1  1  1  1  1  1  1  1  0  0  1  0  0  1  0  1
1  1  1  1  1  1  1  0  0  0  1  0  1  1  1  0  1  1  1  1  1  1  1  1  1  0  0  1  1  1
1  1  1  0  0  1  0  1  1  1  1  1  0  1  0  1  1  1  1  1  0  1  1  1  1  1  0  1  0  0
0  1  1  1  1  1  1  0  1  1  0  1  1  0  1  1  0  1  0  1  1  0  1  0  1  1  1  0  0  1
0  0  1  1  0  1  0  1  1  0  0  1  1  1  1  0  1  1  1  1  1  0  1  1  1  1  0  1  0  1
1  1  1  1  0  1  0  1  1  1  0  0  1  1  1  1  1  1  1  1  1  1  1  0  1  1  1  1  1  1
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  1  1  0  1  1  1  0  1  1  0  1  1  1  1
1  1  1  1  1  1  0  1  1  1  1  1  1  1  1  1  1  1  1  0  1  1  1  1  0  1  0  1  0  1
0  1  1  1  1  0  1  1  1  0  1  1  1  1  0  1  1  1  1  1  1  1  1  1  0  0  1  1  1  0
1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  0  1  1  1  1  0  0  1  1  0  1  1  1  0
1  1  1  1  1  0  1  1  1  1  1  1  0  1  1  1  1  1  0  1  1  0  1  0  0  1  0  1  0  1
1  1  0  1  0  1  1  1  1  1  1  1  0  1  1  0  1  0  1  1  1  1  0  0  0  1  0  1  0  1
1  0  1  1  1  1  1  1  1  0  1  1  1  1  1  1  1  1  1  0  0  1  0  0  1  1  1  1  1  1
1  1  1  1  0  0  1  1  1  1  1  1  0  0  1  1  1  1  0  1  0  0  1  1  1  0  0  1  0  0
1  1  1  1  1  1  1  1  1  1  0  1  0  0  0  1  0  1  1  0  1  1  0  0  1  0  1  1  1  0
1  1  0  1  1  1  1  0  1  1  1  0  1  1  1  0  1  1  1  1  1  1  1  1  1  1  0  0  1
0  1  1  1  0  1  0  0  1  1  0  1  0  1  0  0  0  1  1  0  1  1  0  0  1  0  1  0  1  1
0  1  0  1  0  1  1  0  0  0  1  1  1  1  0  1  1  1  1  1  0  0  1  1  1  0  0  1  1  0
Donner le nombre k pour verifier si il existe une k-clique (k < 30)
3

 Les cliques de taille 3 trouve dans le graph donne sont:
La combinaison : 0 1 2 ===> Une clique
La combinaison : 0 1 4 ===> Une clique
La combinaison : 0 1 5 ===> Une clique
```

```
La combinaison : 24 26 28 ===> Une clique
La combinaison : 24 27 29 ===> Une clique
La combinaison : 24 28 29 ===> Une clique

 Nombre total de clique de taille 3 trouve dans le graph donne est: 1643
Press any key to continue . . .
Temps D'execution en Nanosecondes: 611518800 ns
Press any key to continue . . .
```

Graphe de Complexité de l'algorithme de vérification Expérimentale en fonction de la taille du problème *(taille du graph)* :

VALEURS	taille = 0	taille = 5	taille=10	taille=15	taille=20	taille=25	taille=30
COMPLEXITE Temporelle	0	792500	20191500	62501900	176762000	341295700	611518800
COMPLEXITE Spatiale	0	30	360	1365	3420	6900	12180

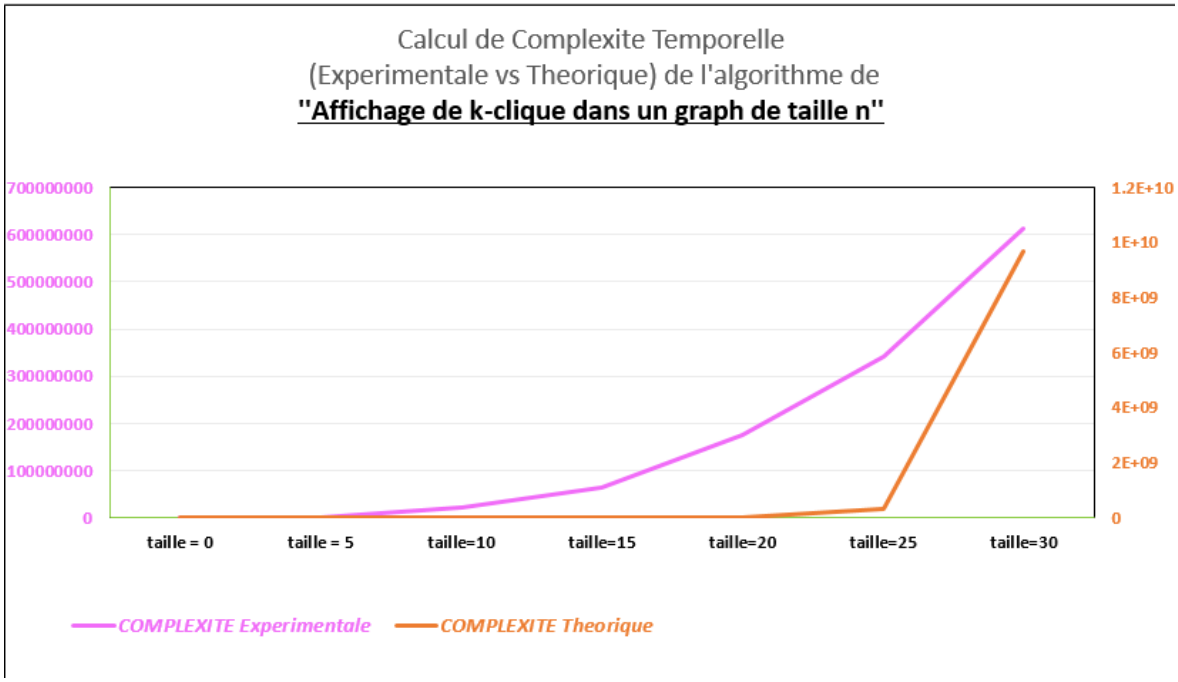


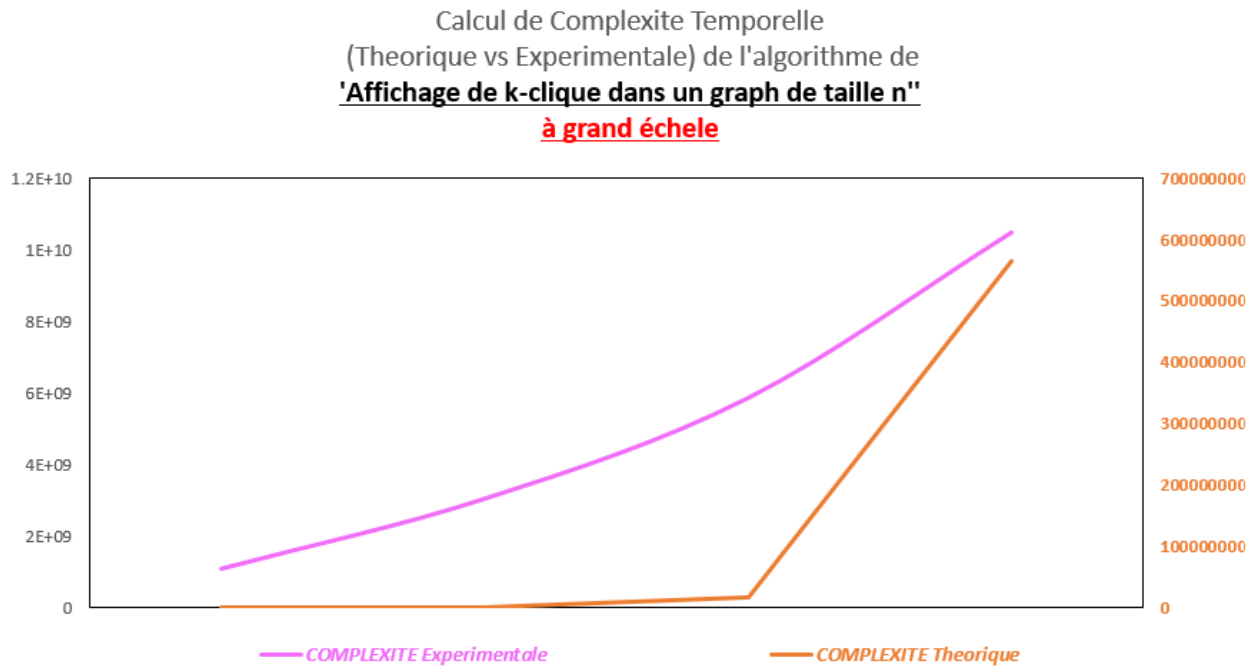


2)-C Analyse des résultats (Comparaison entre Complexite Théorique et Expérimentale)

Complexité Théorique

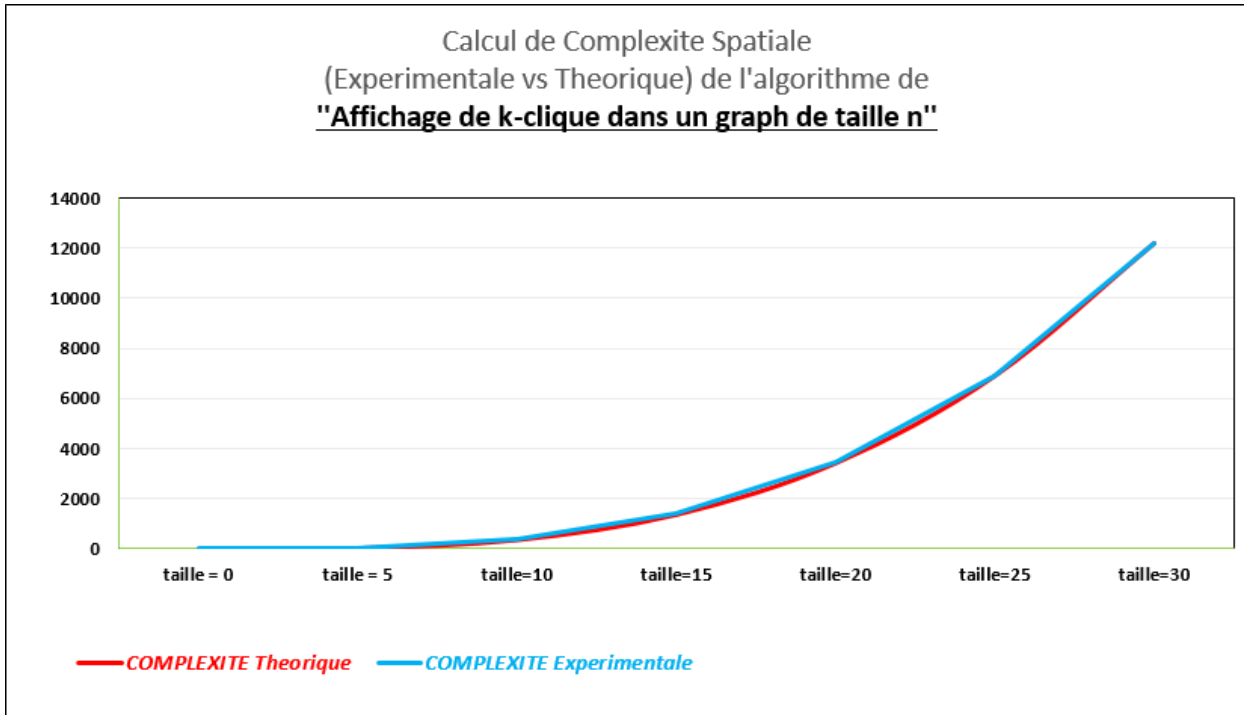
VALEURS	taille = 0	taille = 5	taille=10	taille=15	taille=20	taille=25	taille=30
COMPLEXITE Experimentale	0	792500	20191500	62501900	176762000	341295700	611518800
COMPLEXITE Theorique	0	288	9216	294912	9437184	301989888	9663676416





Complexité Spatiale

VALEURS	taille = 0	taille = 5	taille=10	taille=15	taille=20	taille=25	taille=30
COMPLEXITE Theorique	0	30	360	1365	3420	6900	12180
COMPLEXITE Experimentale	0	30	360	1365	3420	6900	12180



Conclusion :

Le problème de trouver des cliques dans les graphes est un problème complexe et difficile. Ce dernier devient d'autant plus difficile lorsque le graphe sur lequel nous travaillons est grand. Pour résoudre ce problème il existe de nombreux algorithmes et heuristiques. Dans cette 2 -ème partie du projet nous avons proposé et évalué une méthode pour résoudre ce problème. On a remarqué durant le calcul de la complexité que la complexité était polynomiale ce qui fait son appartenance à NP.

Références :

[1] Li, Ronghua, et al. "Ordering Heuristics for k-clique Listing." *Proc. VLDB Endow.* (2020).

[2] A. Angel, N. Koudas, N. Sarkas, and D. Srivastava. Dense subgraph maintenance under streaming edge weight updates for real-time story identification. PVLDB, 5(6):574–585, 2012.

[3] R. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng. Skyline community search in multi-valued networks. In SIGMOD, 2018.

[4] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Influential community search in large networks. PVLDB, 8(5):509–520, 2015.

[5] R.-H. Li, J. X. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. IEEE Trans. Knowl. Data Eng., 26(10):2453–2465, 2014.

[6] L. Qin, R. Li, L. Chang, and C. Zhang. Locally densest subgraph discovery. In KDD, 2015.

[7] A. E. Sariyuce, C. Seshadhri, A. Pinar, and ¨ U. V. C¸ ataly ¨ urek. ¨ Finding the hierarchy of dense subgraphs using nucleus decompositions. In WWW, 2015.

[8] Print combinations of r elements in an array of size n: https://www.youtube.com/watch?v=7IQHYbmuoVU&list=LL&index=7

[9] Find all possible combinations of K numbers from 1 to n: https://www.enjoyalgorithms.com/blog/find-all-combinations-of-k-numbers

FIN