Résolution de problèmes de cliques dans les grands graphes

Jocelyn Bernard, Hamida Seba

Résolution de problèmes de cliques dans les grands graphes

Jocelyn Bernard
sous la direction de : Hamida Seba

Université Claude Bernard - Lyon 1
Équipe GOAL

Abstract. Les problèmes MCE (Maximal Clique Enumeration) et MCP (Maximum Clique Problem) sont des problèmes que l'on rencontre dans l'analyse des graphes de données et leur exploration. Ce sont cependant des problèmes difficiles (NP-difficile pour MCE et NP-Complet pour MCP) pour lesquels des solutions adaptées doivent être conçues pour les grands graphes. Nous nous proposons dans ce travail de répondre à ces problèmes en travaillant sur une version compressée du graphe initial. Une telle approche semble intéressante à la fois en temps de calcul et en espace mémoire. Nous avons donc implémenté notre approche et l'avons comparée à plusieurs solutions de la littérature.

Abstract. The MCE (Maximal Clique Enumeration) and MCP (Maximum Clique Problem) are problems that are encontred in data graph analyses and mining. However, these problems are consequently difficult (NP-hard for MCE and NP-complete for MCP). Consequently, appropriate solutions must be proposed in the case of large graphs. We propose in this work to answer these problems by working on a compressed version of the original graph. This approach seems interesting both in terms of computation time and memory space. So we implemented our approach and we compared with several solutions from the literature.

1 Introduction

Un graphe $\mathbf{G}=(\mathbf{V},\mathbf{E})$ est un outil de modélisation qui comprend un ensemble de nœuds $\mathbf{E}$ ainsi qu'un ensemble d'arêtes $\mathbf{V}$. Une arête est un lien entre deux nœuds. Les arêtes peuvent être orientées, on parle alors de *graphes orientés*. Les arêtes peuvent également être pondérées, nommées ou multiples, à savoir que deux nœuds peuvent être reliés parallèlement par plusieurs arêtes, on se trouve alors respectivement dans des cadres de *graphes pondérés*, *graphes étiquetés* et de *multigraphes*. Dans le cas où les arêtes sont symétriques (elles ne sont pas dirigées), non-multiples et non-pondérées, on parle alors *de graphes simples*.

Les graphes ont comme vocation d'être utilisés en tant que structures de données permettant de modéliser des objets ou des problèmes avec de nombreuses applications et utilisations, notamment dans les domaines des réseaux sociaux,

des réseaux informatiques ou encore de la génétique. Parmi ces applications se trouve la recherche de structures, par exemple, dans le cadre de la génétique on peut rechercher des interactions protéines-protéines [1], on peut également vouloir chercher des communautés dans les graphes issus des réseaux sociaux, etc.

Dans ce travail nous nous intéresserons à des structures de cliques qui ont fait l'objet de plusieurs études [2–4].

Une clique est un sous-ensemble complet de nœuds du graphe. C'est-à-dire que chacun des nœuds du sous-ensemble est connecté avec l'intégralité des autres.

La clique, comme toute structure précise du graphe permet donc d'extraire de l'information de ce dernier. Cependant, trouver ces structures est considéré comme un problème difficile. Dans la recherche de cliques, deux grands problèmes existent :

1. Le problème de la clique maximum ou MCP pour l'anglais Maximum Clique Problem est un problème NP-complet [5] que l'on peut définir de la manière suivante :

   Une clique c de taille k[1] est un sous-semble complet de nœuds $v \in \mathbf{E}$. Le problème de clique maximum consiste à trouver la clique dont la taille k est la plus grande pour le graphe $\mathbf{G}$ (s'il y en a plusieurs avec la même taille, il suffit juste d'en trouver une).

2. Le problème d'énumération de cliques maximales ou MCE pour l'anglais Maximal Clique Enumeration est un problème théoriquement NP-Difficile qui se définit de la manière suivante :

   Une clique c est maximale si elle n'est pas incluse dans une clique de taille supérieure. Le problème d'énumération de cliques maximales consiste à lister l'ensemble des cliques maximales du graphe $\mathbf{G}$.

Les problèmes MCE et MCP ont des applications dans différents domaines tels que la biologie, par exemple pour la détection d'interaction entre protéines [6], ou encore en ingénierie électrique [7], etc. Dans la théorie des graphes le MPC permet par exemple de minorer le nombre chromatique d'un graphe.

Dans le cas des grands graphes ces problèmes deviennent donc difficiles à traiter car les données en entrée sont nombreuses (le nombre de nœuds et le nombre d'arêtes) et le temps d'exécution est exponentiel avec la taille de ces données. Dans ce cas, plusieurs solutions ont été envisagées : trouver des solutions non-exactes mais proches d'un optimal avec des heuristiques,

[1] où k est le nombre de nœuds v de la clique

partitionner le graphe et traiter chaque partition à part, etc. Dans cet article, nous explorons une nouvelle solution qui consiste à travailler sur des données plus petites en compressant le graphe.

La compression de graphes est une opération qui permet la diminution du nombre d'arêtes ou de nœuds du graphe. Cette opération permet par exemple de rendre le graphe plus facile à transmettre ou à lire. Les opérations de compressions de graphes sont possibles sur plusieurs types de graphes : simples [8], orientés [9], étiquetés [10], etc.

Nous proposons dans cet article de résoudre les problèmes MCE et MCP sur un graphe compressé (sans le décompresser). Les avantages d'une telle approche sont :

- Le graphe compressé prend moins d'espace mémoire et peut être chargé en mémoire central lors de son traitement. Ceci permet de réduire le nombre d'entrées/sorties engendrées si le graphe d'origine ne tient pas en mémoire centrale.
- Le graphe compressé étant plus petit, un gain de temps de traitement sera éventuellement réalisé.

Nous avons implémenté cette approche et l'avons comparée à la plupart des algorithmes existants. Les résultats sont prometteurs et ouvrent plusieurs perspectives. Le reste du rapport est structuré comme suit : dans la section 2 nous proposons un état de l'art sur la compression de graphes et sur les algorithmes utilisés pour résoudre les problèmes MCE et MCP. Dans la section 3 nous détaillons notre raisonnement pour la résolution des problèmes MCE et MCP sur le graphe compressé. Dans la section 4 nous proposons une évaluation de notre travail via une expérimentation. Enfin, dans la section 5, nous présentons la conclusion et les perspectives futures qu'offre notre méthode.

2 État de l'art

2.1 La compression de graphes

Nous décrivons ici les principales méthodes de compression que nous avons trouvées dans la littérature. Il existe plusieurs façons de compresser un graphe, dépendant du type de graphe mais aussi de ce que l'on veut faire du graphe.

Dans [8], l'auteur propose une compression de graphe sans perte à l'aide d'un graphe réduit obtenu par agrégation de nœuds dont les arêtes de voisinages sont proches. Le graphe réduit diminue le nombre d'arêtes et de nœuds et permet de retrouver le graphe original à l'aide d'une liste de correction. La figure 1 illustre un exemple de graphe compressé selon [8].

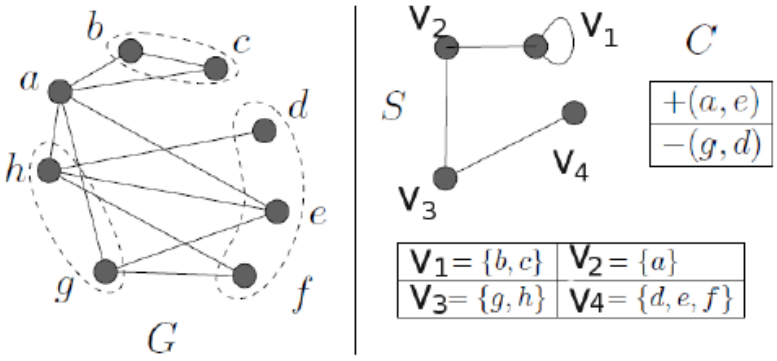


Fig. 1. Exemple de compression selon la méthode de [8]

Dans [9] et [11] les auteurs proposent de compresser des graphes issus du web à l'aide de la différence entre les listes de voisinages entre deux nœuds : un nœud est compressé à l'aide d'un autre auquel on ajoute une correction par rapport à sa liste de voisinage (avec les nœuds voisins en plus ou en moins).

Tian et al. [10] proposent quand à eux une méthode de compression supervisée par l'utilisateur afin de pouvoir visualiser des graphes étiquetés plus facilement selon une taille souhaitée. L'utilisateur définit une taille de graphe et l'algorithme se charge de définir un seuil d'erreur acceptable dans la compression pour répondre à la demande de l'utilisateur tout en évitant les pertes d'informations.

Pour les graphes pondérés Toivonen et al. [12] proposent une méthode de compression avec peu de pertes permettant de compresser de tel type de graphes. Les nœuds et les arêtes sont compressés en super-noeuds et super-arêtes. Le poids d'une super-arête est approximé par le poids des arêtes qui la composent.

La décomposition modulaire [13] peut également être utilisée pour compresser un graphe [14, 15]. La méthode de la décomposition modulaire revient à réaliser différents agglomérats de nœuds, en fonction de leurs voisinages. Ces agglomérats sont appelés des modules. Il existe différents types de modules :

- les modules feuilles : chaque module feuille correspondent à un nœud simple dans le graphe original. C'est une feuille de l'arbre de décomposition produit par la compression modulaire.
- les modules séries : un nœud module série correspond à un ensemble de nœuds formant un sous-graphe complet (chacun des nœuds fils du nœud série est relié avec les autres).

– les modules parallèles : un nœud module parallèle correspond à un ensemble de nœuds formant un stable[2]. Il n'existe aucune arête entre les descendants du nœud parallèle. Le complémentaire du sous-graphe le représentant est un graphe complet.
– les nœuds premiers : un module premier correspond à un sous-graphe qui n'est pas complet et dont le graphe complémentaire[3] est également non complet.

La figure 2 illustre la compression d'un graphe en utilisant la décomposition modulaire. L'imbrication des modules, appelée arbre de décomposition modulaire, est présentée dans la figure 3. Les nœuds qui possèdent un voisinage identique sont coloriés de la même manière dans le graphe initial (voir figure 2(a)).

– Les nœuds 1 et 2 ont le même voisinage et sont tous reliés entre eux. Ils forment un module série S(1-2) (voir figure 2(b)).
– Les nœuds 0 et 8 ont le même voisinage et ne possèdent aucune arête entre eux. Ils forment un module parallèle P(1-2) (voir figure 2(b)).
– Les nœuds 5,6 et 7 possèdent le même voisinage, cependant ils ne sont ni tous reliés entre eux, ni ne possèdent aucune arête entre eux. Pour compresser cette partie du graphe, il faut d'abord compresser les nœuds 5 et 6 en module série S(5-6), car ils sont reliés et possèdent le même voisinage (voir figure 2(b)/2(c)). Il suffit ensuite de compresser le nœud 7 avec le nœud-module que l'on vient de compresser dans un nœud parallèle P(S(5-6)-7) (voir figure 2(c)/2(d)).

On obtient un alors un graphe partiellement compressé (voir figure 2(d)).

On remarque alors que le sous-graphe composé des nœuds P(0-8), S(1-2), 3, 4 et P(S(5-6)-7) ne peut plus être compressé par des modules séries ou parallèles. En effet il n'existe pas de nœuds ayant le même voisinage. Nous sommes alors en présence d'un module premier. La décomposition étant finie, on peut construire l'arbre de décomposition (voir figure 3).

Le tableau 1 résume les différentes méthodes de compression que nous avons étudiés dans cette section. Nous les avons classées selon le type du graphe : simple, orienté, pondéré ou étiqueté.

2.2 Le problème de cliques MCE

Il existe plusieurs algorithmes et heuristiques pour le problèmes MCE. Ces algorithmes dépendant du type et des données des graphes (nombre de nœuds et d'arêtes, densités, diamètres). Nous décrivons dans cette section, les principaux algorithmes proposés dans la littérature.

[2] Un stable dans un graphe est un ensemble de sommets dont aucun n'est adjacent à un autre

[3] Le complémentaire d'un Graphe **G** est le graphe **G'** obtenu avec les mêmes nœuds **V** du graphe **G** mais dont les arêtes entre deux nœuds sont présentes dans **G'** uniquement si les deux nœuds ne sont pas adjacents dans **G**

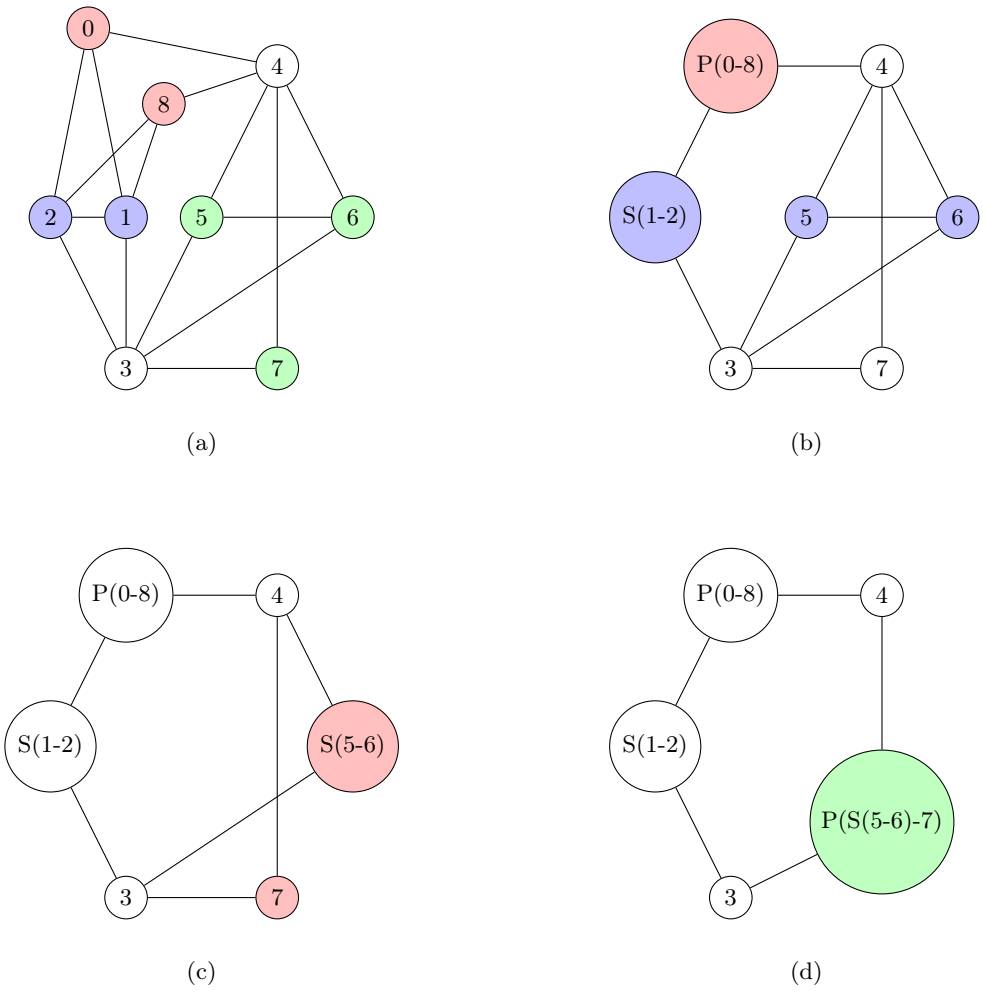


Fig. 2. Compression d'un graphe (a) avec deux étapes intermédiaires (b),(c), et le graphe compressé obtenu (d).

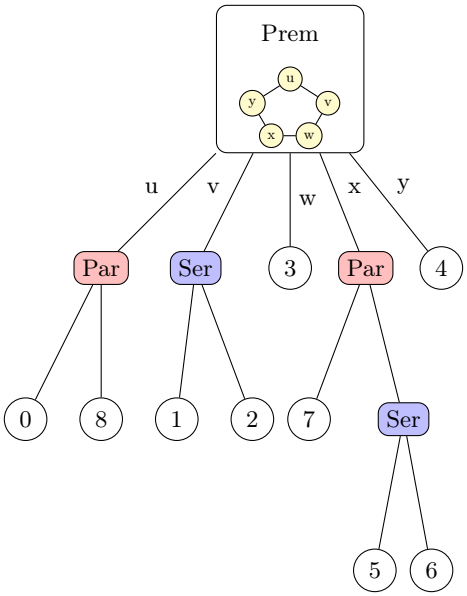


Fig. 3. Arbre de décomposition correspondant au graphe de la figure 2

Type de graphe	**Type de compression**	**Application**
Graphes simples	Compression l'aide d'une liste de correction [8]	Tout type de réseaux
	Décomposition par modules [13]	Réseau sociaux, Graphes biologie
Graphes orientés	Compression par différence entre les nœuds [9, 11]	Graphes web
Graphes étiquetés	Compression supervisée [10]	Schémas de bases de données, grands graphes
Graphes pondérés	Agglomération de nœuds et d'arêtes [12]	Grands graphes de réseaux sociaux ou de biologie

Table 1. Les méthodes de compression de graphes

2.2.1 MCE : L'algorithme issu de l'article de Bron et Kerbosh [2] est le premier algorithme créé pour trouver toutes les cliques maximales d'un graphe. Il fonctionne avec 3 ensembles :

- Un ensemble C qui contient la clique partielle construite à un instant t.
- Un ensemble T qui contient les nœuds candidats pour agrandir la clique partielle.
- Un ensemble D qui contient les nœuds ayant déjà été visités pour la construction d'une clique.

L'algorithme initialise donc T avec tous les nœuds **V** du graphe tandis que C et D sont vides. Ensuite l'algorithme réalise plusieurs itérations. Durant chacune d'elles, il sélectionne un pivot qui l'aide à choisir quels noeuds peuvent être ajoutés à la clique courante. Le choix du pivot permet de trouver toutes les cliques : celles où le pivot en fait partie, car ce ne sont que ses voisins qui sont enlevés de la boucle de sélection et celles où il n'en fait pas partie. La sélection du pivot étant aléatoire, l'expérimentation ne prend pas toujours le même temps et ne retourne pas toujours les cliques dans le même ordre. La procédure est détaillée dans Algorithme 1.

Algorithme 1 Procedure-MCE(C,T,D)

1: **if** $T \cup D = \emptyset$ **then**
2: Retourner C comme clique maximale
3: **end if**
4: choisir aléatoirement un pivot $p \in (T \cup D)$
5: **for each** v $\in T - \Gamma(p)$
6: $T \leftarrow T - v$
7: $C \leftarrow v$
8: Procedure-MCE($C, T \cap \Gamma(v), D \cap \Gamma(v)$)
9: $C \leftarrow C - v$
10: $D \leftarrow v$
11: **end for**

2.2.2 Tomita : L'algorithme de Tomita est aujourd'hui l'algorithme de référence. Sa complexité est de l'ordre de $O(3^{(n/3)})$. Il diffère de la recherche de Bron et Kerbosh par le choix du pivot qui permet un élagage généralement plus efficace. Il utilise lui aussi 3 ensembles :

- Un ensemble Q qui contient la clique partielle qui est construite à un instant t.
- Un ensemble $SUBG$ et un ensemble $CAND$ qui permettent de sélectionner le meilleur pivot possible, puis le meilleur candidat, dans le but d'agrandir la clique.

L'algorithme initialise l'ensemble Q à vide tandis que $SUBG$ et $CAND$ sont initialisés avec l'ensemble des nœuds V. L'élagage se fait par le choix du pivot. Ce choix permet une recherche plus rapide car il maximise la taille de l'ensemble $CAND$. De plus si l'ensemble $CAND$ est vide mais pas $SUBG$ on retourne en arrière car la clique courante C générée peut être un sous ensemble d'une clique maximale. La procédure est présenté dans Algorithme 2.

Algorithme 2 Procedure-Tomita(Q,$SUBG$,$CAND$)

1: **if** $SUBG = \emptyset$ **then**
2: Retourner Q comme clique maximale
3: **end if**
4: choisir pivot $u \in SUBG$ qui maximise $|CAND \cap \Gamma(u)|$
5: **while** $CAND - \Gamma(u) \neq \emptyset$
6: choisir q $\in CAND - \Gamma(u)$
7: $Q \leftarrow Q + q$
8: Procedure-Tomita(Q,$SUBG \cap \Gamma(u)$, $CAND \cap \Gamma(u)$)
9: $Q \leftarrow Q - q$
10: $CAND \leftarrow CAND - q$
11: **end while**

La figure 4 issue de Tomita [3] illustre cet algorithme.

2.2.3 Eppstein : Eppstein et al. [16] ont proposé une amélioration de l'algorithme de Tomita. Avant d'exécuter Tomita, ils proposent d'utiliser la dégénérescence de graphe.

Définition : La *dégénérescence* d'un graphe G est le plus petit nombre k de telle sorte que chaque sous-graphe S ∈ G contient un sommet de degré au plus k.

Pour chaque graphe G, on peut calculer son ordre de dégénérescence, qui est un ordre linéaire des sommets tels que chaque sommet a au plus d voisins plus tard dans l'ordre. La dégénérescence d'un graphe et son ordre peuvent être calculés en temps linéaire O(m) [17].

L'algorithme utilise les mêmes ensembles que Tomita, cependant ils ne sont pas initialisés de la même manière. (voir Algorithme 3).

Algorithme 3 Procedure-Eppstein(Q,$SUBG$,$CAND$)

1: **for each** vertex v_i dans l'ordre de dégénérescence $v_0, v_1, v_2, ...$ de G **do**
2: $SUBG \leftarrow \Gamma(v_i) \cap \{v_{i+1},...,v_{n-1}\}$
2: $CAND \leftarrow \Gamma(v_i) \cap \{v_0,...,v_{i-1}\}$
4: Procedure-Tomita(v_i,$SUBG$,$CAND$)
5: **end for**

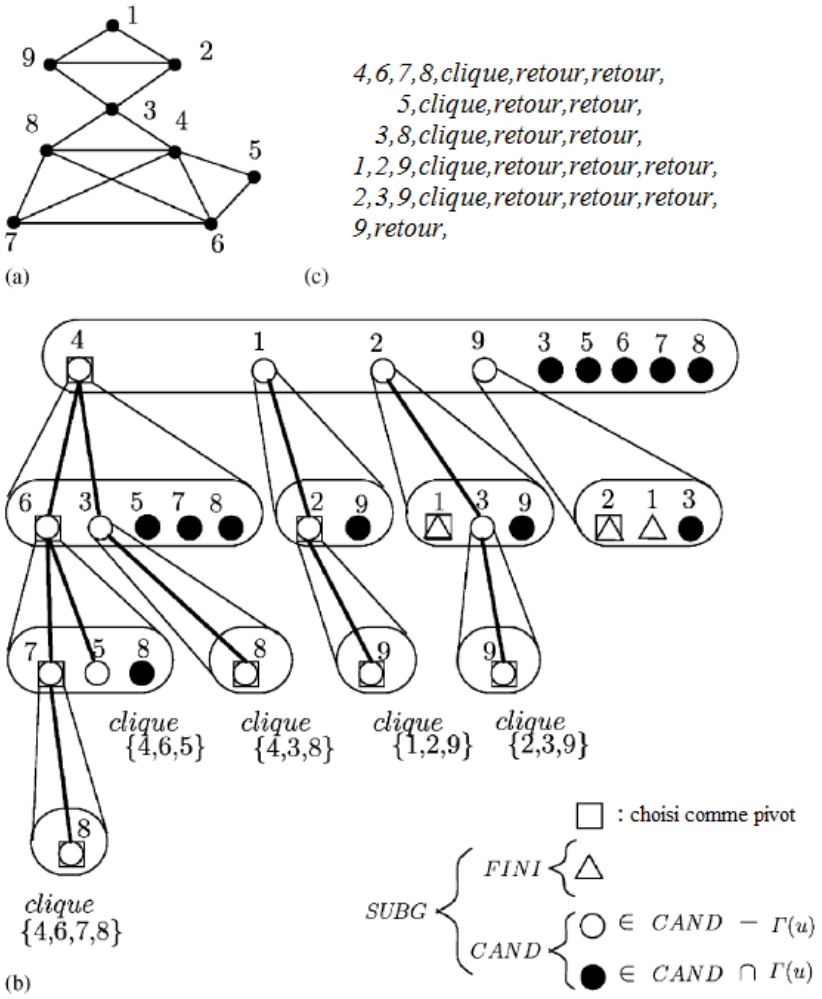


Fig. 4. Exécution de TOmita sur le graphe en entrée (a), l'arbre d'exploration de l'algorithme (b) et le résultat donné par l'algorithme (c)

2.2.4 RMC : Wang et al. [18] proposent quand à eux de lister seulement une partie des cliques maximales d'un graphe en évitant de retourner des cliques qui se chevauchent, c'est-à-dire dont les identifiants des nœuds qui les composent sont quasiment les mêmes. Pour cela, ils proposent la notion de τ-visibilité. La τ-visibilité est une variable qui permet d'écarter des cliques qui sont proches de la clique trouvée précédemment selon le seuil défini par l'utilisateur.

Les auteurs déclinent également l'algorithme en deux versions : une qui applique strictement le taux τ et une version randomisée qui l'applique à quelques nuances près. L'algorithme ressemble à celui de Tomita et d'Eppstein mais sort plus tôt de la fonction si la clique qui est en train d'être construite se trouve être trop similaire à la précédente.

2.3 Le problème de clique maximum (MCP)

Nous présentons dans cette partie une liste d'algorithmes proposés dans la littérature pour résoudre le problème de clique maximum dans un graphe.

2.3.1 MC et dérivés : MC est un algorithme exact qui trouve la clique maximum dans un graphe simple. Cet algorithme possède plusieurs extensions (voir [4]). Dans ce qui suit, nous présentons le principe de base de MC ainsi que ses principales variantes.

Le premier algorithme, **MC**, utilise 2 ensembles :

- Un ensemble C de nœuds, initialement vide, qui contient la clique partielle construite à un instant t et qui contient la clique maximum à la fin de l'algorithme.
- Un ensemble P, initialisé avec tous les nœuds du graphe, qui contient les nœuds candidats pour agrandir la clique partielle.

L'algorithme utilise en plus une variable max qui contient la taille de la clique maximum trouvée. La procédure détaillée est présentée dans Algorithme 4.

Algorithme 4 Procedure-MC(C,P,max)

```
1: while P != ∅ do
2:     if |C|+|P| < max : return
3:     choisir v le dernier élément de P
4:     C ← C + v
5:     if |P ∩ Γ(v)| == 0 && |C| > max
6:         Sauvegarder C comme plus la plus grande clique
7:         max ← |C|
8:     end if
9:     if |P ∩ Γ(v)| != 0
10:        Procedure-MC(C,P ∩ Γ(v),max)
11:    end if
12:    C ← C − v
13:    P ← P − v
14: end while
```

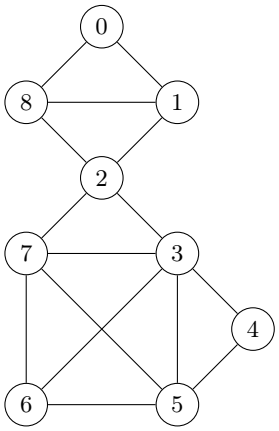
Une première modification permettant de gagner un peu de temps est la procédure MC0, on ne prend en compte que les voisins dont l'identifiant est inférieur au nœud pivot v sélectionné, ce qui évite des doublons.

[4] présente ensuite différentes versions de l'algorithme MC : MCQ, MCS et BBMC. MCS est lui-même décliné en 2 versions MCSa et MCSb. Tous ces algorithmes ont pour point commun de colorer le graphe, en fonction du tri préalablement effectué par une procédure de coloration choisie. Ces colorations, qui permettent de grouper les nœuds, sont ensuite utilisées par chacun des algorithmes pour résoudre le problème de clique maximale (voir[4] pour une description exhaustive de ces algorithmes). Ces fonctions de coloration (présentés par les algorithmes 5-1 à 5-3 présents dans l'annexe) prennent en paramètres la liste des nœuds V, de leurs degrés *deg* et, pour la troisième procédure, de la somme des degrés de leurs voisins *nebDeg*. Les procédures retournent la liste *ColOrd*, qui contient la liste ordonnée des nœuds pour les algorithmes.

La figure 5 présente un exemple d'un graphe et de l'ordre des nœuds obtenu en fonction de la procédure choisie.

2.3.2 FMC : Pattabiraman et al. [19] proposent quand à eux un algorithme exact et une heuristique pour le problème de la clique maximum appliqué aux grands graphes.

Les auteurs proposent des méthodes d'élagage lors du déroulement de l'algorithme. Ces élagages permettent d'éviter d'explorer certaines voies qui n'apporteraient pas de meilleures solutions. L'heuristique proposée par les auteurs diffère de l'algorithme standard dans le sens où elle ne teste pas chacun des nœuds de l'ensemble U mais prend celui avec le degré maximum. L'algorithme consiste cependant à explorer les noeuds dont le degré est supérieur à la taille de



lise 1 : 4 0 8 6 1 7 5 2 3

lise 2 : 0 1 8 2 4 3 5 6 7

lise 3 : 0 4 8 1 6 5 2 7 3

Fig. 5. Exemple d'ordonnancement de nœuds dans les dérivés de l'algorithme MC

la clique maximum trouvée (voir détails dans l'algorithme 6-1 et sa sous-fonction 6-2).

Algorithme 6-1 FMC()

```
1:  max ← 0
2:  for each v ∈ |E| do
3:     if deg(v) ≥ max then                ⇒ Elagage 1
4:       U ← ∅
5:       for each n ∈ Γ(v)
6:         if id(v) > id(n)                ⇒ Elagage 2
7:           ifdeg(n) ≥ max then           ⇒ Elagage 3
8:              U ← U + n
9:           end if
10:        end if
11:      end for
12:      CLIQUE(U,1,max)
13:    end if
14: end for
```

Algorithme 6-2 CLIQUE(U,$size$,max)

```
1:  if U = ∅ then
2:     if size > max then
3:       max ← size
4:       return
5:     end if
6:  end if
7:  while|U| > 0 do
8:     if (size + |U|) ≤ max : return                ⇒ Elagage 4
9:     tirer aléatoirement u ∈ U
10:     U ← U - u
11:    Γ'(u) = {w|w ∈ Γ(u) && deg(w) ≥ max}          ⇒ Elagage 5
12:    CLIQUE(U∪ Γ'(u),size+1,max)
13: end while
```

3 Problème de MCE/MCP sur un graphe compressé

Cette section porte notre contribution qui consiste à résoudre les problèmes de MCE et MCP sur un graphe compressé. Nous avons utilisé la décomposition modulaire pour compresser les graphes mais d'autres méthodes peuvent être envisagées. Nous commencerons par décrire l'algorithme que nous proposons pour le problème de MCE puis celui pour le MCP en utilisant un graphe compressé.

3.1 Énumération de cliques maximales sur un graphe compressé

L'algorithme que nous proposons est un algorithme récursif qui part de la racine de l'arbre de décomposition modulaire et qui l'explore en profondeur. Les cliques sont relevées de manières différentes en fonction du type du nœud parcouru.

- Si le nœud est une feuille on retourne uniquement l'identifiant correspondant au nœud dans le graphe initial.
- Si le nœud est de type série : étant donné que tous ses fils sont reliés entre eux (et forment donc un graphe complet), on retourne un ensemble de listes d'identifiants qui correspondent aux nœuds pouvant être compris dans la clique trouvée. La figure 6 illustre un exemple.
- Si le nœud est de type parallèle : étant donné qu'aucun de ses fils n'est relié avec un autre (le complémentaire est premier), on retourne un ensemble de listes d'identifiants où chaque liste est issue du retour des nœuds fils du module parallèle. La figure 7 donne un exemple.
- Si le nœud est de type premier, il est nécessaire de lancer une recherche de clique. On commence alors par construire le graphe compressé correspondant $\mathbf{Gc}$=($\mathbf{V}_c$,$\mathbf{E}_c$) à l'aide de ses nœuds fils $\mathbf{V}_c$ et de leurs arêtes $\mathbf{E}_c$ issues des listes de voisinages créés lors de la compression du graphe initial. On lance ensuite un algorithme de recherche de cliques maximales sur le graphe

compressé qui est basé sur Tomita. Ensuite, on parcourt les cliques trouvées de la manière suivante : on explore chacun des nœuds-module de la clique et on remplace leur identifiant de module par la liste de nœuds qu'ils retournent lors de leur appel de la fonction de recherche (qui dépend du type du module). La figure 8 montre un exemple pour ce cas.

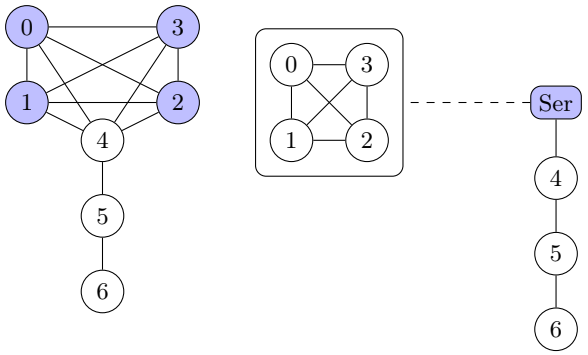


Fig. 6. Exemple du problème MCE avec un nœud série
Exemple avec un graphe simple, à gauche. La racine de l'arbre est un nœud premier dont il est possible de voir le graphe compressé correspondant à droite. Le nœud de type série, lorsqu'il est interrogé dans la fonction de recherche de cliques retourne la liste de ses nœuds fils (0,1,2 et 3). Ainsi les cliques trouvées par l'algorithme dans ce cas sont : {5,6}, {4,5} et {0,1,2,3,4}.

3.2 Recherche de clique maximum sur un graphe compressé

L'algorithme que nous proposons est également un algorithme récursif. Les tailles de cliques sont relevées de manières différentes en fonction du type du nœud parcouru.

- Si le nœud est du type nœud feuille on retourne la valeur 1.
- Si le nœud est de type série : on retourne la somme de chacun de ses nœuds fils. Un exemple est illustré dans la figure 9.
- Si le nœud est de type parallèle : on retourne la valeur maximum issue de chacun de ses nœuds fils. Un exemple est illustré dans la figure 10.
- Si le nœud est de type premier : on retourne la taille de la clique maximum du graphe premier que nous donne l'algorithme FMC que nous avons adapté dans la partie du graphe compressé. La figure 11 présente un exemple pour ce cas.

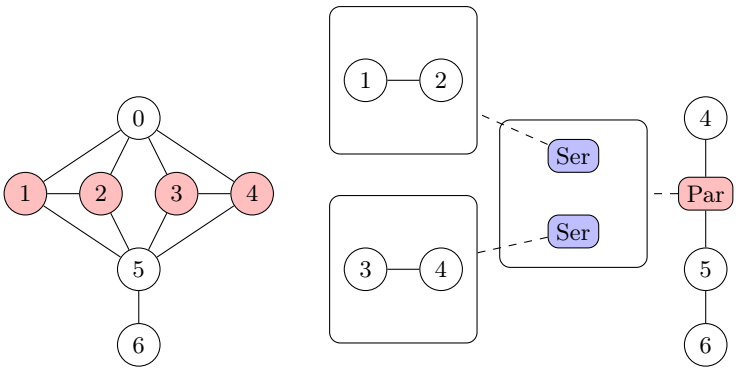


Fig. 7. Exemple du problème MCE avec un nœud parallèle
Exemple avec un graphe simple, à gauche. La racine de l'arbre est un nœud premier dont il est possible de voir le graphe compressé correspondant à droite. Le nœud de type parallèle, lorsqu'il est interrogé dans la fonction de recherche de cliques retourne un ensemble de listes issues des listes de retour de ses nœuds fils, dans ce cas il retourne ({1,2},{3,4}). Ainsi les cliques trouvées par l'algorithme dans ce cas sont : {5,6}, {0,1,2},{0,3,4},{5,1,2} et {5,3,4}.

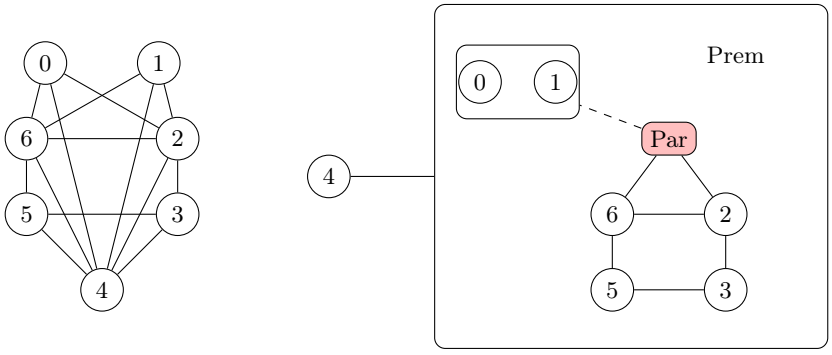


Fig. 8. Exemple du problème MCE avec un nœud premier
Exemple avec un graphe simple, à gauche. La racine de l'arbre est un nœud série qui relie le nœud 4 à un nœud premier dont on peut observer le graphe compressé correspondant à droite. Notre algorithme va donc lancer une recherche de clique maximale dans le graphe compressé et retourner une liste de cliques issue de sa recherche, à savoir ({0,2,6},{1,2,6},{2,3},{3,5},{5,6}). Ainsi les cliques trouvées par l'algorithme dans ce cas sont : {0,2,4,6},{1,2,4,6},{2,3,4},{3,4,5} et {4,5,6}.

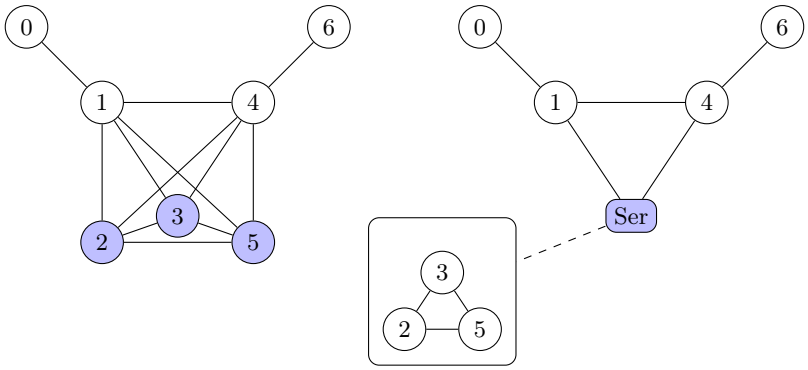


Fig. 9. Exemple du problème MCP avec un nœud série
Exemple avec un graphe simple, à gauche. La racine de l'arbre est un nœud premier dont il est possible de voir le graphe compressé correspondant à droite. Le nœud de type série, lorsqu'il est interrogé dans la fonction de recherche de clique maximale retourne 3 (la somme de ses nœuds fils, qui sont 3 nœuds feuilles, retournant chacun 1). Ainsi la clique maximum de ce graphe compressé est de taille 5 : la clique est composée des nœuds 1, 4 et Ser, qui vaut 3.

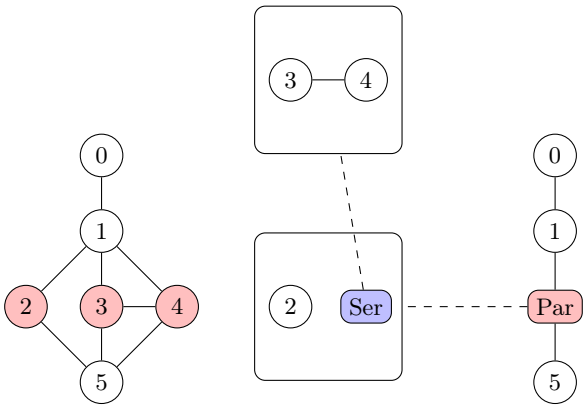


Fig. 10. Exemple du problème MCP avec un nœud parallèle
Exemple avec un graphe simple, à gauche. La racine de l'arbre est un nœud premier dont il est possible de voir le graphe compressé correspondant à droite. Le nœud de type parallèle, lorsqu'il est interrogé dans la fonction de recherche de clique maximale retourne 2 (la fonction prend le maximum entre le nœud feuille 2, qui vaut 1 et le nœud série composé des nœuds 3 et 4, qui vaut donc 2). Ainsi la clique maximum de ce graphe compressé est de taille 3 : la clique est composée des nœuds 1 ou 5 ainsi que de Par qui vaut 2 et qui se reporte aux nœuds 3 et 4.

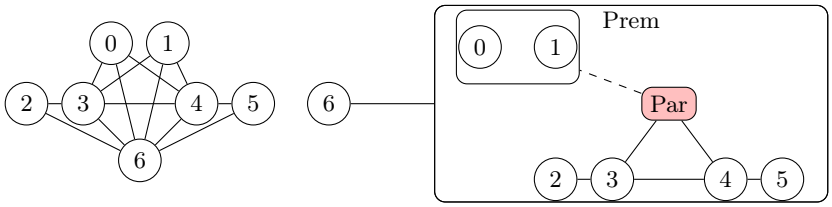


Fig. 11. Exemple du problème MCP avec un nœud premier
Exemple avec un graphe simple, à gauche. La racine de l'arbre est un nœud série qui relie le nœud 6 à un nœud premier dont on peut observer le graphe compressé correspondant à droite. Notre algorithme va donc lancer une recherche de clique maximale dans le graphe compressé et retourner une clique de taille 3 (3,4 et Par, qui retourne 1).

3.3 Optimisation

Afin d'améliorer la vitesse des algorithmes, nous avons ajouté une optimisation. Cette optimisation permet à un nœud-module dont le type est différent du module feuille et dont le père est de type premier de sauvegarder la valeur qu'il a renvoyée au premier appel. Cela permet de ne pas avoir à parcourir à nouveau tout son sous-arbre s'il fait à nouveau appel à ce nœud. En effet si le père est de type premier, il est possible que ce nœud se retrouve dans plusieurs cliques et qu'il soit donc appelé plusieurs fois. Par exemple, dans le cas du problème MCE, si un nœud-module est lui-même premier et qu'il se trouve dans deux cliques différentes lorsque l'on évalue les cliques dans le graphe compressé de son père, on va appeler deux fois sa fonction de retour. Or, lors du second appel on va devoir à nouveau reconstruire un graphe, lancer la recherche de clique et reconstruire la liste de retour, ce qui peut être coûteux en temps. Nous choisissons donc d'enregistrer la liste de retour (dans le cadre du MCE) ou la taille maximum du module (pour le MCP) et de retourner cette valeur en cas de second appel.

3.4 Exemples illustratifs

Dans cette section nous allons détailler deux exemples, un pour chacun des problèmes, afin d'illustrer nos algorithmes.

3.4.1 Exemple pour le problème MCE

Nous allons commencer par le problème de l'énumération de cliques. Nous travaillerons sur le graphe et sa version compressée présentés en figure 12.

Le résultat de l'algorithme de compression pour ce graphe est présenté sur la figure 13. Nous reviendrons sur le taux de compression et le temps d'exécution dans la section 4.

Notre algorithme d'énumération de cliques maximales va regarder le type de la racine, le nœud module 0. Ce dernier est de type premier. Dans ce cas,

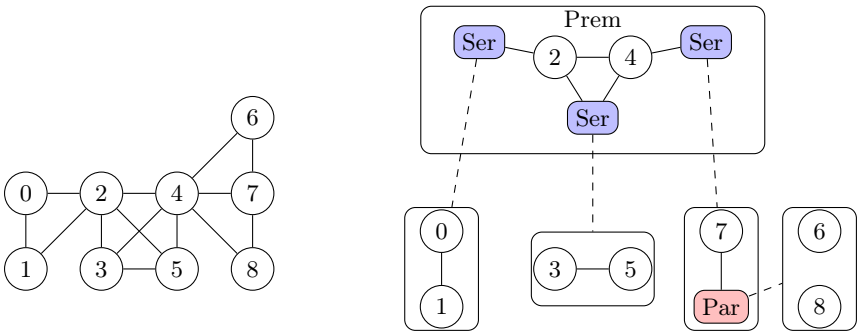


Fig. 12. Graphe exemple pour la décomposition modulaire

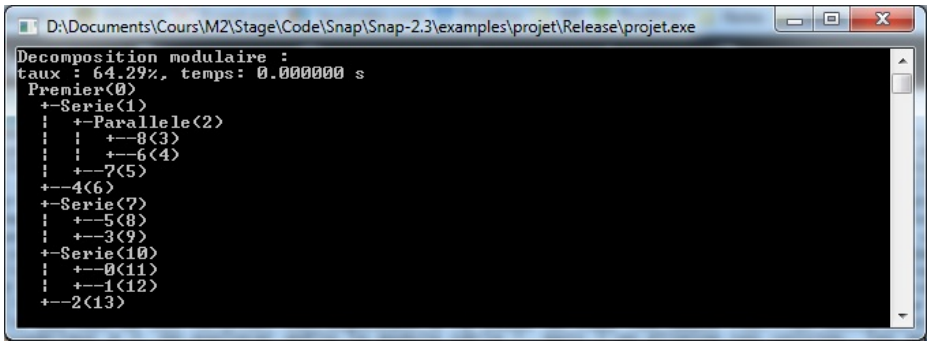


Fig. 13. Résultat de compression, les identifiants des modules sont entre parenthèses.

notre algorithme va reconstruire un graphe à l'aide des relations de voisinage entre les nœuds modules descendants du nœud premier, à savoir ceux dont les identifiants sont 1,6,7,10 et 13.

Nous allons alors lancer notre algorithme de recherche de cliques sur ce graphe. Nous avons donc un ensemble de candidat qui est constitué des nœuds modules {1,6,7,10,13}. Le premier candidat de clique sélectionné est le numéro 6 car c'est celui avec le plus petit indice parmi ceux qui ont le degré maximum. Nous mettons alors à jour notre liste de candidats qui contient les nœuds modules {1,7,13}. Le nouveau candidat choisi est alors 7, il ne reste plus qu'un candidat, le nœud module 13 car le nœud module 1 n'est pas voisin avec 7. Une première clique est ainsi identifiée : elle est composée des nœuds modules {6,7,13}. On revient alors au moment où nous n'avions que 6 comme candidat et l'on trouve une autre clique avec le nœud module 1. On revient alors à l'origine où l'on identifie la dernière clique {10,13}. Nous avons donc identifié 3 cliques dans le graphe compressé : {6,7,13},{1,6} et {10,13}.

Dans la deuxième partie de notre algorithme, nous explorons en profondeur ces cliques pour remplacer les identifiants des noeuds modules par les noeuds du graphe original qui sont identifiés par leurs descendants feuilles. Ainsi les cliques {6,7,13} et {10,13} trouvées dans le graphe compressé deviennent respectivement les cliques {2,3,4,5} et {0,1,2}. Pour la clique {1,6} on remplace l'identifiant du module feuille 6 par celui de son descendant : 4. Pour le module série 1, on regarde ses descendants : les nœud feuille 5 et le nœud parallèle 2. Le nœud parallèle retourne une liste d'ensemble à savoir {{6},{8}}. Il est alors nécessaire, pour ses parents de créer également des listes d'ensemble pour identifier les différentes cliques. Ainsi le module série 1 renvoie la liste d'ensemble {{7,6},{7,8}} et les cliques identifiées à l'aide de la clique du graphe compressé {1,6} sont {4,6,7} et {4,7,8}.

Les cliques identifiées par notre algorithme sont alors les cliques {0,1,2}, {2,3,4,5}, {4,6,7} et {4,7,8}.

3.4.2 Exemple de problème MCP

Dans cet exemple nous allons prendre le graphe présenté dans la figure 14 dont le graphe compressé (avec l'imbrication de ses modules) est présenté sur la figure 15.

Notre algorithme de recherche de clique maximum regarde le type du module racine, qui est parallèle. Il va donc appeler chacun de ses fils, les modules premiers 1 puis 10, qui vont lui retourner une clique de taille maximum et comparer les tailles des cliques pour choisir la plus grande.

Le nœud module 1 est premier, on doit donc tenter de déterminer la taille de sa plus grande clique. On doit trouver parmi les modules fils du module 1 la plus grande clique. L'algorithme va donc prendre le premier module fils, qui est

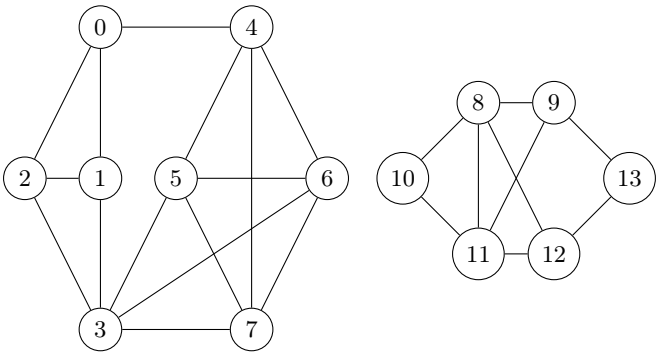


Fig. 14. Graphe exemple pour la décomposition modulaire

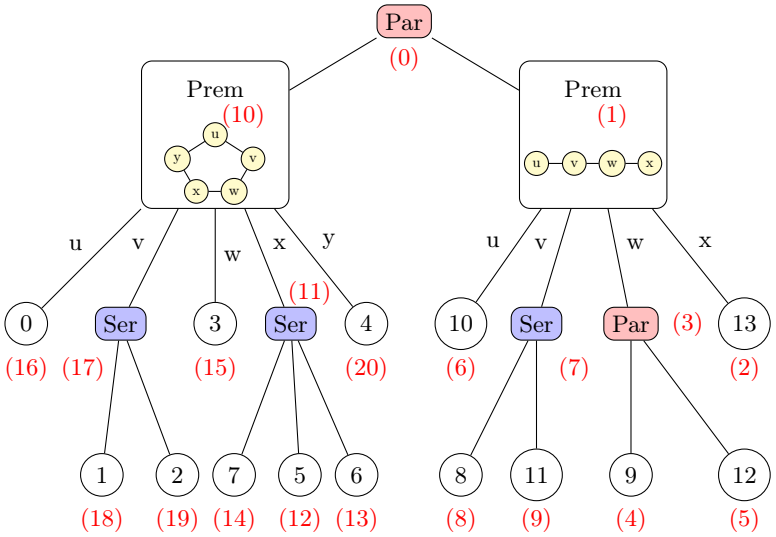


Fig. 15. Graphe compressé avec détail des modules

le module feuille 2 et qui correspond au nœud 13. En regardant le voisinage du module 2, on se rend compte qu'il n'y a que le module parallèle 3, qui si l'on fait appel à sa fonction de clique maximum, va retourner 1 correspondant au module 4, premier module fils, qui identifie le nœud 9. À ce moment-là, notre clique de taille maximum est donc une clique de taille 2 qui correspond à la clique {13,9}. Toujours dans le module 1, on va donc passer à son fils suivant, le module parallèle 3. On va éliminer le module 2 de la liste de voisinage intéressante car son identifiant de module (2) est inférieur à celui actuellement testé (3) et a donc déjà été visité. Il ne reste donc que le module série 7 dans le voisinage. Le module série 7 retourne une clique de taille 2, correspondant à ses deux modules fils : les feuilles 8 et 9, qui identifient les nœuds 8 et 11. Dès cet instant la clique maximum trouvée est une clique de taille 3 correspondant à la clique {8,9,11}. On regarde ensuite le module feuille 6, qu'on élimine tout de suite car sa taille plus la taille maximum de ses voisins, correspondant à la somme de tous les nœuds feuilles descendants de ses voisins, n'est pas supérieure à 3. On explore enfin le module série 7, mais l'on élimine ses voisins, les modules 3 et 6, car leurs identifiants sont inférieurs au sien. La recherche de clique maximum dans le module premier 1 et son graphe compressé est donc terminée, l'algorithme retourne une clique de taille 3 : {8,9,11}.

Le nœud racine 0 demande alors au nœud premier 10 sa clique de taille maximum. Ce dernier va lui retourner une clique de taille 4 : {3,5,6,7}. Cette clique étant plus grande, la clique de taille maximum retournée par le programme sera donc cette dernière.

4 Évaluation

Dans cette section nous allons décrire les expérimentations réalisées afin d'évaluer notre approche. Pour cela nous avons implémenté nos algorithmes ainsi que tous les algorithmes que nous avons présentés dans la partie état de l'art. Nous avons implémenté tous les algorithmes en C++ à l'aide la libraire SNAP [20]. Ceci afin d'utiliser les mêmes structures pour l'ensemble des algorithmes. Nous avons travaillé sur une machine 64 bits avec un système d'exploitation Ubuntu v14.04 LTS, un processeur i5-4690 de fréquence 3.5GHz et une mémoire vive de 8G.

Les algorithmes ont été compilés avec la version 4.8.2 du compilateur g++ et les options -std=c++11 et l'optimisation -03.

Notre algorithme de compression est basé sur l'algorithme de décomposition modulaire de Fabien de Montgolfier [21] qui a une complexité en O(m log n). L'algorithme de décomposition est lui-même issu de la concaténation de deux algorithmes distincts :

1. Le premier réalise une permutation factorisante des sommets [22] du graphe [23] grâce à une technique d'affinage des partitions [24].
2. La seconde construit l'arbre de décomposition modulaire à l'aide la permutation issue du premier algorithme [25].

De plus, afin de permettre aux algorithmes de recherche de cliques de pouvoir fonctionner, nous avons ajouté un troisième algorithme qui ajoute aux nœuds fils d'un graphe premier une liste de voisinage. Concrètement, cela signifie que si un nœud est voisin avec un autre nœud dans le graphe initial, les modules les représentants dans un graphe premier auront également une arête de voisinage. L'algorithme est un algorithme récursif qui se contente de parcourir l'arbre en ajoutant une arête lorsqu'elle était existante dans le graphe précédent. Il teste chacun des descendants d'un nœud premier pour savoir si il existe une arête entre les nœuds qu'ils représentent dans le graphe original et ajoute une arête entre les descendants le cas échéant. L'algorithme 7 présente le détail de cette opération.

Algorithme 7 creeVoisins(Noeud n)

```
1:  if n est de type premier
2:      for i allant de 0 à nombre de descendant
3:        for j allant de i à nombre de descendant
4:          if i est voisin de j dans le graphe
5:            Ajouter i à la liste de voisinage de j dans le graphe compressé
6:            Ajouter j à la liste de voisinage de i dans le graphe compressé
7:          end if
8:        end for
9:      end for
10: end if
11: if n a des descendants premiers
12:     for each s descendant de n
13:       creeVoisins(s)
14:     end for
14: end if
```

Afin d'évaluer notre approche, nous avons implémenté différents algorithmes et heuristiques pour les deux problèmes.

Pour le problème de l'énumération de cliques nous avons implémenté les algorithmes suivants :

- MCE : c'est l'algorithme témoin, celui de base, il ne fait pas une sélection intéressante contrairement à Tomita sur le choix du pivot, qui est sélectionné aléatoirement pour MCE. Il repose sur l'article de Bron et Kerbosh [2].
- Tomita [3] : c'est l'algorithme de référence, sa complexité est de l'ordre de $O(3^{(n/3)})$. Il donne généralement de bons résultats et c'est sur lui que repose notre algorithme de recherche de clique dans notre graphe compressé.
- Eppstein [16]

– RMC [18]

Pour le problème de la clique maximum nous avons implémenté les algorithmes suivants :

– MC et ses variantes notés MCQ, MCSa, MCSb et BBMC.
– FMC [19] et son heuristique. C'est lui notre algorithme de référence, notre algorithme de recherche de clique maximum dans le graphe compressé utilise ses mécaniques.

Plus spécifiquement pour le problème de la clique maximum nous avons implémenté deux heuristiques qui n'évaluent pas l'ensemble des solutions mais visent à trouver une réponse acceptable le plus vite possible. La première ne teste pas l'ensemble des modules du voisinage d'un nœud candidat mais uniquement celui dont le nombre de ses descendants feuilles plus le nombre de descendants feuilles de ses voisins est le maximum. La seconde ne sélectionne quant à elle que le voisin dont le nombre de descendants feuilles plus le nombre de modules voisins est maximum.

4.1 Graphes de test

Le tableau 2 présente les graphes sur lesquels nous avons testé les algorithmes. Ces graphes sont tirés de la base de données SNAP[26], de celle des graphes Dimacs[27] ainsi que du projet RI[28].

ID	**Nom**	**|V|**	**|E|**	densité	d_{Min}	d_{Max}	d_{Moy}	Taille
(1)	16pk	4919	4972	0.000411	1	4	2.021	51.2Ko
(2)	3djd	11862	12010	0.000171	1	4	2.025	131Ko
(3)	c-fat500-5	500	23191	0.185900	92	95	92.764	194Ko
(4)	6pfk	17123	17263	0.000118	1	4	2.016	198Ko
(5)	Caenorhabditis_elegans	6173	26184	0.001374	1	418	8.483	273Ko
(6)	Takifugu_rubipres	5872	27077	0.001571	1	162	9.222	281Ko
(7)	3dmk	30386	30773	0.000067	1	4	2.025	369Ko
(8)	c-fat500-10	500	46627	0.373764	185	188	186.508	390Ko
(9)	Drosophila_melanogaster	8624	39466	0.001061	1	311	9.153	413Ko
(10)	Rattus_norvegicus	8763	39932	0.001040	1	334	9.114	419Ko

Table 2. Présentation des graphes de tests

Pour chaque graphe nous donnons dans le tableau 3 le temps nécessaire pour le compresser ainsi que la taille du graphe compressé (le nombre de modules), le nombre total d'arêtes dans l'ensemble des nœuds premiers de l'arbre et enfin ce que nous avons appelé le taux de compression qui correspond au rapport nombre d'arêtes dans le graphe compressé sur le nombre d'arêtes dans le graphe initial.

Nom	nombre modules	nombre d'arêtes	temps	taux
16pk	5532	4199	0.112408	15.547064
3djd	13488	9976	0.118331	16.935887
c-fat500-5	517	16	0.012088	99.931008
6pfk	19467	14226	0.412102	17.592539
Caenorhabditis_elegans	6656	24570	0.252191	6.164070
Takifugu_rubipres	6651	23530	0.248460	13.099679
3dmk	34537	25468	1.621132	17.239138
c-fat500-10	509	8	0.017551	99.982843
Drosophila_melanogaster	9137	38297	0.656012	2.962043
Rattus_norvegicus	9517	37279	0.611329	6.643794

Table 3. Présentation de la compression des graphes

4.2 Résultats et discussions

Nous avons lancé la totalité de nos algorithmes sur l'ensemble des graphes. Les tableaux 4 et 5 présentent les résultats.

Algorithme/Graphe	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
MCE nombre	4972	12010	16	17263	19386	18877	30773	8	30613	41118
MCE temps	0.191345	1.332851	0.187388	3.299488	2.392071	2.451714	7.331336	0.596337	3.782695	4.775979
Tomita nombre	4972	12010	16	17263	19386	18877	30773	8	30613	41118
Tomita temps	0.527728	3.056610	0.067527	6.455378	1.242608	1.305117	19.944410	0.203265	2.596694	2.758777
Eppstein nombre	4972	12010	16	17263	19386	18877	30773	8	30613	41118
Eppstein temps	0.819774	5.020200	0.061227	11.018320	1.708660	1.564697	37.313217	0.166300	3.069022	3.671179
Eppstein-hybrid nombre	4972	12010	16	17263	19386	18877	30773	8	30613	41118
Eppstein-hybrid temps	0.819760	5.021373	0.061290	11.017769	1.706829	1.577936	37.303516	0.165351	3.076410	3.677518
RMC(0.5) nombre	4968	12006	13	17259	13432	13701	30769	5	23700	22536
RMC(0.5) temps	0.011404	0.029687	0.249176	0.042841	0.320944	0.301336	0.077673	0.583173	0.454680	0.630219
RMC-random(0.5) nombre	2028	4447	5	6754	8620	8818	12254	2	16476	13868
RMC-random(0.5) temps	0.007758	0.020021	0.224963	0.029426	0.257120	0.231431	0.053029	0.615800	0.373287	0.476301
RMC(0.7) nombre	4968	12006	14	17259	14102	14449	30769	6	24663	28098
RMC(0.7) temps	0.011545	0.030304	0.251984	0.042500	0.344738	0.324195	0.076215	0.523292	0.486889	0.785311
RMC-random(0.7) nombre	3209	7435	7	10778	10789	11020	19422	4	19755	19912
RMC-random(0.7) temps	0.009345	0.024908	0.246753	0.035217	0.286739	0.278023	0.063146	0.586587	0.420592	0.623904
RMC(0.9) nombre	4968	12006	14	17259	15213	15620	30769	6	26588	38843
RMC(0.9) temps	0.010959	0.030049	0.259891	0.043244	0.370070	0.351391	0.076529	0.527128	0.535801	0.925838
RMC-random(0.9) nombre	4351	10459	11	15108	13579	13977	26945	6	24106	32121
RMC-random(0.9) temps	0.010720	0.029037	0.241220	0.041068	0.348385	0.331754	0.073176	0.526113	0.508659	0.827991
RMC(1.0) nombre	4968	12006	14	17259	15370	15836	30769	6	27120	39223
RMC(1.0) temps	0.011493	0.031217	0.292904	0.044416	0.373217	0.357608	0.080293	0.764414	0.546714	0.940165
RMC-random(1.0) nombre	4968	12006	14	17259	15370	15836	30769	6	27120	39223
RMC-random(1.0) temps	0.011092	0.030558	0.259119	0.043744	0.374400	0.355942	0.079295	0.527740	0.546354	0.930474
AlgorithmeCompresse nombre	4972	12010	16	17263	19386	18877	30773	8	30613	41118
AlgorithmeCompresse temps	0.419758	0.259966	0.000204	1.289428	1.412488	1.292600	5.646112	0.000253	3.448094	4.411429

Table 4. Résultats pour le problème d'énumération de cliques

Le graphique 16 présente les différents temps de calcul. Par soucis de clarté, nous présentons les courbes dans différents graphiques. Les deux graphiques du haut représentent les résultats des algorithmes présentés dans la section état de l'art pour les problèmes MCE et MCP. Pour les algorithmes qui avaient différentes options tels que RMC, MCQ, MCS ou BBMC, les résultats présentés sont ceux où l'algorithme a réalisé le plus petit temps d'exécution. Les deux graphiques du milieu représentent les différents temps de calcul entre Tomita (le meilleur algorithme exact pour le problème MCE) et notre

Algorithme/Graphe	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)
MC nombre	2	2	64	2	45	43	2	(*)	45	23
MC temps	0.18644	1.32221	0.250830	3.26847	0.58341	0.55342	7.32325	(*)	1.35264	1.13382
MCQ-style1 nombre	2	2	64	2	45	43	2	126	45	23
MCQ-style1 temps	0.21738	1.59742	0.13809	3.16210	0.21565	0.15468	7.66038	2.99407	0.27436	0.26241
MCQ-style2 nombre	2	2	64	2	45	43	2	126	45	23
MCQ-style2 temps	0.20930	1.47573	0.12624	2.10227	0.14749	0.14038	6.38786	1.38440	0.25421	0.23719
MCQ-style3 nombre	2	2	64	2	45	43	2	126	45	23
MCQ-style3 temps	0.22299	1.65145	0.13833	3.07288	0.20400	0.15048	8.36420	2.99163	0.28595	0.26792
MCSa-style1 nombre	2	2	64	2	45	43	2	126	45	23
MCSa-style1 temps	0.34137	2.65295	0.13831	5.14018	0.21959	0.15456	11.95276	2.98490	0.27384	0.26222
MCSa-style2 nombre	2	2	64	2	45	43	2	126	45	23
MCSa-style2 temps	0.31601	2.33061	0.15859	3.11691	0.14866	0.13979	9.39508	1.82358	0.25300	0.23907
MCSa-style3 nombre	2	2	64	2	45	43	2	126	45	23
MCSa-style3 temps	0.348134	2.73486	0.13832	5.09251	0.21525	0.15150	13.16387	2.98704	0.283592	0.267109
MCSb-style1 nombre	2	2	64	2	27	17	2	126	45	14
MCSb-style1 temps	0.343416	2.666885	0.051836	5.178642	0.150885	0.138746	11.976229	0.320805	0.26783	0.33338
MCSb-style2 nombre	2	2	62	2	25	10	2	124	13	13
MCSb-style2 temps	0.31493	2.33532	0.04351	3.12379	0.11763	0.14532	9.38984	0.27898	0.22119	0.23865
MCSb-style3 nombre	2	2	64	2	34	25	2	126	38	15
MCSb-style3 temps	0.34467	2.73418	0.05216	5.09255	0.15943	0.13875	13.17318	0.32056	0.27309	0.30702
BBMC-style1 nombre	2	2	64	2	45	43	2	126	45	23
BBMC-style1 temps	1.39118	8.70469	0.05467	19.79411	2.82950	2.51877	52.64940	0.18899	5.65066	5.89882
BBMC-style2 nombre	2	2	64	2	45	43	2	126	45	23
BBMC-style2 temps	1.38961	8.58911	0.05661	19.71385	2.80261	2.52039	52.65193	0.16055	5.65604	5.66581
BBMC-style3 nombre	2	2	64	2	45	43	2	126	45	23
BBMC-style3 temps	1.40044	8.82759	0.06462	20.14705	3.08633	2.52640	53.26737	0.18763	5.66387	5.92335
FMC nombre	2	2	64	2	45	43	2	(*)	45	23
FMC temps	0.00202	0.00611	22.60908	0.00783	0.91282	1.15473	0.08029	(*)	0.26029	0.935434
FMC-heuristique nombre	2	2	64	2	45	43	2	(*)	45	23
FMC-heuristique temps	0.00213	0.24167	0.25912	0.02407	0.03437	0.35594	0.01604	(*)	0.04077	0.05916
AlgorithmeCompresse nombre	2	2	64	2	45	43	2	126	45	23
AlgorithmeCompresse temps	0.09835	0.04139	0.00003	0.27847	211.4017	157.6894	1.47763	0.00005	630.6908	404.0871
HeuristiqueCompresse1 nombre	2	2	64	2	45	43	2	126	45	23
HeuristiqueCompresse1 temps	0.10490	0.04612	0.00004	0.30415	9.57522	7.76619	5.64611	0.00006	20.74985	21.65489
HeuristiqueCompresse2 nombre	2	2	63	2	44	30	2	125	12	22
HeuristiqueCompresse2 temps	0.09349	0.03792	0.00001	0.25318	8.10292	5.96132	5.64611	0.00003	19.07009	17.52005

Table 5. Résultats pour le problème de la clique maximum

algorithme pour le problème MCE. Les graphiques présentés sont ceux dont les graphes ont un taux de compression de l'ordre de 0 à 20% pour le graphe de gauche et de l'ordre de 99% pour le graphe de droite. Nous avons également rajouté une courbe représentant le temps de notre algorithme plus le temps de compression du graphe. Les deux graphiques du bas représentent les temps de calculs de FMC et de notre algorithme pour le problème MCP. Les graphiques présentés sont ceux dont les graphes ont un taux de compression de l'ordre de 0 à 20% pour le graphe de gauche et de l'ordre de 99% pour le graphe de droite.

Pour le problème d'énumération de cliques, même en ajoutant le temps de compression de graphe, on remarque que notre algorithme fait mieux que Tomita, notre algorithme de référence, dans les cas où le graphe a un taux de compression supérieur à 15%. Les autres algorithmes tels que MCE et RMC donnent certaines fois de meilleurs résultats en terme de temps, c'est pourquoi il serait intéressant de les adapter sur le graphe compressé.

Pour le problème de la clique Maximum, notre algorithme ne fait jamais mieux que FMC (sauf pour les graphes qui sont compressés à 99%). Nous expliquons cela par le fait que la compression de graphe fait perdre le gain de certains élagages. En effet, dans le graphe compressé le degré du voisinage d'un module n'est pas suffisant pour éliminer un module car ce module peut représenter de nombreux nœuds dans le graphe compressé et doit donc être pris en compte. Le fait de prendre en compte l'ensemble des nœuds modules a

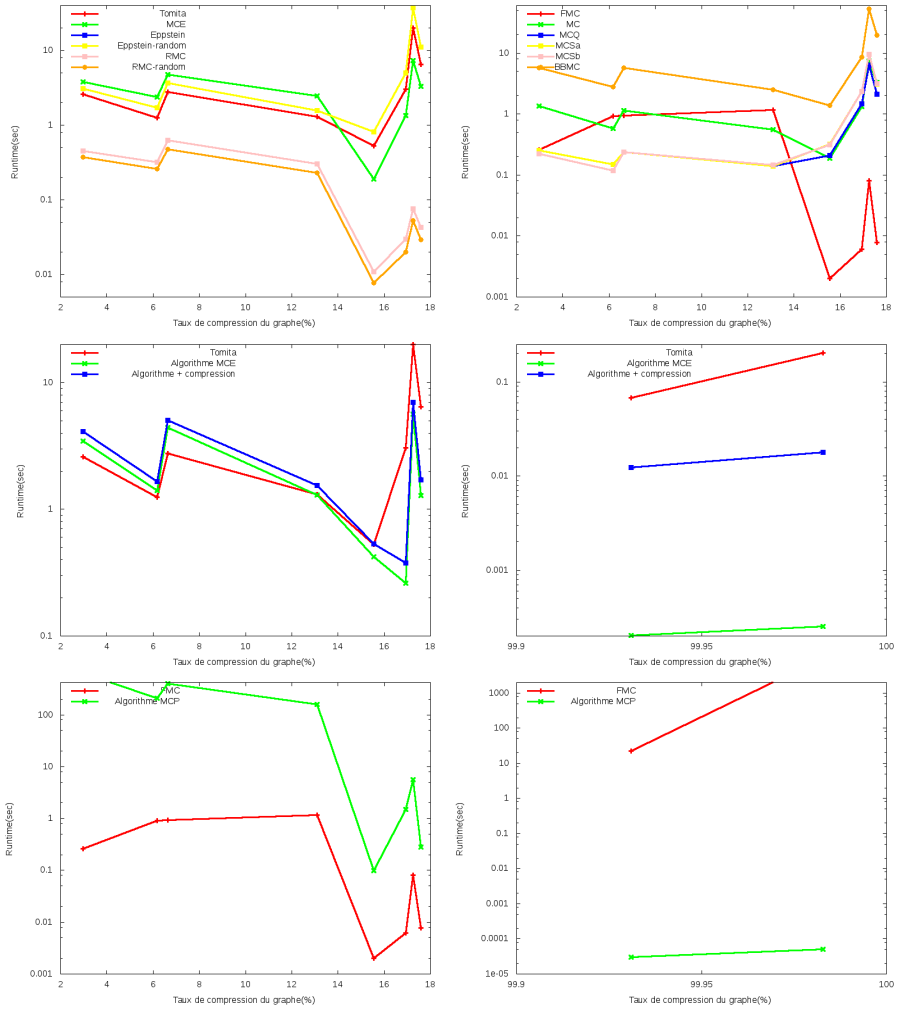


Fig. 16. Graphiques montrant les temps de calculs des algorithmes en fonction du taux de compression des graphes, l'échelle du temps est logarithmique.

donc un coup trop important pour permettre un gain de temps. L'adaptation de FMC dans le graphe compressé ne nous paraît donc pas intéressant.

En revanche il pourrait être intéressant, lors de la compression du graphe, de directement mettre un poids aux modules ou aux arêtes, qui représenteraient le nombre de nœuds maximum pour une clique dans le module. Ceci permettrait de créer un algorithme qui se serve du poids pour trouver la taille de la clique maximum du graphe.

4.3 Étude de la complexité

Nous conjecturons que la complexité de l'algorithme d'énumération de cliques est de l'ordre de $\sum_{i=1}^{ms} n_i + \sum_{j=1}^{mp} n_j + \sum_{k=1}^{mr} 3^{n_k/3} + n$ où ms, mp et mr représentent respectivement le nombre de modules séries, parallèles et premiers et n_i,n_j et n_k représentent respectivement le nombre de nœuds fils de chacun d'entre eux. n est le nombre de nœuds dans le graphe initial.

5 Conclusion et ouvertures

Les problèmes d'énumération de cliques maximales et de recherche de cliques maximums dans les graphes sont des problèmes complexes et difficiles. Ces problèmes deviennent d'autant plus difficiles lorsque les graphes sur lesquels nous travaillons sont grands. Pour résoudre ces problèmes il existe de nombreux algorithmes et heuristiques, plus ou moins adaptés aux grands graphes. Dans ce papier nous avons proposé et évalué une nouvelle méthode pour résoudre ces problèmes.

Notre algorithme se déroule en deux phases. La première consiste à compresser le graphe, via la décomposition modulaire pour diminuer la taille de celui-ci. La décomposition modulaire est une méthode de compression qui nous permet de garder les informations nécessaires à la recherche de cliques. La deuxième phase consiste à chercher les cliques sur le graphe compressé

L'expérimentation empirique réalisée démontre que notre algorithme pour le problème de la clique maximum n'est pas suffisamment efficace. Cependant notre algorithme d'énumération de cliques montre de meilleures performances dès que la compression du graphe est supérieure à 15%.

Il nous semble intéressant de continuer le travail commencé en poursuivant plusieurs options :

- Augmenter l'efficacité des algorithmes en ajoutant des informations lors de la compression de graphes, notamment sur la taille des cliques dans les modules séries et parallèles.

– Développer de nouveaux algorithmes sur les graphes compressés.
– Regarder la consommation de mémoire vive de l'algorithme.
– Adapter la méthode aux autres types de graphes tels que les graphes orientés.
– Proposer une méthode de compression et de recherche de cliques en parallèle, l'algorithme de recherche de cliques démarrant dès qu'une compression du graphe est suffisante.

References

1. N. Przulj, "Graph theory approaches to protein interaction data analysis," *proteins*, vol. 120, p. 000, 2003.
2. C. Bron and J. Kerbosch, "Algorithm 457: finding all cliques of an undirected graph," *Communications of the ACM*, vol. 16, no. 9, pp. 575–577, 1973.
3. E. Tomita, A. Tanaka, and H. Takahashi, "The worst-case time complexity for generating all maximal cliques and computational experiments," *Theoretical Computer Science*, vol. 363, no. 1, pp. 28–42, 2006.
4. P. Prosser, "Exact algorithms for maximum clique: A computational study," *Algorithms*, vol. 5, no. 4, pp. 545–587, 2012.
5. R. M. Karp, *Reducibility among combinatorial problems*. Springer, 1972.
6. R. Samudrala and J. Moult, "A graph-theoretic algorithm for comparative modeling of protein structure," *Journal of molecular biology*, vol. 279, no. 1, pp. 287–302, 1998.
7. Z. Prihar, "Topological properties of telecommunication networks," *Proceedings of the IRE*, vol. 44, no. 7, pp. 927–933, 1956.
8. F. Zhou, "Graph compression," *Department of Computer Science and Helsinki Institute for Information Technology HIIT*, pp. 1–12.
9. M. Adler and M. Mitzenmacher, "Towards compressing web graphs," in *Data Compression Conference, 2001. Proceedings. DCC 2001.* IEEE, 2001, pp. 203–212.
10. Y. Tian, R. A. Hankins, and J. M. Patel, "Efficient aggregation for graph summarization," in *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. ACM, 2008, pp. 567–580.
11. P. Boldi and S. Vigna, "The webgraph framework i: compression techniques," in *Proceedings of the 13th international conference on World Wide Web*. ACM, 2004, pp. 595–602.
12. H. Toivonen, F. Zhou, A. Hartikainen, and A. Hinkka, "Compression of weighted graphs," in *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2011, pp. 965–973.
13. T. Gallai, "Transitiv orientierbare graphen," *Acta Mathematica Hungarica*, vol. 18, no. 1, pp. 25–66, 1967.
14. R. M. McConnell and J. P. Spinrad, "Modular decomposition and transitive orientation," *Discrete Mathematics*, vol. 201, no. 1, pp. 189–241, 1999.
15. M. Habib and C. Paul, "A survey of the algorithmic aspects of modular decomposition," *Computer Science Review*, vol. 4, no. 1, pp. 41–59, 2010.
16. D. Eppstein and D. Strash, "Listing all maximal cliques in large sparse real-world graphs," in *Experimental Algorithms*. Springer, 2011, pp. 364–375.
17. V. Batagelj and M. Zaversnik, "An o (m) algorithm for cores decomposition of networks," *arXiv preprint cs/0310049*, 2003.
18. J. Wang, J. Cheng, and A. W.-C. Fu, "Redundancy-aware maximal cliques," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 122–130.

19. B. Pattabiraman, M. M. A. Patwary, A. H. Gebremedhin, W.-k. Liao, and A. Choudhary, "Fast algorithms for the maximum clique problem on massive sparse graphs," in *Algorithms and Models for the Web Graph*. Springer, 2013, pp. 156–169.
20. "Site du stanford network analysis project, qui propose une librairie c++ pour la manipulation de graphes," https://snap.stanford.edu/snap-2.3/, accédé le : 05-02-2015.
21. "Site de fabien de montgolfier," http://www.liafa.jussieu.fr/ fm/algos/index.html, accédé le : 24-04-2015.
22. C. Capelle, M. Habib, and F. De Montgolfier, "Graph decompositions and factorizing permutations," *Discrete Mathematics and Theoretical Computer Science*, vol. 5, no. 1, pp. 55–70, 2002.
23. C. Capelle, "Decompositions de graphes et permutations factorisantes," Ph.D. dissertation, 1997.
24. M. Habib, C. Paul, and L. Viennot, "Partition refinement techniques: An interesting algorithmic tool kit," *International Journal of Foundations of Computer Science*, vol. 10, no. 02, pp. 147–170, 1999.
25. A. Bergeron, C. Chauve, F. De Montgolfier, and M. Raffinot, "Computing common intervals of k permutations, with applications to modular decomposition of graphs," *SIAM Journal on Discrete Mathematics*, vol. 22, no. 3, pp. 1022–1039, 2008.
26. "Base de donnée de grands graphes du stanford network analysis project," http://snap.stanford.edu/data/, accédé le : 05-02-2015.
27. "Site du dimacs challenge," http://dimacs.rutgers.edu/Challenges/, accédé le : 05-02-2015.
28. "Base de donnée de grands graphes du projet ri," http://ferrolab.dmi.unict.it/ri/datasets.html, accédé le : 17-04-2015.

6 Annexe

Algorithme 5-1 Procedure-Style1()

```
1: while V != ∅ do
2:     if |C|+|P| < max : return
3:     choisir n le premier élément de V
4:     j ← 0
5:     for each i ∈ V
6:         if deg(i) < deg(n) || (deg(i) == deg(n) && i > n)
7:             n ← i
8:         end if
9:     end for
10:    ColOrd ← ColOrd + n
11:    V ← V − n
12: end while
```

Algorithme 5-2 Procedure-Style2()

```
1: while V != ∅ do
2:     if |C|+|P| < max : return
3:     choisir n le premier élément de V
4:     j ← 0
5:     for each i ∈ V
6:         if deg(i) < deg(n)
7:             n ← i
8:         end if
9:     end for
10:    ColOrd ← ColOrd + n
11:    V ← V − n
12: for each k ∈ Γ(n) : deg(k)−
13: end while
```

Algorithme 5-3 Procedure-Style3()

```
1: while V != ∅ do
2:     if |C|+|P| < max : return
3:     choisir n le premier élément de V
4:     j ← 0
5:     for eachi ∈ V
6:         if deg(i) < deg(n) || (deg(i) == deg(n) && nebDeg(i) < nebDeg(n) ) ||
               (deg(i) == deg(n) && nebDeg(i) == nebDeg(n) && i > n )
7:             n ← i
8:         end if
9:     end for
10:    ColOrd ← ColOrd + n
11:    V ← V − n
12: end while
```