Gérer et analyser les grands graphes des entités nommées

Jocelyn Bernard

NNT : 2019LYSE1067

THESE de DOCTORAT DE L'UNIVERSITE DE LYON
opérée au sein de
l'Université Claude Bernard Lyon 1

Ecole Doctorale ED512
Informatique et Mathématiques de Lyon (InfoMaths)

Spécialité de doctorat : Informatique

Soutenue publiquement le 06/06/2019, par :
Jocelyn BERNARD

Gérer et analyser les grands graphes des entités nommées

Devant le jury composé de :

BARIL Jean-Luc	Professeur des Universités	Université de Bourgogne	Rapporteur
DAO Thi-Bich-Hanh	Maître de Conférences - HDR	Université d'Orléans	Rapporteure
BONIFATI Angela	Professeure des Universités	Université Lyon 1	Examinatrice
GRIGORI Daniela	Professeure des Universités	Université Paris-Dauphine	Examinatrice
KHEDDOUCI Hamamache	Professeur des Universités	Université Lyon 1	Directeur de thèse
GONCALVES Julien	Directeur R&D	ReportLinker	Invité

Résumé

Dans cette thèse nous étudierons des problématiques de graphes. Nous allons étudier des problématiques théoriques en recherche de motifs (patterns) et des problématiques appliquées en diffusion d'informations. Nous proposons deux études théoriques sur la recherche et l'énumération de sous-graphes denses que sont les cliques et quasi-cliques. Ensuite nous proposons une étude appliquée sur la propagation d'information dans un graphe d'entités nommées.

Dans un premier temps, nous allons étudier la recherche de cliques dans des graphes compressés. Les problèmes MCE (pour l'anglais *Maximal Clique Enumeration*) et MCP (pour l'anglais *Maximum Clique Problem*) sont des problèmes rencontrés dans l'analyse des graphes de données. Ce sont 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 proposons de répondre à ces problèmes en travaillant sur une version compressée du graphe initial. Nous montrons les bons résultats obtenus par notre méthode pour l'énumération de cliques maximales sur des graphes compressés.

Dans un second temps, nous étudierons l'énumération de quasi-cliques maximales. Nous proposons un algorithme distribué qui énumère l'ensemble des quasi-cliques maximales du graphe. Nous démontrons que cet algorithme liste l'ensemble des quasi-cliques maximales du graphe. Nous proposons également une heuristique qui liste un ensemble de quasi-cliques plus rapidement. Nous montrons l'intérêt de l'énumération de ces quasi-cliques par une évaluation des relations en regardant la co-occurrence des noeuds dans l'ensemble des quasi-cliques énumérées.

Dans un troisième temps, nous travaillerons sur la diffusion d'événements dans un graphe d'entités nommées. De nombreux modèles existent pour simuler des problèmes de diffusion de rumeurs ou de maladies dans des réseaux sociaux ainsi que des problèmes de propagation de faillites dans les milieux bancaires. Nous proposons de répondre au problème de diffusion d'événements marquant dans des réseaux hétérogènes représentant un environnement économique du monde. Nous proposons un problème de diffusion, nommé problème de classification de l'infection, qui consiste à déterminer quelles entités sont concernées par un événement. Pour résoudre ce problème, nous proposons deux modèles inspirés du modèle de seuil linéaire auxquels nous ajoutons différentes fonctionnalités. Finalement, nous testons et validons nos modèles sur un ensemble d'événements.

Mots-clefs : Théorie des graphes, problème d'énumération, cliques, quasi-cliques, graphes compressés, diffusion d'événement, propagation d'information, problème de classification, modèles de seuils.

Abstract

In this thesis we will study graph problems. We will study theoretical problems in pattern research and applied problems in information diffusion. We propose two theoretical studies on the identification/detection and enumeration of dense subgraphs, such as cliques and quasi-cliques. Then we propose an applied study on the propagation of information in a named entities graph.

First, we will study the identification/detection of cliques in compressed graphs. The MCE problems (for *Maximal Clique Enumeration*) and MCP (for *Maximum Clique Problem*) are problems that are encountered in the analysis of data graphs. These problem are difficult to solve (NP-Hard for MCE and NP-Complete for MCP), and adapted solutions must be found for large graphs. We propose to solve these problems by working on a compressed version of the initial graph. We show the correct results obtained by our method for the enumeration of maximal cliques on compressed graphs.

Secondly, we will study the enumeration of maximal quasi-cliques. We propose a distributed algorithm that enumerates the set of maximal quasi-cliques of the graph. We show that this algorithm lists the set of maximal quasi-cliques of the graph. We also propose a heuristic that lists a set of quasi-cliques more quickly. We show the interest of enumerating these quasi-cliques by an evaluation of relations by looking at the co-occurrence of nodes in the set of enumerated quasi-cliques.

Finally, we work on the event diffusion in a named entities graph. Many models exist to simulate diffusion problems of rumors or diseases in social networks and bankruptcies in banking networks. We address the issue of significant events diffusion in heterogeneous networks, representing a global economic environment. We propose a diffusion problem, called infection classification problem, which consists to dertemine which entities are concerned by an event. To solve this problem we propose two models inspired by the linear threshold model to which we add different features. Finally, we test and validate our models on a set of events.

Keywords : Graph theory, enumerating problem, cliques, quasi-cliques, compressed graph, event diffusion, information propagation, classification problem, linear threshold models.

Table des matières

Table des figures

Liste des algorithmes

Liste des tableaux

Remerciements

Introduction

Je résume souvent mon intérêt pour la science au fait que je me comporte comme un enfant. En primaire, lors des cours de mathématiques, j'ai toujours aimé résoudre le « Problème » que l'on nous posait en fin de chapitre. À la différence des autres exercices, il ne fallait pas simplement mettre en application ce que l'on nous avait appris mais utiliser les connaissances qu'on nous avait transmises pour émettre des hypothèses et arriver à une conclusion. J'ai continué ce jeu au collège et au lycée en cherchant constamment une autre solution que celle qui nous était enseignée. Lorsque j'ai été initié à la théorie des graphes à l'IUT de Bourg-en-Bresse, j'ai trouvé un nouveau terrain de jeux permettant de modéliser le monde et résoudre des problèmes de manière élégante. Les graphes sont en effet de puissants outils permettant de proposer des problèmes compliqués de façon simpliste. Ainsi, lorsqu'il m'a été proposé d'effectuer mon stage de recherche de Master sur une problématique de graphes, j'ai saisi l'opportunité et ai rejoint l'équipe GOAL.

J'ai par la suite eu la chance de pouvoir continuer dans cette voie lorsque l'on m'a proposé d'effectuer une thèse CIFRE conjointement avec le laboratoire LIRIS de l'Université Lyon 1 et la société Ubiquick/ReportLinker. Il y a plusieurs manières d'aborder une thèse de ce genre. On pourrait initialement craindre le fait de devoir naviguer entre deux endroits, et, par conséquent, de ne participer qu'à une partie des réunions, de ne plus être à jour des derniers avancements de tels ou tels projets, etc. Si l'on prend néanmoins ces contraintes comme des opportunités, on remarque qu'on a la chance de participer à plus de projets. Cette variété de projets permet d'avoir des échanges intéressants dans plusieurs domaines, ce qui évite de tomber dans une routine et permet de travailler aussi bien sur des problèmes théoriques qu'appliqués. Ce mémoire de thèse en est le parfait exemple, puisqu'il présente des travaux théoriques qui ont des motivations et applications concrètes, sur des données réelles.

Dans la première partie, nous étudierons des problèmes théoriques de graphes via la recherche de sous-graphes denses.

Le chapitre 1 est le résultat de l'approfondissement du travail effectué lors de mon stage de Master. Ce travail a initialement été motivé par l'intérêt qu'offrent les graphes compressés. La compression permet en effet d'obtenir des gains en mémoire pour le stockage des données. La question qui s'est posée était de savoir s'il était possible de travailler directement sur les graphes compressés et de résoudre des problèmes existant dans des graphes non-compressés. Nous avons donc cherché à résoudre, sur des graphes compressés, des problèmes connus : la recherche et l'énumération de cliques. Après avoir choisi une méthode de compression sans pertes, la décomposition modulaire, nous avons cherché à résoudre les deux problèmes sur la représentation compressée des graphes. Nous proposons pour cela une adaptation des algorithmes de recherche et d'énumération de cliques sur le modèle de graphe compressé. Le principe repose sur des regroupements de noeuds lorsque le voisinage est identique, ces regroupements permettent d'obtenir un graphe avec moins d'arêtes et de travailler sur ce dernier. Lorsqu'un agglomérat de noeuds est visité par un algorithme, notre méthode retourne les sous-ensembles de noeuds représentés par l'agglomérat.

Dans le chapitre 2, nous étudions l'énumération de quasi-cliques maximales. Ce travail a initialement été motivé par la constatation d'arêtes manquantes dans le graphe d'entités nommées de ReportLinker. Il est par exemple dommageable de réussir à trouver un lien entre *Google* et *Toyota* pour un projet de voitures autonomes et de rater celui entre *Google* et *Apple* alors que les deux sont des acteurs importants de la téléphonie mobile. Il nous a semblé intéressant de proposer une évaluation des liens entre sociétés. Dans l'exemple précédent, nous avons cherché à trier les entreprises par rapport à *Google* en énumérant celles qui avaient le plus d'interactions, même indirectes avec le noeud *Google*. Pour résoudre ce problème d'évaluation de liens (y compris en prenant en compte ceux manquants), nous avons cherché à étudier le nombre de voisins communs entre deux noeuds, en suivant l'idée que l'ami de mon ami est également mon ami. Nous avons donc proposé un algorithme d'énumération de l'ensemble des quasi-cliques maximales[1] du graphe. Face au constat de la difficulté du problème et du temps engendré par sa résolution, nous avons ensuite proposé une heuristique qui liste seulement une partie des quasi-cliques maximales. Dans le cadre d'échanges avec l'université de Tucson en Arizona, nous avons testé notre heuristique sur des graphes représentant des échanges textuels entre hackers, dans l'idée d'évaluer les relations malicieuses entre ces derniers.

1. une quasi-clique maximale ne peut être contenue dans une quasi-clique de plus grande taille

Dans la seconde partie, après avoir introduit le contexte du graphe des entités nommées de ReportLinker, nous avons cherché à modéliser la propagation d'événements dans le graphe.

Ainsi, le chapitre 3 présente l'étude de problèmes de propagations d'informations dans un graphe économique. La motivation derrière cette étude est de pouvoir fournir des informations sur l'impact supposé d'un événement sur une entité. L'idée à long terme, pour ReportLinker, est de pouvoir offrir des liens vers des nouvelles d'actualité en fonction des recherches de ses clients. Ces liens seraient choisis par les événements impactant les entités liées à la recherche de l'utilisateur. Pour représenter cette situation nous avons proposé un nouveau problème de diffusion dont le but est de déterminer quelles entités sont concernées par un événement donné. Nous avons donc défini, avec l'aide d'analystes économiques de ReportLinker, une liste d'événements tel que le rachat de Bayer par Monsanto ou l'ouragan Katrina qui a touché la Nouvelle-Orléans en 2005. Pour ces événements, les analystes ont dû définir les noeuds qu'ils pensaient concernés par l'événement. Nous avons ensuite développé des modèles qui devaient proposer une classification des entités du graphe de ReportLinker en disant lesquelles lui semblait concernées par l'événement. Les résultats des modèles ont ensuite été comparés à ceux demandés par les analystes : le modèle donnant les meilleurs résultats étant celui dont les classifications sont les plus proches de celles souhaitées par les analystes.

Ce manuscrit se termine par conclusion ainsi que la présentation des perspectives qui nous semblent les plus intéressantes à explorer pour l'ensemble des chapitres.

Terminologie

Ensembles

Un *ensemble* E correspond à une collection d'objets, appelés *éléments*, qui peut être fini ou non.

On dit que les éléments appartiennent à l'ensemble, ainsi l'élément e appartenant à E sera noté $e \in E$.

La *cardinalité* ou taille d'un ensemble correspond au nombre de ces éléments, elle est notée $|E|$. Ainsi l'ensemble $E = \{e_1,e_2,...,e_n\}$ sera de taille n.

Un ensemble peut être composé d'aucun objets, on parle alors d'ensemble vide $\emptyset$. La cardinalité de l'ensemble vide est de 0.

Le *complémentaire* d'un ensemble E noté $\overline{E}$ représente l'ensemble des éléments n'appartenant pas à E.

De nombreuses opérations sont possibles afin de manipuler les ensembles ;

On peut notamment ajouter un élément e à l'ensemble que l'on notera $e \rightarrow E$, l'ensemble formé pourra se noter $E + e$.

On peut également supprimer un élément e de l'ensemble E, que l'on notera $E - e$.

L'*union* est une opération qui consiste à regrouper deux ensembles E et F et qui est noté $E \cup F$. Ainsi tout élément e appartenant à E ou F appartiendra à l'union $E \cup F$.

L'*intersection* est une opération qui consiste à sélectionner l'ensemble des éléments appartenant à deux ensembles E et F, elle est notée $E \cap F$. Ainsi tout élément e appartenant à E et à F appartiendra à l'intersection $E \cap F$.

La *soustraction* est une opération sur deux ensembles E et F visant à conserver les éléments présents uniquement dans le premier ensemble, elle est noté $E - F$. Ainsi tout élément e appartenant à $E \cap \overline{F}$ appartiendra à la soustraction $E - F$.

L'*inclusion* correspond à une relation d'appartenance d'un premier ensemble E à un deuxième F. Ainsi tout élément de E sera présent dans F. On dira alors que E est un *sous-ensemble* de F. L'inclusion sera notée $E \subset F$.

Un sous-ensemble respectant certaines conditions sera dit *maximal* s'il ne peut être inclus dans un sous-ensemble de plus grande cardinalité respectant les mêmes conditions.

L'*équivalence* correspond à une relation d'égalité entre deux ensembles E et F. Ainsi tout élément de E est dans F et tout élément de F est dans E, $E \subset F$ et $F \subset E$. L'équivalence sera notée $E \equiv F$.

Deux ensembles E et F sont dits *disjoints* si leur intersection est vide : $E \cap F \equiv \emptyset$.

Une *fonction* est une relation qui permet de passer d'un élément du premier ensemble à un élément du second ensemble, elle est noté f et $f(e)$ lorsqu'elle s'applique à l'élément e.

Une *application* est une fonction qui pour tout élément élément du premier ensemble E est relié à un unique élément du second ensemble F.

Une *bijection* ou application bijective est une fonction qui pour tout élément du second ensemble à un unique antécédent dans le premier ensemble. Ainsi on peut définir la fonction réciproque de f, notée f^{-1} qui est une application qui permet de passer du second ensemble F au premier ensemble E.

Graphe

Un graphe $G(V,E)$ est défini par un ensemble de noeuds V et un ensemble d'arêtes E, $E \subseteq V^2$. Une arête $(u,v) \in E$, $u,v \in V$, correspond à un lien entre deux noeuds. Soit $e = (u,v)$ une arête, u et v sont appelés les *extrémités* de e.

Un graphe peut être *orienté*, les arêtes sont alors *dirigées*, $(u,v) \neq (v,u)$. Si les arêtes sont orientées on parle alors de *graphe orienté*. Dans le cas contraire on parle de *graphe non-orienté*. Pour un graphe dirigé, on distinguera la source $(u,.)$ d'une arête de la destination $(.,v)$. Ainsi un noeud u aura un ensemble d'*arêtes entrantes* $(v,u) \in E$ et un ensemble d'*arêtes sortantes* $(u,v) \in E$.

Les noeuds ou les arêtes d'un graphe peuvent être *pondérés*. La fonction w donne alors un poids $w(u)$, $u \in V$, aux noeuds et un poids $w((u,v))$, $(u,v) \in E$, aux arêtes. Le poids d'un noeud sera noté w_u, celui d'une arête $w_{(u,v)}$. Un graphe n'utilisant pas de fonction de pondération sera appelé *graphe non-pondéré*.

Un graphe peut également avoir une ou plusieurs fonctions d'étiquetages, on parle alors de *graphe étiqueté*. Cela permet de classifier les noeuds ou les arêtes afin de donner des informations utiles pour un utilisateur ou un algorithme. L'*identifiant* des noeuds ne sera pas considéré, dans ce manuscrit, comme une fonction d'étiquetage. Ainsi un graphe n'ayant pas de fonction d'étiquetage ou uniquement une servant à l'identification des noeuds sera considéré comme un *graphe non-étiqueté*.

Une *boucle* est une arête dont les deux extrémités sont identiques, $(u,u) \in E$, $u \in V$.

Lorsque deux arêtes ou plus relient deux mêmes noeuds on parle alors d'*arêtes multiples*. Un graphe composé d'arêtes multiples est appelé *multi-graphe*.

Un graphe qui n'est composé ni de boucles ni d'arêtes multiples est appelé *graphe simple*.

Dans la première partie, sauf mentions contraires, nous étudierons des graphes simples non-orientés, non-pondérés et non-étiquetés. Dans la seconde partie, sauf mentions contraires, nous étudierons des multi-graphes orientés, pondérés et étiquetés.

Si (u,v) est une arête du graphe on dit que les noeuds u et v sont voisins. On appelle *voisinage* l'ensemble des voisins d'un noeud, noté $\Gamma(u)$.

Le *degré* d'un noeud dans G, noté $deg(u)$ ou $d(u)$, correspond au nombre d'arêtes dans lesquelles le noeud est présent. Si le graphe est orienté on distinguera le degré entrant $deg^-(u)$, correspondant au nombre d'arêtes entrantes, du degré sortant $deg^+(u)$, correspondant au nombre d'arêtes sortantes. Dans le cas d'un graphe simple le degré est égal à la taille du voisinage $deg(u) = |\Gamma(u)|$, ce qui n'est pas le cas pour les multi-graphes. Le *degré maximal* du graphe G, noté $\Delta(G)$, correspond au plus grand degré de l'ensemble des noeuds : $\forall u \in V$, $\Delta(G) = max(deg(u))$. Le *degré minimal* du graphe G, noté $\delta(G)$, correspond au plus petit degré de l'ensemble des noeuds : $\forall u \in V$, $\delta(G) = min(deg(u))$.

La *densité* d'un graphe correspond au rapport entre le nombre d'arêtes présentes dans le graphe et le nombre théorique maximal d'arêtes dans le graphe. Pour un graphe non-orienté simple, il est défini par : $densité = \frac{2|E|}{|V|(|V|-1)}$.

Une *chaîne* entre deux noeuds u et v correspond à une suite de noeuds $u = u_0,u_1,...,u_{k-1},v = u_k$ de telle sorte qu'il existe, pour tout entier i compris entre 0 et $k-1$, une arête entre les noeuds u_i et u_{i+1} : $\forall i \in \{0,...,k-1\}$, $\exists(u_i,u_{i+1}) \in E$.

Un *chemin* correspond à une chaîne dont les noeuds sont distincts. Un chemin est dit *court* s'il n'existe pas de chemin de taille inférieur à k entre u et v.

Le *diamètre* d'un graphe correspond au plus long chemin de l'ensemble des plus courts chemins du graphe. Le diamètre d'un graphe est noté $D(G)$.

Un *cycle* correspond à un chemin fermé, c'est à dire que $u = v$.

Un graphe est *connexe* s'il existe au moins un chemin entre n'importe quels noeuds du graphe. Une *composante connexe* est un sous-graphe maximal connexe d'un graphe *non-connexe*. Dans le cas de graphe orienté on parle de *composante fortement connexe* si et seulement si, pour toute paire de noeuds u et v appartenant à la composante fortement connexe, il existe un chemin de u vers v et un chemin de v vers u.

Une *clique* est un sous-ensemble complet de noeuds du graphe. C'est-à-dire que chacun des noeuds du sous-ensemble est connecté avec l'intégralité des autres. La taille de la plus grande clique d'un graphe G est notée $\omega(G)$.

Un *stable* est un sous-graphe indépendant du graphe. C'est-à-dire que chacun des noeuds du sous-ensemble ne possède aucun voisin dans le sous-ensemble.

Une *forêt* est un graphe sans cycle. Un *arbre* est une forêt composée d'une seule composante connexe. Dans une forêt, les noeuds de degré 1 sont appelés des *feuilles*. Un noeud d'un arbre peut être parfois considéré comme une *racine*. Tous les autres noeuds ont alors une *profondeur* définie par la taille du chemin entre eux et la racine. Une relation *père-fils* est alors créée entre les noeuds liés par une arête, les noeuds de profondeurs p étant les pères des noeuds de profondeur $p+1$ auxquels ils sont liés. En conséquence cela créé un graphe dirigé où les feuilles sont les noeuds étant uniquement une destination d'une seule arête. Ces feuilles n'ont donc pas de fils et la racine de l'arbre n'a pas de père.

Première partie

Théorie des Graphes

Chapitre 1

Cliques dans un graphe compressé

Résumé

Dans ce chapitre, nous étudierons les problèmes d'énumérations de cliques maximales et de recherche de clique maximum sur des graphes compressés. Nous présenterons les méthodes de compressions et de recherche de cliques présents dans la littérature dans la section 1.2. Dans la section 1.3, nous détaillerons notre raisonnement pour la résolution de problèmes de cliques sur un graphe compressé. Dans la section 1.4, nous proposerons une évaluation de nos méthodes via une expérimentation.

1.1 Introduction

La compression de graphe est une technique correspondant à une suite d'opérations permettant la diminution du nombre d'arêtes et/ou de noeuds du graphe. Cette technique permet notamment de rendre un graphe plus facile à lire ou à transmettre en diminuant les informations. Plusieurs opérations élémentaires peuvent être combinées pour arriver à la compression d'un graphe.

Parmi ces opérations élémentaires, on trouve :

— La *suppression d'une arête*, qui consiste à retirer une arête e de l'ensemble E.

— La *suppression de noeud*, qui consiste à retirer un noeud u à l'ensemble V. En conséquence, on utilise également la suppression de toutes les arêtes où u est une extrémité de l'ensemble E.

— L'*ajout d'une arête*, qui consiste à ajouter une arête $e = (u,v)$ à l'ensemble E, $u,v \in V$.

— L'*ajout de noeud*, qui consiste à ajouter un noeud u à l'ensemble V. On peut alors, au besoin, utiliser l'ajout d'arêtes pour relier ce noeud à d'autres noeuds déjà présents dans V.

Ces opérations sont présentées dans la figure 1.1.

La *fusion* de noeuds est une opération permettant de regrouper deux ou plusieurs noeuds en un seul gardant le voisinage des noeuds regroupés. La fusion consiste en la succession de plusieurs étapes élémentaires :

1. La définition d'une ensemble $S \in V$ de noeuds à fusionner.
2. L'ajout d'un nouveau noeud s à l'ensemble V.
3. L'ajout d'arêtes ayant strictement une extrémité dans S et une dans $V - S$, où le noeud u dans S est remplacé par le noeud s :

$$\begin{aligned} &\forall u \in S,\ v \in V - S, \\ &\quad \exists (u,v) \in E \Rightarrow (s,v) \rightarrow E, \\ &\quad \exists (v,u) \in E \Rightarrow (v,s) \rightarrow E \end{aligned}$$

4. La suppression des noeuds de l'ensemble S et de leurs arêtes incidentes dans E.

La figure 1.2 présente l'opération de fusion.

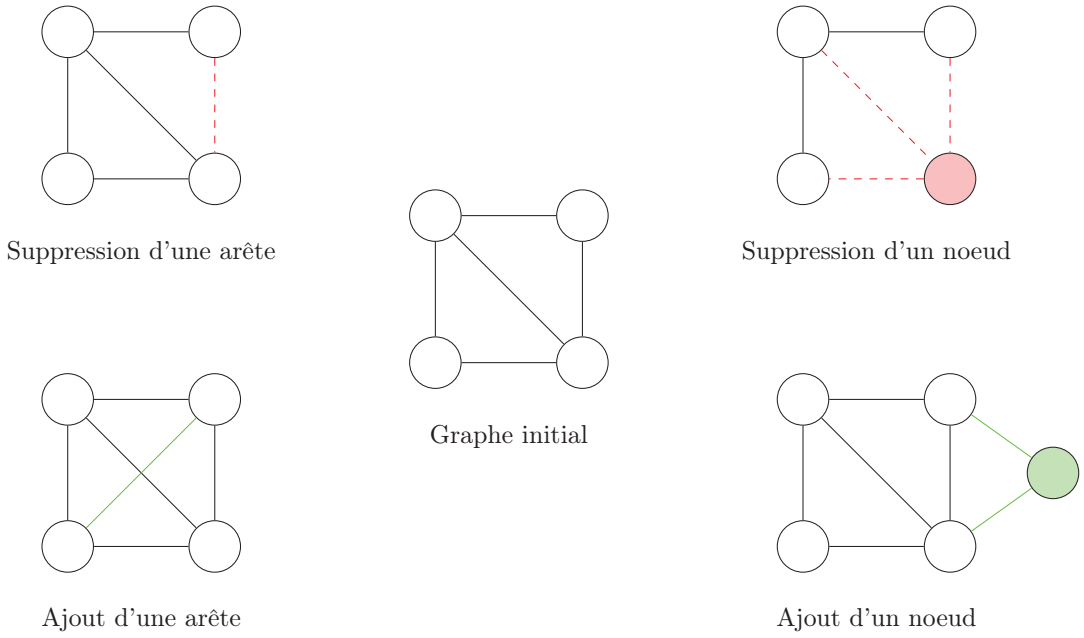


FIGURE 1.1 – Présentation d'opérations élémentaires : Suppression d'une arête, suppression d'un noeud, ajout d'une arête et ajout d'un noeud.

L'étape de *fusion de noeuds* est l'étape principale de la compression de graphes.

La *compression de graphe* correspond à une suite d'étapes de fusions de noeuds qui aboutit à la transformation d'un graphe initial G en un graphe compressé G_C. Le graphe compressé G_C possède généralement un nombre de noeuds et d'arêtes inférieur aux nombres de noeuds et d'arêtes du graphe G.

Les techniques de compression de graphes se classent en deux catégories : les compressions avec perte et celles sans perte. La *perte* correspond à l'altération d'une partie des données initiales lors de la compression. Il n'est donc pas possible de recréer avec exactitude le graphe initial lors d'une étape de *décompression*.

Les opérations de compression de graphes sont possibles sur différents types de graphes : simples [83], orientés [2], étiquetés [106], etc.

Les cliques sont des ensembles de noeuds correspondant à des sous-graphes complets. Une clique de taille n, notée K_n est donc un sous-ensemble de V où chacun des noeuds possède une arête avec les autres noeuds de la clique K dans E.

On retrouve dans l'étude des cliques deux problèmes connus :

— Le *problème de recherche de clique maximum* ou *MCP* pour l'anglais *Maximum Clique Problem*. Ce problème consiste à trouver, pour un graphe G, la taille de la plus grande clique présente dans ce graphe.

— Le *problème d'énumération de cliques maximales* ou *MCE* pour l'angalis *Maximal Clique Enumeration*. Ce problème consiste à lister, pour un graphe G, l'ensemble des cliques maximales. Une clique est considérée comme *maximale* si elle n'est pas incluse dans une clique de taille supérieure.

Nous proposons dans ce chapitre de résoudre les problèmes *MCP* et *MCE* sur un graphe compressé. La recherche d'une telle méthode vise à résoudre des contraintes liées à l'étude de ces problèmes :

— Un graphe compressé prend moins d'espace mémoire et peut dont être chargé en mémoire centrale lors du traitement. Ceci permet de réduire le nombre d'entrées / sorties si le graphe initial ne tient pas en mémoire centrale.

— Le graphe compressé étant plus petit, un gain de temps lors des opérations de traitement des problèmes peut éventuellement être réalisé.

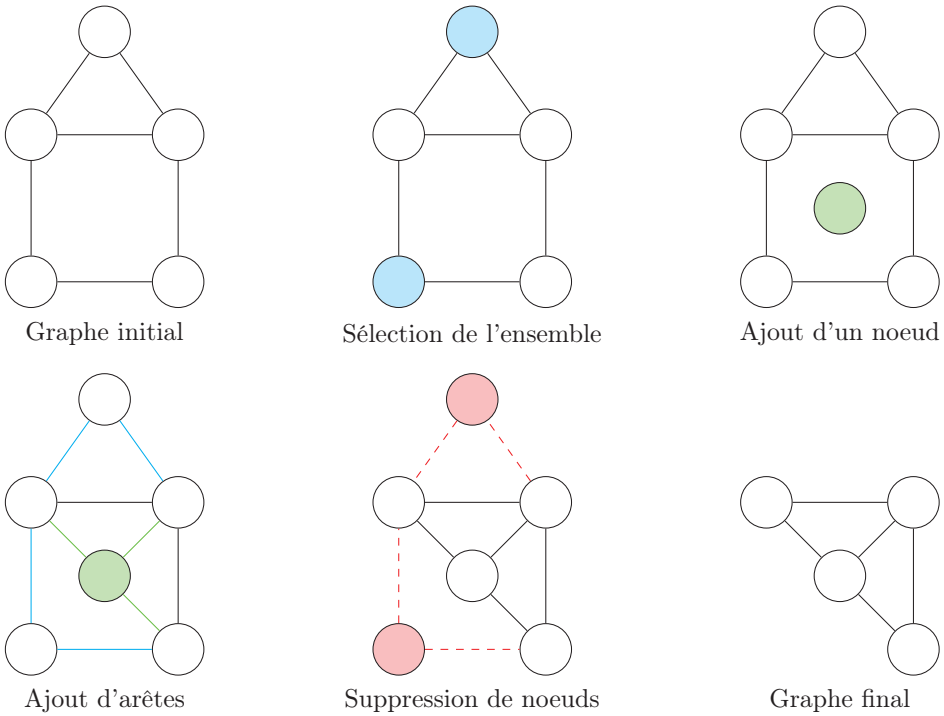


FIGURE 1.2 – Présentation des étapes correspondant à une suite d'opérations élémentaires aboutissant à la *fusion* de noeuds : Sélection de l'ensemble des noeuds à fusionner, création d'un noeud, ajout d'arêtes correspondant au voisinage de l'ensemble et suppression de l'ensemble.

1.2 État de l'art

1.2.1 Compression de graphe

La compression de graphe peut être opérée de différentes façons, selon le type de graphe sur lesquels elle s'applique et selon le type de compression souhaité (avec ou sans perte).

Navlakha et al. [83] proposent une compression de graphe sans perte qui aboutit à la création d'un graphe compressé $G_C(V_S,E_S,S,C)$ où :

— V_S correspond à l'ensemble des nouveaux noeuds du graphe G_C
— E_S correspond à l'ensemble des nouvelles arêtes du graphe G_C
— S correspond à une fonction de transition des noeuds de V vers V_S : $\forall u \in V, \exists u_S, S(u) = u_S$
— C correspond à un ensemble d'arêtes correctives s'appliquant à l'ensemble des arêtes E_S permettant de préserver les informations du graphe initial. Les arêtes sont étiquetées. L'étiquette est un $+$ ou un $-$ selon s'il faut ajouter ou supprimer ces arêtes au graphe obtenu par décompression du graphe G_C

La figure 1.3 présente la compression d'un graphe selon la méthode Navlakha et al. [83]. On remarque que les informations présentes dans la partie droite correspondent au graphe compressé G_C. Ces informations permettent de recréer le graphe initial en deux étapes : une étape de décompression des noeuds V_S en V grâce à la table induite par la fonction de transition S ; puis une étape de correction des arêtes par l'ajout et la suppression des arêtes données dans l'ensemble de correction C.

Alder et Mitzenmacher [2] puis Boldi et Vigna [16] proposent de compresser les graphes du web à l'aide de la différence entre les listes de voisinages de deux noeuds. Chaque noeud u est compressé à l'aide d'un autre noeud v et d'une correction correspondant à la différence entre les voisins de u et de v (on ajoute les

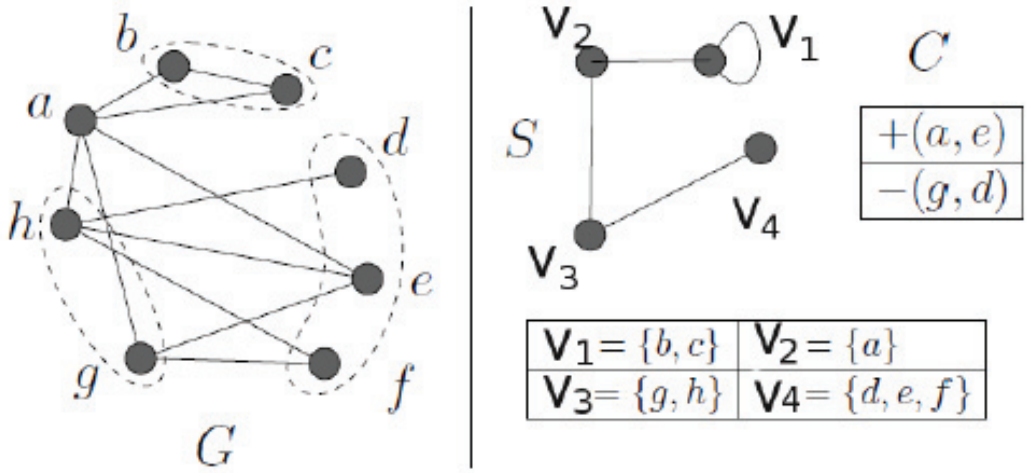


FIGURE 1.3 – Exemple de compression selon la méthode de Navlakha et al.[83], à gauche se trouve le graphe initial G et à droite le graphe compressé ainsi que la liste de correction C et la table de transitions induite par la fonction S.

voisins manquant et enlève les voisins en trop) :

$$\forall u,v,x \in V, \exists (u,v) \in E_C \Rightarrow C((u,v)) = \\ +(u,x),\ x \in \Gamma(u) - \Gamma(v) \\ -(u,x),\ x \in \Gamma(v) - \Gamma(u)$$

Tian et al. [106] proposent une compression de graphes étiquetés supervisée par l'utilisateur afin que ce dernier puisse observer le graphe selon une taille souhaitée. Cette compression est avec perte, 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.

Pour les graphes pondérés Toivonen et al. [107] proposent une méthode de compression avec peu de pertes. L'idée est de regrouper des noeuds ayant des voisinages similaires (en terme de voisins et de poids). Ainsi les noeuds sont agrégés en *super-noeuds*, nommés par la succession des identifiants des noeuds initiaux. Les arêtes sont elles agrégées en *super-arêtes*, dont le poids est calculé par la moyenne des arêtes initiales (s'il n'y avait pas d'arête entre deux noeuds on simule une arête de poids nulle pour le calcul).

Décomposition modulaire

Une autre méthode, issue des travaux de Gallai [36], appelée la *décomposition modulaire* peut être utilisée pour compresser des graphes [80, 50]. La méthode de décomposition modulaire revient à réaliser des agglomérats de noeuds ou super-noeud, appelés *modules*.

Définition 1.1. Un module du graphe $G(V,E)$ est un sous-ensemble $M \subset V$ où chacun des noeuds $u \in M$ a le même voisinage dans $V - M$. Ainsi :

$$\forall u,v \in M \Rightarrow \\ \Gamma(u) \cap (V - M) = \Gamma(v) \cap (V - M)$$

Le graphe compressé obtenu est représenté sous la forme d'un arbre de modules $A(M,E_A,f,C)$, appelé *arbre de décomposition modulaire* où :

— M est l'ensemble des modules, correspondant à des noeuds dans l'arbre.
— E_A est l'ensemble des arêtes de l'arbre de décomposition modulaire.
— f est une fonction attribuant un module feuille à un noeud u de V.
— C est un ensemble d'arêtes liant des modules fils de modules premiers.

Différents types de modules sont utilisés, ils sont définis en fonction du voisinage des noeuds initiaux compressés.

Les modules *feuilles* : chaque module feuille correspond à un noeud du graphe initial. Ce sont également des feuilles de l'arbre de décomposition modulaire produit. L'ensemble des modules feuilles est noté M_f.

Définition 1.2. Un *module feuille* correspond à un noeud feuille u_f dans l'arbre de décomposition modulaire A.

La fonction f, qui s'applique aux noeuds appartenant à M_s, est une fonction bijective qui permet d'attribuer à tout noeud u du graphe initial G un module u_s de l'arbre de décomposition modulaire.

Définition 1.3. f est une fonction bijective correspondant à une relation binaire de V dans M_s :

$$\forall u \in V,\ f(u) = u_s,\ u_s \in M_s$$

Sa fonction réciproque f^{-1} correspond à une application de M_s dans V :

$$\forall u_s \in M_s,\ f^{-1}(u_s) = u$$

Les modules *séries* : un module série correspond à un ensemble complet dont le voisinage est identique. Ainsi, toute clique de modules dont le voisinage est identique deviendra un module série dans l'arbre de décomposition modulaire. Les noeuds fils du module série correspondront aux modules de la clique. L'ensemble des modules séries est noté M_{ser}.

Définition 1.4. Un *module série* correspond à un noeud u_{ser} dans l'arbre de décomposition modulaire A. Tout module fils de u_{ser} possède un voisinage identique dans G et est voisin des autres modules fils de u_{ser}.

Les modules *parallèles* : un module parallèle correspond à un ensemble indépendant dont le voisinage est identique. Ainsi, tout stable de modules dont le voisinage est identique deviendra un module parallèle dans l'arbre de décomposition modulaire. Les noeuds fils du module parallèle correspondront aux modules du stable. L'ensemble des modules parallèles est noté M_{par}.

Définition 1.5. Un *module parallèle* correspond à un noeud u_{par} dans l'arbre de décomposition modulaire A. Tout module fils de u_{par} possède un voisinage identique dans G et est indépendant des autres modules fils de u_{par}.

Les modules *premiers* : un module premier correspond à un ensemble ni complet, ni indépendant et dont le voisinage est identique. Ainsi, tout ensemble de modules n'étant ni un stable, ni une clique et dont le voisinage est identique deviendra un module premier dans l'arbre de décomposition modulaire. Les noeuds fils du module premier correspondront aux modules de l'ensemble. L'ensemble des modules premiers est noté M_{PREM}.

Définition 1.6. Un *module premier* correspond à un noeud u_{PREM} dans l'arbre de décomposition modulaire A. Tout module fils m_f de u_{PREM} possède un voisinage identique dans G et est voisin d'un sous-ensemble SE des noeuds fils u_{PREM} (le sous-ensemble SE ne correspond ni à l'ensemble vide $\emptyset$, ni à l'ensemble des noeuds fils E_f privé du-dit module fils $E_f - m_f$).

L'ensemble des arêtes C est un ensemble d'arêtes pour conserver les liens dans les modules premiers. C permet à chaque module fils d'un module premier son voisinage parmi l'ensemble des modules fils du noeud premier.

Définition 1.7. L'ensemble de correction C correspond à des arêtes (u,v) où u et v sont des modules fils d'un même module premier. L'arête (u,v) est présente dans C si u et v possèdent une arête dans le sous-graphe induit par l'ensemble des modules fils du module premier.

L'ensemble des modules M est donc l'union des ensembles des modules feuilles, séries, parallèles et premiers.

Définition 1.8. Soit M l'ensemble des modules :

$$M = M_f \cup M_{ser} \cup M_{par} \cup M_{PREM}$$

Les modules M_f, M_{ser}, M_{par} et M_{PREM} sont disjoints.

L'ensemble des arêtes E_A représente les arêtes de l'arbre de décomposition modulaire.

Définition 1.9. Pour toute relation entre un module père p et un module fils f dans l'arbre de décomposition modulaire A, il existe une arête $(p,f) \in E_A$.

La figure 1.4 illustre la compression d'un graphe en utilisant la décomposition modulaire. L'arbre de décomposition modulaire correspondant à la compression est présenté dans la figure 1.5. Les noeuds qui possèdent un voisinage identique sont coloriés de la même manière que dans le graphe initial (voir figure 1.4(a)).

Ainsi :

— Les noeuds 1 et 2 ont le même voisinage et l'ensemble forme une clique. Ils forment un module série S(1,2) (voir figure 1.4(b)).

— Les noeuds 0 et 8 ont le même voisinage et l'ensemble forme un stable. Ils forment un module parallèle P(0,8) (voir figure 1.4(b)).

— Les noeuds 5, 6 et 7 possèdent le même voisinage. Les noeuds 5 et 6 formant une clique dans le sous-ensemble 5,6,7, ils forment un module série S(5,6). Ce module série forme alors un stable avec le noeud 7 dans le sous-ensemble S(5,6),7. Ce stable est représenté par un module parallèle P(S(5,6),7) (voir figure 1.4(c) / 1.4(d)).

— Le sous-ensemble P(0,8),S(1,2),3,4,P(S(5,6),7) possède un voisinage identique (l'ensemble vide $\emptyset$) et aucun des noeuds ne possède un voisinage identique dans le sous-ensemble. Ainsi, on créé le module premier $Prem(P(0,8),S(1,2),3,4,P(S(5,6),7)$. On ajoute ensuite les arêtes $(P(0,8),S(1,2))$, $(P(0,8),4)$, $(3,S(1,2))$, $(3,P(S(5,6),7))$ et $(4,P(S(5,6),7))$ à C.

L'arbre de décomposition modulaire présenté en figure 1.5 permet donc de représenter la version compressée du graphe initial.

Le tableau 1.1 présente les modèles de compression présentés dans ce chapitre et des applications possibles sur différents types de graphes.

Type de graphe	**Type de compression**	**Application**
Graphes simples	Compression l'aide d'une liste de correction [83]	Tout type de réseaux
	Décomposition par modules [36, 80]	Réseau sociaux, Graphes biologiques
Graphes orientés	Compression par différence entre les nœuds [2, 16]	Graphes du web
Graphes étiquetés	Compression supervisée [106]	Schémas de bases de données, grands graphes
Graphes pondérés	Agglomération de nœuds et d'arêtes [107]	Grands graphes de réseaux sociaux ou de biologie

Tableau 1.1 – Méthodes de compressions de graphes.

1.2.2 Cliques Maximales

Les cliques *maximales* sont des cliques qui ne peuvent pas être incluses dans des cliques de tailles plus grandes.

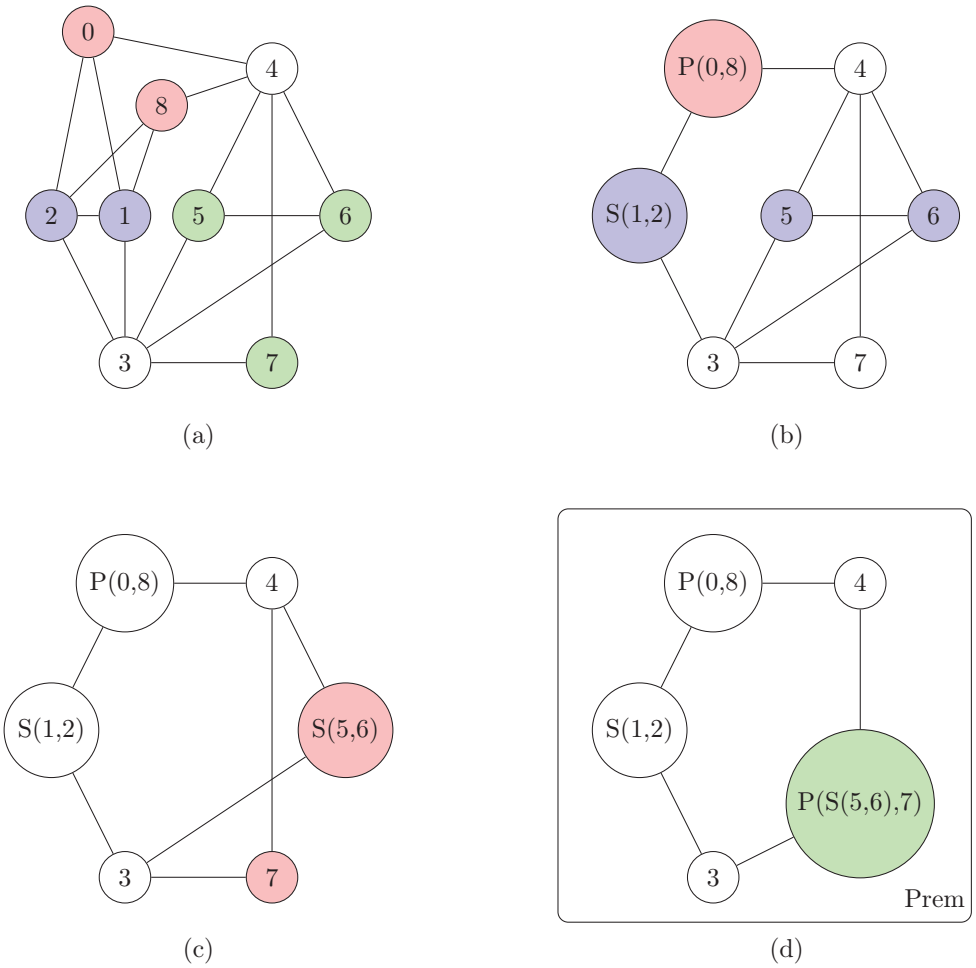


FIGURE 1.4 – Compression d'un graphe (a) par décomposition modulaire avec deux étapes intermédiaires (b), (c), et le graphe compressé obtenu (d).

Définition 1.10. Une clique K_n de taille n est dite maximale s'il n'existe pas de clique $K_m \subset V$ de taille m tel que :

$$K_n \subset K_m \textbf{ ET } m > n$$

La recherche de ces sous-sembles est étudiée notamment dans le domaine de la biologie pour l'interaction protéines-protéines [91] ou pour la recherche d'espèces virales proches [112].

La recherche de la clique maximum est également utilisée pour la coloration de graphes. La valeur de la clique maximum permet de donner une borne inférieure au nombre chromatique [1].

1.2.3 Recherche de clique maximum

La *recherche de clique maximum* abrégée *MCP* pour l'anglais *Maximum Clique Problem* est un problème où le but est de trouver la clique de plus grande taille du graphe.

Définition 1.11. Pour tout graphe $G(V,E)$ la clique de taille maximum notée $\omega(G)$ correspond à la clique de plus grande cardinalité du graphe :

$$\forall K \subset V, \omega(G) \geq |K|$$

1. Le nombre chromatique est le nombre minimum de couleurs utilisées de manières à ce que tout voisin d'un noeud u ne soit pas coloré de la même manière que le noeud u.

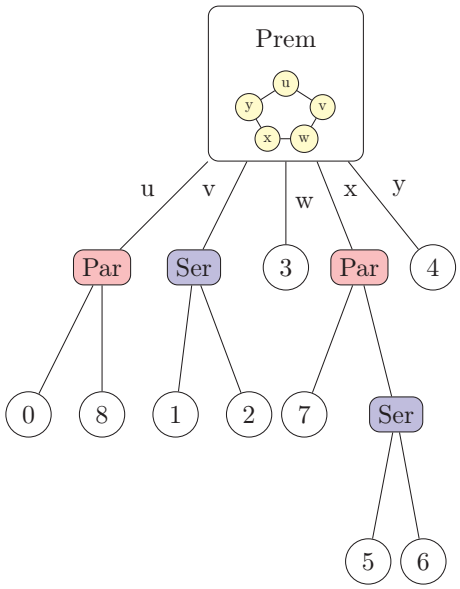


FIGURE 1.5 – Arbre de décomposition modulaire correspondant au graphe de la figure 1.4.

Ce problème fait partie des problèmes *NP-Complets* énoncés par Karp [59].
Nous présenterons ici certains des principaux algorithmes de la littérature.

Algorithmes MC et dérivés

Fahle [33] propose un algorithme exact qui permet de définir la clique maximum d'un graphe simple. Son algorithme, que nous appellerons *MC*, utilise 3 ensembles :

— Un ensemble de noeud C, initialement vide, qui contient à un instant t une clique partielle.
— Un ensemble de noeuds candidats P, initialisé avec tous les noeuds du graphe. P contient, à un instant t, l'ensemble des noeuds voisins de chacun des noeuds de l'ensemble C.
— Un ensemble K, initialement vide, qui permet de conserver en mémoire l'ensemble des noeuds composant la plus grande clique trouvée.

L'algorithme utilise le principe du *retour sur trace* (nommé *backtrack* en anglais) qui consiste à revenir en arrière après une décision prise à un instant t pour explorer d'autres solutions à partir de l'état dans lequel on était à l'instant juste avant cette décision. L'algorithme MC est présenté en algorithme 1.

Une première amélioration de l'algorithme, appelée *MC0*, est de choisir uniquement des nouveaux noeuds n dont l'identifiant est inférieur à celui de v. Cette optimisation permet d'éviter les doublons. Cela permet un gain de temps de l'ordre de 10% selon Prosser [90].

Proposition 1.12. Pour MC0, la construction de la clique C par ajout successif de noeuds, dont l'identifiant est toujours plus petit que celui précédemment ajouté, permet d'éviter les doublons.

Preuve. Soit un instant t avec C_t la clique partielle et n,$v \in P_t$, $n < v$, ainsi :

$$\begin{aligned} C_t + v &\Rightarrow n \in P_{C_t+v} \\ &\Rightarrow C_t + v + n \end{aligned}$$

$$\begin{aligned} C_t + n &\Rightarrow v \notin P_{C_t+n} \\ &\Rightarrow C_t + n \end{aligned}$$

On ne reconstruit donc pas une clique $C_t + n + v$ □

Algorithme 1 $ProcedureMC(C,P,K)$

1: **tant que** $P \neq \emptyset$ **faire**
2: **si** $|C| + |P| \leq |K|$ **alors**
3: retourner $\emptyset$
4: **fin si**
5: choisir v le dernier élément de P
6: $C \leftarrow C + v$
7: **si** $|P \cap \Gamma(v)| = 0$ **ET** $|C| > |K|$ **alors**
8: $K \leftarrow C$
9: **fin si**
10: **si** $|P \cap \Gamma(v)| \neq 0$ **alors**
11: $T \leftarrow Procedure - MC(C,P \cap \Gamma(v),K)$
12: **si** $|T| > |K|$ **alors**
13: $K \leftarrow T$
14: **fin si**
15: **fin si**
16: $C \leftarrow C - v$
17: $P \leftarrow P - v$
18: **fin tant que**
19: retourner K

D'autre améliorations de MC ont été proposées, notamment celle fonctionnant sur un ordonnancement de l'exploration des noeuds (ordonnancement lui-même basé sur une coloration).

Ainsi Tomita et Seki [109] proposent un algorithme, *MCQ*, où la liste des candidats P est ordonnée selon la coloration proposée par Welsh and Powell [116]. Par la suite, Tomita va par deux fois proposer des variations sur cet algorithme avec les algorithmes *MCR* [108] et *MCS* [110]. San Segundo proposera également une version différente [96], nommée *BBMC*, qui recalcule à chaque étape l'ordonnancement des noeuds (en utilisant de plus des chaînes de bits pour effectuer l'ordonnancement en temps constants).

Afin de réaliser l'ordonnancement des noeuds, 3 techniques différentes sont proposées : Le premier sur la cardinalité du voisinage des noeuds (voir algorithme 2), le second sur la taille du voisinage des noeuds qui n'ont pas déjà été classés (voir algorithme 3) et le dernier sur la taille du voisinage, suivie de la somme de la taille des voisinages des voisins du noeud (voir algorithme 4). L'effet des ordonnancements sont présentés sur la figure 1.6.

Algorithme 2 $CardinalitéVoisinage(V)$

1: $R \leftarrow \emptyset$
2: **tant que** $V \neq \emptyset$ **faire**
3: prendre n, le premier élément de V
4: **pour tout** $i \in V$ **faire**
5: **si** $|\Gamma(i)| < |\Gamma(n)|$ **OU** $(|\Gamma(i)| = |\Gamma(n)|$ **ET** $i > n)$ **alors**
6: $n \leftarrow i$
7: **fin si**
8: **fin pour**
9: $R \leftarrow R + n$
10: $V \leftarrow V - n$
11: **fin tant que**
12: retourner R

Une étude des performances de ces algorithmes est proposée par Prosser [90].

Algorithme 3 $CardinalitéTriée(V)$

1: $R \leftarrow \emptyset$
2: **tant que** $V \neq \emptyset$ **faire**
3: prendre n, le premier élément de V
4: **pour tout** $i \in V$ **faire**
5: **si** $|\Gamma(i) \cap V| < |\Gamma(n)|$ **alors**
6: $n \leftarrow i$
7: **fin si**
8: **fin pour**
9: $R \leftarrow R + n$
10: $V \leftarrow V - n$
11: **fin tant que**
12: retourner R

Algorithme 4 $CardinalitéVoisinagesVoisins(V)$

1: $R \leftarrow \emptyset$
2: **pour tout** $i \in V$ **faire**
3: **pour tout** $j \in \Gamma(i)$ **faire**
4: $voisinsDegres(i) \leftarrow voisinsDegres(i) + |\Gamma(j)|$
5: **fin pour**
6: **fin pour**
7: **tant que** $V \neq \emptyset$ **faire**
8: prendre n, le premier élément de V
9: **pour tout** $i \in V$ **faire**
10: **si** $|\Gamma(i)| < |\Gamma(n)|$ **OU** $(|\Gamma(i)| = |\Gamma(n)|$ **ET** $voisinsDegres(i) < voisinsDegres(n))$ **OU** $(|\Gamma(i)| = |\Gamma(n)|$ **ET** $voisinsDegres(i) = voisinsDegres(n)$ **ET** $i > n)$ **alors**
11: $n \leftarrow i$
12: **fin si**
13: **fin pour**
14: $R \leftarrow R + n$
15: $V \leftarrow V - n$
16: **fin tant que**
17: retourner R

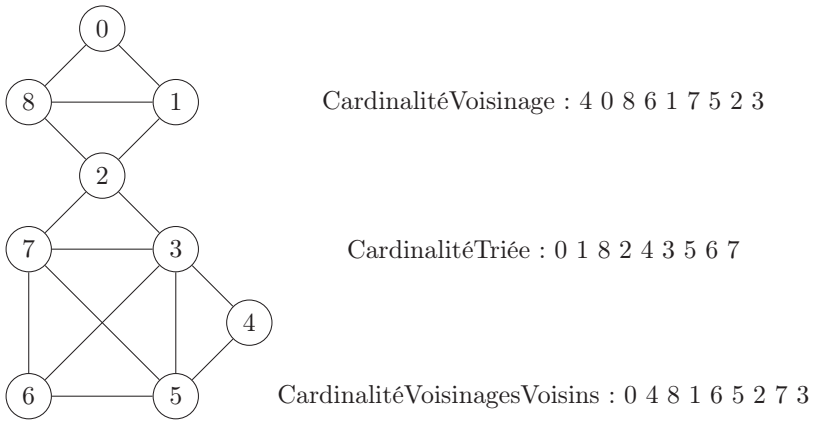


FIGURE 1.6 – Exemple d'ordonnancements de noeuds en fonction des voisinages.

Algorithme FMC

Pattabiraman et al. [88] proposent un algorithme exact, que nous nommerons *FMC* et une heuristique, nommée *FMC-H*, pour le problème MCP. L'idée est de proposer des méthodes d'élagage lors de l'exécution

de l'algorithme. Ces élagages permettent d'éviter d'explorer certaines branches de l'arbre d'exploration qui n'apporteraient pas de meilleures solutions. L'heuristique diffère de l'algorithme exact en ne prenant pas en compte la totalité des noeuds de l'ensemble des candidats P. En effet, l'heuristique considère uniquement le noeud de plus grand degré tandis que l'algorithme exact testera tous les candidats dont le degré est supérieur à la taille de la clique maximum trouvée jusque là. L'algorithme et sa sous-fonction sont présentés respectivement en algorithmes 5 et 6. FMC utilise 3 ensembles en plus de l'ensemble V des noeuds du graphe :

- Un ensemble de noeud C, initialement vide, qui contient à un instant t une clique partielle.
- Un ensemble K, initialement vide, qui permet de conserver en mémoire l'ensemble des noeuds composant la plus grande clique trouvée.
- Un ensemble de noeuds candidats P, qui est composé des noeud voisins de la clique partielle C (et dont le degré est supérieur à plus grande clique trouvée jusqu'alors, $\forall x \in P, \Gamma(x) > |K|$).

Algorithme 5 $FMC(V,K)$

```
1: pour tout u ∈ V faire
2:   si |Γ(u)| ≥ |K| alors
3:     C ← u
4:     P ← ∅
5:     pour tout v ∈ Γ(u) faire
6:       si v > u alors
7:         si |Γ(v)| ≥ |K| alors
8:           P ← P + v
9:         fin si
10:      fin si
11:    fin pour
12:    T ← CLIQUE(C,P,K)
13:    si |T| > |K| alors
14:      K ← T
15:    fin si
16:  fin si
17: fin pour
18: retourner K
```

Algorithme 6 $CLIQUE(C,P,K)$

```
1: tant que |C| + |P| > |K| faire
2:   prendre aleatoirement u ∈ P
3:   P ← P − u
4:   Q ← ∅
5:   pour tout v ∈ (Γ(u) ∩ P) faire
6:     si |Γ(v) ∩ (P ∪ C)| ≥ |K| alors
7:       Q ← Q + v
8:     fin si
9:     T ← CLIQUE(C + u, Q, K)
10:    si |T| > |K| alors
11:      K ← T
12:    fin si
13:  fin pour
14: fin tant que
15: retourner K
```

Les 5 élagages qui permettent d'éviter certaines explorations sont :

1. Algorithme 5, ligne 2 : On élimine les noeuds dont le degré est inférieur à la taille de la plus grande clique trouvée jusqu'alors.
2. Algorithme 5, ligne 6 : On élimine les doublons par l'élimination des voisins dont l'identifiant est plus grand que celui du noeud initial de la clique en cours.
3. Algorithme 5, ligne 7 : On élimine les voisins du noeud initial dont le degré est inférieur à la taille de la plus grande clique trouvée jusqu'alors.
4. Algorithme 6, ligne 1 : On continue l'agrandissement de la clique en cours uniquement si l'on a la possibilité de dépasser en taille la plus grande clique trouvée jusqu'alors.
5. Algorithme 6, ligne 6 : On élimine les voisins du dernier noeud ajouté dont le degré est inférieur à la taille de la plus grande clique trouvée jusqu'alors.

1.2.4 Énumération de cliques maximales

L'*énumération de cliques maximales* abrégée *MCE* pour l'anglais *Maximal Clique Enumeration* est un problème où le but est de lister l'ensemble des cliques maximales d'un graphe.

Ce problème est *NP-Difficile* [68].

Nous présenterons ici certains des principaux algorithmes trouvés dans la littérature.

Algorithme MCE

En 1973, Bron et Kerbosh proposent un algorithme exact que nous appellerons *BronKerbosh* [18]. L'idée repose sur l'agrandissement noeud par noeud d'une clique courante : lorsque l'on ne peut plus ajouter un noeud à l'ensemble représentant la clique courante, on ajoute cette dernière à la liste des cliques maximales. On revient ensuite sur une trace où l'on avait encore des noeuds candidats. BronKerbosh utilise 3 ensembles pour énumérer l'ensemble des cliques maximales :

— Un ensemble C, initialement vide, représentant les noeuds présents dans la clique courante.
— Un ensemble P, initialisé avec l'ensemble des noeuds du graphe V, représentant les noeuds candidats pour agrandir la clique courante à l'étape courante.
— Un ensemble D, initialement vide, représentant les noeuds déjà explorés à l'étape courante.

L'algorithme de Bron et Kerbosh est détaillé dans l'algorithme 7.

Algorithme 7 $BronKerbosh(C,P,D)$

1: **si** $P \cup D = \emptyset$ **alors**
2: ajouter C à la liste des cliques maximales
3: **fin si**
4: choisir aléatoirement $p \in (P \cup D)$
5: **pour tout** $u \in (P - \Gamma(p))$ **faire**
6: $P \leftarrow P - u$
7: $BronKerbosh(C + u,\ P \cap \Gamma(u),\ D \cap \Gamma(u))$
8: $D \leftarrow D + u$
9: **fin pour**

Afin d'éviter d'avoir à explorer la totalité des noeuds candidats P à chaque étape, les auteurs proposent de définir un noeud *pivot* à chaque étape d'agrandissement t de la clique. L'idée étant d'explorer, à l'étape t, uniquement le noeud pivot et ceux qui ne sont pas dans son voisinage au sein de l'ensemble des candidats P. En effet, les voisins du pivot seront conservés pour l'étape suivante lors du calcul de l'ensemble P pour les cliques composées de la clique courante C et du pivot p.

Algorithme de Tomita

Tomita et al. [111] proposent une amélioration aux travaux de Bron et Kerbosh par une meilleure sélection du pivot. La complexité de l'algorithme, que nous nommerons *Tomita*, est de l'ordre de $O(3^{(n/3)})$ [111], où n est le nombre de noeuds $|V|$ du graphe. Tomita utilise 3 ensembles :

— Un ensemble C, initialement vide, représentant la clique courante.
— Un ensemble S, initialisé avec l'ensemble des noeuds du graphe V, représentant les noeuds pouvant être utilisés comme pivot.
— Un ensemble P, initialisé avec l'ensemble des noeuds du graphe V, représentant les candidats pour agrandir la clique courante.

Le fonctionnement de Tomita est détaillé dans l'algorithme 8. Le résultat d'une exécution sur un graphe donné est représenté dans la figure 1.7.

Algorithme 8 *Tomita*(C,S,P)

1: **si** $S = \emptyset$ **alors**
2: ajouter C à la liste des cliques maximales
3: **fin si**
4: choisir $p \in S$ qui maximise la taille $|P \cap \Gamma(p)|$
5: **pour tout** $u \in (P - \Gamma(p))$ **faire**
6: $Tomita(C + u,\ S \cap \Gamma(u),\ P \cap \Gamma(u))$
7: $P \leftarrow P - u$
8: **fin pour**

La ligne 4 de l'algorithme 8 permet un élagage maximal en maximisant le nombre de noeuds éliminés à l'étape courante. Cet élagage est optimal car l'on maximise l'intersection entre le voisinage du pivot et les noeuds candidats.

Algorithme d'Eppstein

Eppstein et al. [29] ont proposé une variante de l'algorithme de Tomita. L'idée est de trier les noeuds du graphe G selon leur *ordre de dégénérescence* [31].

Définition 1.13. La dégénérescence d'un graphe $G(V,E)$ est le plus petit entier k de telle sorte que chaque sous-graphe $H \subset G$ contient un noeud de degré au plus k. On dit alors que le graphe est *k-dégénéré*.

Définition 1.14. L'ordre de dégénérescence d'un noeud du graphe correspond à un entier k de telle sorte que le nombre d'arêtes vers des noeuds de degrés plus petits est au plus k.

La dégénérescence d'un graphe peut être calculée en temps linéaire $O(m)$ où m est le nombre d'arêtes $|E|$ du graphe. L'algorithme d'énumération proposé sera nommé *Eppstein*. Il utilise l'ordre de dégénérescence des noeuds du graphe pour initialiser les ensembles S et P. Il utilise ensuite le même algorithme que Tomita. Son fonctionnement est présenté dans l'algorithme 9.

Algorithme 9 *Eppstein*(V)

pour tout $u_i \in V$, selon l'ordre de dégénérescence u_1,u_2,...,u_n **faire**
$Tomita(u_i,\ \Gamma(u_i) \cap \{u_{i+1},...,u_n\},\ \Gamma(u_i) \cap \{u_0,...,u_{i-1}\})$
fin pour

Les auteurs proposent deux versions de l'algorithme, selon si les données structurées de l'ordre de dégénérescence sont gardés ou non en mémoire.

1.3 Cliques dans un graphe compressé

Dans cette section, nous étudierons les problèmes de recherche de clique maximum et d'énumération de cliques maximales sur un graphe compressé par décomposition modulaire.

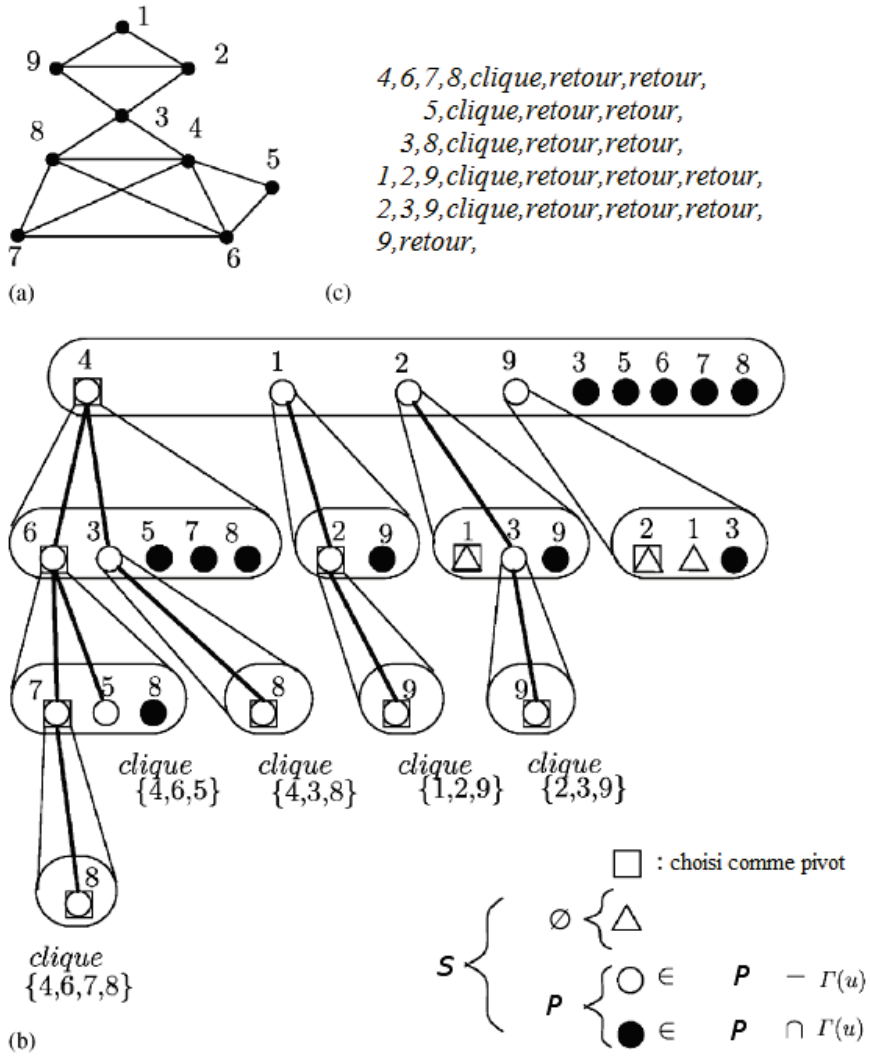


FIGURE 1.7 – Déroulement de l'algorithme de Tomita [111] avec le graphe en entrée (a), l'arbre d'exploration de l'algorithme (b) et le résultat donné par l'algorithme (c). Dans l'arbre d'exploration : un carré signifie que le noeud a été sélectionné comme pivot ; un triangle signifie une fin d'exploration sans clique maximale ; un cercle blanc correspond à un noeud exploré tandis qu'un cercle noir représente un noeud élagué car voisin du pivot.

1.3.1 Clique Maximum

Nous proposons un algorithme récursif d'exploration de l'arbre de décomposition modulaire qui retourne, pour chaque module, le plus grand des sous-ensembles complets qui le compose. Le traitement est différent pour chaque type de module :

Module Feuille

Les modules feuilles représentent des noeuds uniques dans le graphe initial, on retourne l'identifiant du noeud.

Module Série

Les modules séries représentent des sous-graphes composés d'un ensemble de modules complet. On retourne donc l'ensemble des modules fils. Un exemple est présenté en figure 1.8 :

Le graphe initial est présenté à gauche. La racine de l'arbre de décomposition modulaire est un module premier dont il est possible de voir le graphe compressé correspondant à droite. Le module de type série, lorsqu'il est évalué par l'algorithme de recherche de clique maximum, retourne un ensemble composé des modules feuilles 2, 3 et 5. Ainsi, la clique maximum du graphe compressé est de taille 5 : c'est la clique qui est composée des modules feuilles 1 et 4 et du module série *Ser* de cardinalité 3.

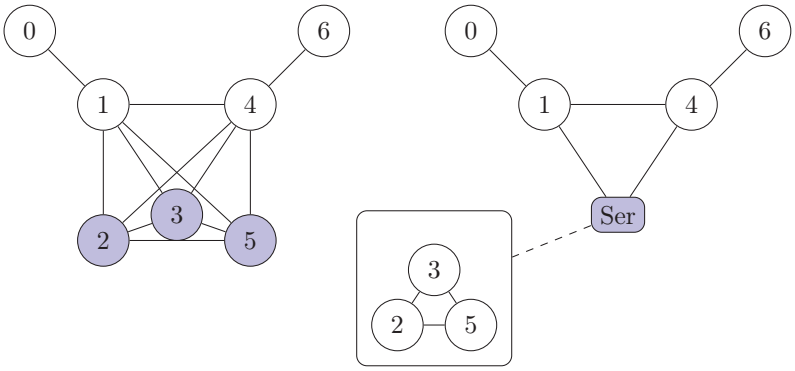


FIGURE 1.8 – Exemple du problème de clique maximum avec un module série sur un graphe compressé par décomposition modulaire.

Module Parallèle

Les modules parallèles représentent des sous-graphes composés d'un ensemble de modules indépendant. On retourne donc le plus grands ensemble de ses modules fils. Un exemple est présenté en figure 1.9 :

Le graphe initial est présenté à gauche. La racine de l'arbre de décomposition modulaire est un module premier dont il est possible de voir le graphe correspondant à droite. Le module de type série est composé de deux modules : un module feuille représentant le noeud 2 et un module série représentant un ensemble composé des modules feuilles 3 et 4. Lorsque la fonction de recherche de clique maximum interroge le module parallèle, ce dernier retourne le plus grand des sous-ensemble qui le compose. Dans ce cas, le sous-ensemble retourné est le module série composé des modules feuilles 3 et 4. Ainsi, la clique maximum du graphe compressé est de taille 3 : c'est la clique composée ou du module feuille 1 ou du module feuille 3 et du module parallèle (qui se reporte au plus grand sous-ensemble le composant : le module série composé des modules 3 et 4).

Module Premier

Les modules premiers représentent des sous-graphes composés d'un ensemble de modules qui ne peut pas être compressés. On leur adjoint en conséquence un ensemble d'arêtes liant les modules fils composant le module premier. Le sous-graphe induit par cet ensemble d'arêtes est alors recomposé et l'algorithme de recherche de clique fait appel à un algorithme de la littérature pour résoudre localement le problème de clique maximum. Un exemple est présenté en figure 1.10 :

Le graphe initial est présenté à gauche. L'arbre de décomposition modulaire est présenté à droite. La racine de l'arbre est un module série composé du module feuille 6 qui était un noeud universel [2] dans le graphe initial et d'un module premier. Lorsque l'algorithme de recherche de clique maximum arrive au niveau du module premier, il se sert de la liste d'arêtes C pour reconstruire un sous-graphe H. Ce sous-graphe est composé des modules feuilles 2, 3, 4 et 5 et du module parallèle composé des modules feuilles 0 et 1. On lance alors une recherche de clique maximum sur H qui nous retourne un ensemble composé des modules 3, 4 et 0 (ou

2. Un noeud universel u est un noeud possédant une arête commune avec l'ensemble des autres noeuds du graphe : soit u un noeud universel, $\forall v \in V - u, \exists (u,v) \in E$

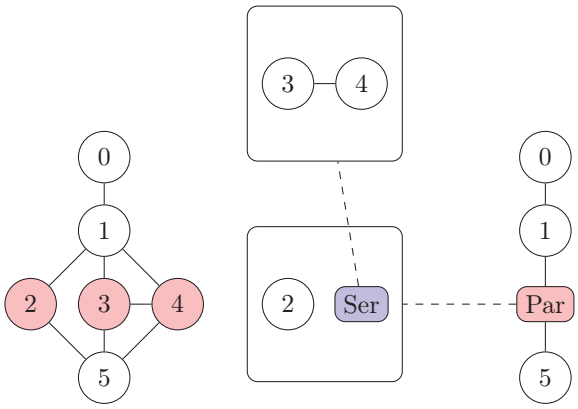


FIGURE 1.9 – Exemple du problème de clique maximum avec un module parallèle sur un graphe compressé par décomposition modulaire.

1). C'est cet ensemble qui est retourné par le module premier. Ainsi, dans le cadre de la figure 1.10, la plus grande clique trouvée est un ensemble de cardinalité 4 composé des noeuds 3, 4, 6 et 0 (ou 1).

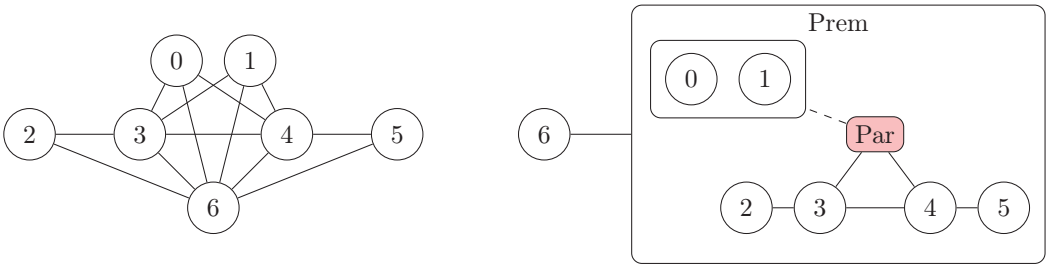


FIGURE 1.10 – Exemple du problème de clique maximum avec un module premier sur un graphe compressé par décomposition modulaire.

Exploration de l'arbre de décomposition modulaire

Le fonctionnement de la recherche de la clique maximum se base sur une exploration de l'arbre à partir de la racine et en descendant dans ses modules fils (en appliquant récursivement la fonction). De plus, pour les modules premiers, on applique un algorithme de recherche de clique maximum sur le sous-graphe induit H. Cet algorithme, tiré de la littérature, est ajusté pour prendre en compte la pondération des noeuds composant le graphe H.

Le déroulement est présenté en algorithme 10. Il utilise deux entrées :

- L'arbre de décomposition modulaire A, composé des sous-ensembles :
  - $A.M$ représentant les modules
  - $A.E_A$ les arêtes de l'arbre (représentant les relations père-fils)
  - $A.f$ une fonction attribuant à chaque module feuille de $A.M_f$ l'identifiant du noeud dans le graphe initial
  - $A.C$ les arêtes liants les modules fils de chaque module premier
- La racine r de l'arbre, ou du sous-arbre local, initialisée avec la racine de l'arbre de décomposition modulaire

Algorithme 10 $CliqueMaximumModulaire(A,r)$

1: **si** $r \in A.M_f$ **alors**
2: retourner l'ensemble composé du noeud $A.f(r)$
3: **sinon**
4: $K \leftarrow \emptyset$
5: **pour tout** $v \in M$, $(r,v) \in A.E_A$ **faire**
6: $K \leftarrow K \cup CliqueMaximumModulaire(A,v)$
7: **fin pour**
8: **si** $r \in A.M_{ser}$ **alors**
9: retourner l'union de tous les sous-ensembles de K
10: **fin si**
11: **si** $r \in A.M_{par}$ **alors**
12: retourner le plus grand des sous-ensembles de K
13: **fin si**
14: **si** $r \in A.M_{PREM}$ **alors**
15: construire le graphe $H(K,\ A.\ C(r),\ w)$ où
- K est un ensemble de noeuds représentant les retours des modules fils de r
- $A.C(r)$ est l'ensemble des arêtes liant les noeuds de l'ensemble K
- w est une fonction de pondération. Elle attribue à chaque élément u de K un poids correspondant à la cardinalité de l'ensemble qui est représenté par u.

16: retourner l'ensemble $CliqueMaximum(H)$
17: **fin si**
18: **fin si**

Exemple d'exploration d'un arbre de décomposition modulaire pour la clique maximum

Nous proposerons dans cette section un exemple du déroulement de l'exploration de l'arbre de décomposition modulaire pour le problème de recherche de la clique maximum. Nous travaillerons sur le graphe initial présenté en figure 1.11 et son arbre de décomposition modulaire présenté en figure 1.12, où l'identifiant des modules est noté en rouge.

L'algorithme de recherche part de la racine de l'arbre, qui est un module parallèle. On va donc appeler, pour chacun des modules fils (qui sont tous deux des modules premiers), l'algorithme de recherche avec comme racine les modules premiers 1 et 10.

Pour le module premier 1, on calcule pour chacun de ses modules fils les retours de ces derniers (ligne 6 de l'algorithme 10). Soit l'ensemble {10} pour le module 6, l'ensemble {8,11} pour le module 7, l'ensemble {9} (ou {12}) pour le module 3 et enfin l'ensemble {13} pour le module 2. Pour le module premier 1, le retour sera composé de la clique locale formée des modules 6 et 7 (ou 3 et 7) qui représentent des cliques de cardinalité 3. Ce sera donc une clique formée des noeuds {8,10,11}, {8,9,11} ou {8,11,12}.

Pour le module premier 10, on calcule les retours de ses modules fils. Leurs retours respectifs sont les ensembles {0} pour le module 16, {1,2} pour le module 17, {3} pour le module 15, {12,13,14} pour le module 11 et {4} pour le module 20. Le retour du module premier 10 sera donc composé de la clique locale formée par les modules 11 et 15 (ou 11 et 20). Ces cliques locales représentent des cliques de cardinalité 4 dans le graphe initial, à savoir les cliques {3,5,6,7} ou {4,5,6,7}.

Ainsi le module parallèle 0 retournera le plus grand des sous-ensembles qui lui est retourné parmi chacun des modules premiers. Dans cette situation, c'est le sous-ensemble de cardinalité 4 issu du module premier 10.

Heuristiques pour la clique maximum sur un graphe compressé par décomposition modulaire

Nous proposons également 2 heuristiques qui n'évaluent pas l'ensemble des cliques du graphe mais qui visent à trouver une réponse acceptable tout en diminuant le temps de calcul. Elles fonctionnent sur une variation du fonctionnement des recherches de cliques dans les sous-graphes induits par les modules premiers. L'idée est d'élaguer les voisinages des cliques en construction en fonction de la taille des ensembles retournés par les modules fils.

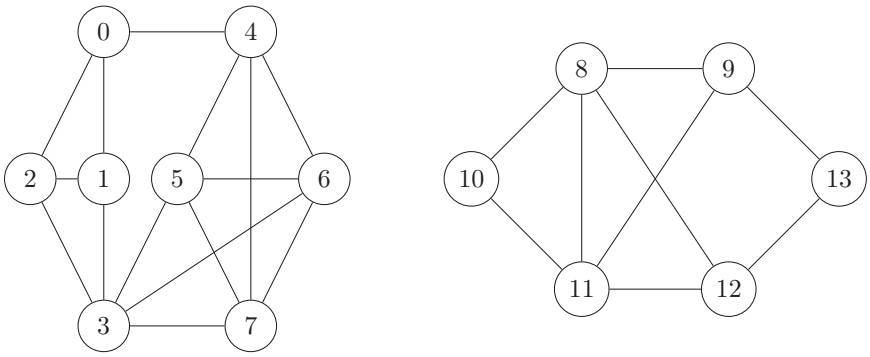


FIGURE 1.11 – Graphe initial pour la recherche de clique maximum par décomposition modulaire.

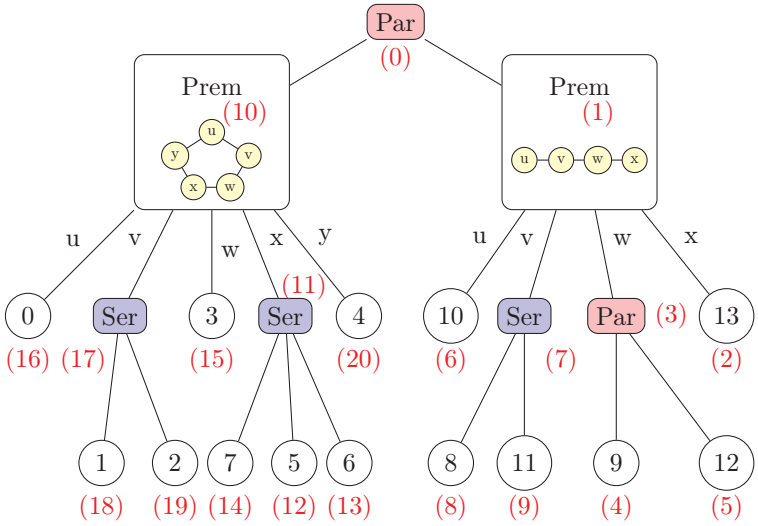


FIGURE 1.12 – Arbre de décomposition modulaire pour la recherche de clique maximum par décomposition modulaire.

La première heuristique, $CliqueMaximumModulaire_{h1}$, ajoute à la clique locale uniquement le module voisin de la clique qui maximise l'ensemble de ses descendants feuilles et de ceux de ses voisins.

La seconde heuristique, $CliqueMaximumModulaire_{h2}$, ajoute à la clique le module voisin de la clique qui a le plus grand nombre de descendants feuilles.

Un exemple est donné en figure 1.13. On considère le sous-graphe induit présenté : les noeuds représentent chacun un module. Le nombre inscrit dans le noeud correspond à la taille du sous-ensemble retourné par le module. On se place dans la situation où l'on est en train de construire une quasi-clique à partir des modules coloriés en vert et que l'on considère les voisins de cette dernière (les noeuds bleus et rouges). La première heuristique sélectionnera le module bleu clair pour agrandir la clique à la prochaine étape, tandis que la seconde sélectionnera le module rouge. Ainsi, la première heuristique retournera un ensemble de taille 10, tandis que la seconde retournera un ensemble de taille 9.

1.3.2 Énumération de cliques maximales

Nous proposons un algorithme récursif d'exploration de l'arbre de décomposition modulaire qui retourne, pour chaque module, un ensemble composé de chacun des sous-ensembles complets qui composent le module. Le traitement est différent pour chaque type de module :

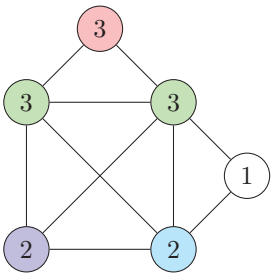


FIGURE 1.13 – Sélection de nouveaux noeuds par les heuristiques pour le problème de recherche de clique maximum.

Module Feuille

Les modules feuilles représentent des noeuds uniques dans le graphe initial, on retourne un ensemble composé de l'identifiant du noeud.

Module Série

Les modules séries représentent des sous-graphes composés d'un ensemble de modules complet. On retourne donc l'ensemble des modules fils. Un exemple est présenté en figure 1.14 :

Le graphe initial est présenté à gauche. La racine de l'arbre décomposition modulaire est un module premier dont il est possible de voir le graphe compressé correspondant à droite. Le module de type série, lorsqu'il est évalué par l'algorithme d'énumération de cliques maximales, retourne un ensemble composé des modules feuilles 0, 1, 2 et 3. Ainsi, les cliques maximales du graphe compressé seront les cliques {5,6}, {4,5} et {0,1,2,3,4}.

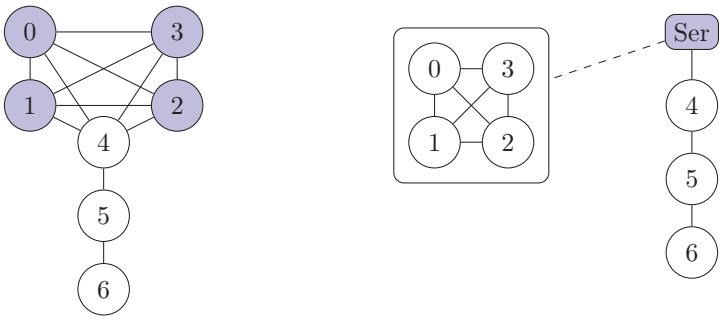


FIGURE 1.14 – Exemple du problème d'énumération de cliques maximales avec un module série sur un graphe compressé par décomposition modulaire.

Module Parallèle

Les modules parallèles représentent des sous-graphes composés d'un ensemble de modules indépendant. On retourne donc des sous-ensembles correspondant à chacun des retours des modules fils. Un exemple est présenté en figure 1.15 :

Le graphe initial est présenté à gauche. La racine de l'arbre décomposition modulaire est un module premier dont il est possible de voir le graphe compressé correspondant à droite. Le module de type parallèle, lorsqu'il est évalué par l'algorithme d'énumération de cliques maximales, retourne deux ensembles composés respectivement des modules feuilles 1 et 2, et 3 et 4. Ainsi, les cliques maximales du graphe compressé seront les cliques {5,6}, {0,1,2}, {0,3,4}, {5,1,2} et {5,3,4}.

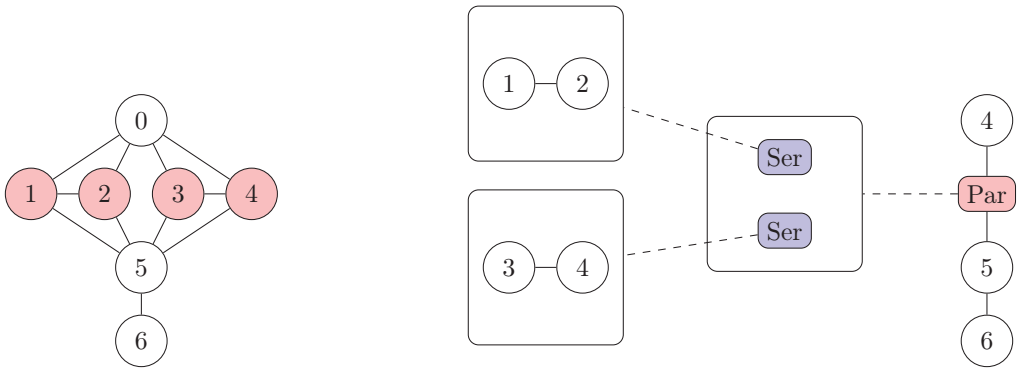


FIGURE 1.15 – Exemple du problème d'énumération de cliques maximales avec un module parallèle sur un graphe compressé par décomposition modulaire.

Module Premier

Les modules premiers représentent des sous-graphes composés d'un ensemble de modules qui ne peut pas être compressé. On leur adjoint en conséquence un ensemble d'arêtes liant les modules fils composant le module premier. Le sous-graphe induit par cet ensemble d'arêtes est alors recomposé et l'algorithme de recherche de clique fait appel à un algorithme de la littérature pour résoudre localement le problème d'énumération de cliques maximales. Un exemple est présenté en figure 1.16 :

Le graphe initial est présenté à gauche. L'arbre de décomposition modulaire est présenté à droite. La racine de l'arbre est un module série composé d'un module premier et du module feuille 4 (qui était un noeud universel dans le graphe initial). Lorsque l'algorithme d'énumération de cliques maximales arrive au niveau du module premier, il se sert de la liste d'arêtes C pour reconstruire un sous-graphe H composé des modules feuilles (2, 3, 5 et 6) et du module parallèle composé des modules feuilles 0 et 1. On lance alors une énumération de cliques maximales sur H qui retourne des sous-ensembles composés respectivement des modules {2,3}, {3,5}, {5,6}, {0,2,6} et {1,2,6}. Ces ensembles sont retournés par le module premier. Ainsi, dans le cadre de la figure 1.16, les cliques maximales énumérées sont les cliques{2,3,4}, {3,4,5}, {4,5,6}, {0,2,4,6} et {1,2,4,6}.

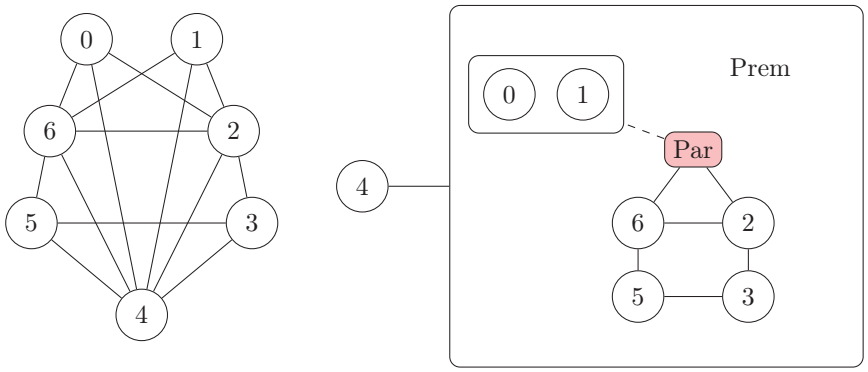


FIGURE 1.16 – Exemple du problème d'énumération de cliques maximales avec un module premier sur un graphe compressé par décomposition modulaire.

Exploration de l'arbre de décomposition modulaire

Le fonctionnement de l'énumération de cliques maximales se base sur une exploration de l'arbre à partir de la racine et en descendant dans ses modules fils (en appliquant récursivement la fonction). De plus, pour

les modules premiers, on applique, sur le sous-graphe induit H, un algorithme d'énumération de cliques maximales de la littérature. Cet algorithme est ajusté pour prendre en compte les retours d'ensembles des noeuds composants le graphe H.

Le déroulement est présenté en algorithme 11. Il utilise deux entrées :

- L'arbre de décomposition modulaire A, composé des sous-ensembles :
  - $A.M$ représentant les modules
  - $A.E_A$ les arêtes de l'arbre (représentant les relations père-fils)
  - $A.f$ une fonction attribuant à chaque module feuille de $A.M_f$ l'identifiant du noeud dans le graphe initial
  - $A.C$ les arêtes liants les modules fils de chaque module premier
- La racine r de l'arbre, ou du sous-arbre local, initialisée avec la racine de l'arbre de décomposition modulaire

Algorithme 11 $CliquesMaximalesModulaire(A,r)$

1: **si** $r \in A.M_f$ **alors**
2: retourner l'ensemble composé du noeud $A.f(r)$
3: **sinon**
4: $K \leftarrow \emptyset$
5: **pour tout** $v \in M$, $(r,v) \in E_A$ **faire**
6: $K \leftarrow K \cup CliqueMaximumModulaire(A,v)$
7: **fin pour**
8: **si** $r \in A.M_{ser}$ **alors**
9: retourner la distributivité de tous les sous-ensembles de K
10: **fin si**
11: **si** $r \in A.M_{par}$ **alors**
12: retourner K
13: **fin si**
14: **si** $r \in A.M_{PREM}$ **alors**
15: construire le graphe $H(K,\ A.\ C(r),\ w)$ où
- K est un ensemble de noeuds représentant les retours des modules fils de r
- $A.C(r)$ est l'ensemble des arêtes liant les noeuds de l'ensemble K
- w est une fonction de pondération attribuant à chaque élément u de K un poids correspondant à la cardinalité de l'ensemble représenté par u.

16: retourner l'ensemble de $CliqueMaximales(H)$ composés de toutes les cliques maximales du sous-graphe H
17: **fin si**
18: **fin si**

La *distributivité* utilisée en ligne 9 de l'algorithme 11 correspond à une opération visant à regrouper 2 à 2 chacun des sous-ensembles fils d'enfants parents. Ainsi, deux ensembles X et Y, composés respectivement des sous-ensembles $\{X_1,X_2\}$ et $\{Y_1,Y_2\}$ se verront distribués en 4 ensembles $\{X_1,Y_1\}$, $\{X_1,Y_2\}$, $\{X_2,Y_1\}$ et $\{X_2,Y_2\}$.

Exemple d'exploration d'un arbre de décomposition modulaire pour l'énumération de cliques maximales

Nous proposerons dans cette section un exemple du déroulement de l'exploration de l'arbre de décomposition modulaire pour le problème d'énumération de cliques maximales. Nous travaillerons sur le graphe initial présenté en figure 1.17 et son arbre de décomposition modulaire présenté en figure 1.18, où l'identifiant des modules est noté en rouge.

L'algorithme de recherche part de la racine de l'arbre, qui est un module premier. On va donc appeler, pour chacun des modules fils, l'algorithme d'énumération avec comme racine les modules 1, 4, 5, 8 et 9.

Les modules 4 et 8 sont des modules feuilles qui retournent respectivement les noeuds 2 et 4. Les modules séries 1 et 5 retournent respectivement les ensembles de noeuds {0,1} et {3,5}. Le module série 9 est composé du module feuille 10 qui retourne le noeud 7 et du module parallèle 11 qui retourne les sous-ensembles de noeuds {{6},{8}}. Avec l'opération de distributivité, le module série 9 retourne donc les sous-ensembles de noeuds {{6,7},{7,8}}.

Pour le module premier 0, l'algorithme d'énumération de cliques sur le sous-graphe induit retourne les cliques composés des modules-fils {1,4}, {4,5,8} et {8,9}. En prenant en compte les retours des modules-fils, on se retrouve avec les cliques composés des noeuds {0,1,2}, {2,3,4,5}, {4,6,7} et {4,7,8}.

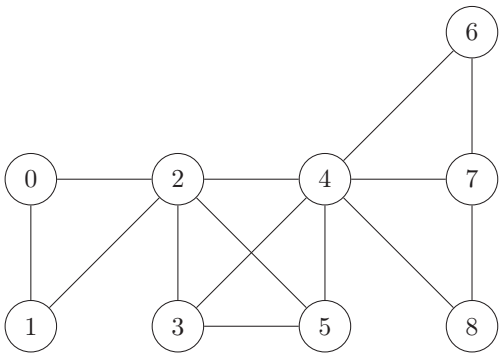


FIGURE 1.17 – Graphe initial pour l'énumération de cliques maximales par décomposition modulaire.

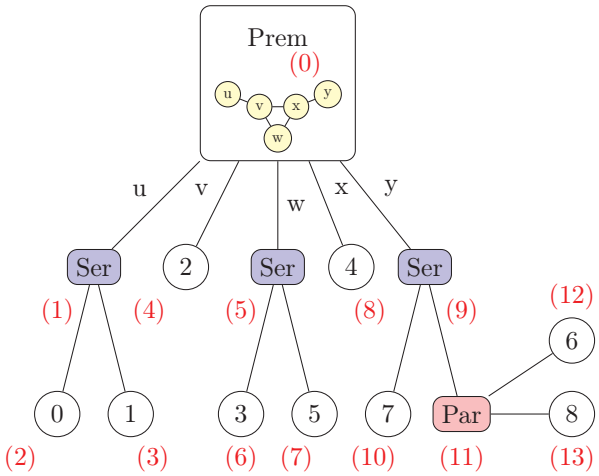


FIGURE 1.18 – Arbre de décomposition modulaire pour l'énumération de cliques maximales par décomposition modulaire.

1.4 Expérimentations

Dans cette section, nous décrirons 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. Les algorithmes ont été implémentés en C++ à l'aide de la libraire SNAP [71, 103]. 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 16G.

Notre algorithme de compression est basé sur l'algorithme de décomposition modulaire présenté par Habib et al. [49] 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 [24] du graphe [23] grâce à une technique d'affinage des partitions [51].
2. La seconde construit l'arbre de décomposition modulaire à l'aide la permutation issue du premier algorithme [12].

Pour construire l'ensemble C des arêtes liants les modules-fils d'un module premier, nous avons ajouté un troisième algorithme. Cet algorithme teste, pour chacun des descendants d'un module premier, s'il existe une arête entre les noeuds qui les composent. Dans ce cas, une arête liant les modules parents est ajoutée à l'ensemble C. Le déroulement est présenté dans l'algorithme 12. Il utilise 3 entrées :

— L'arbre de décomposition modulaire A dont le sous-ensemble C est vide
— La racine de l'arbre de décomposition modulaire
— Le graphe initial $G(V,E)$

La ligne 5 de l'algorithme vise à éviter les doublons en comparant les identifiants des modules. La ligne 6 permet de regarder si, pour n'importe quel descendant feuille de u et v, les noeuds correspondant aux modules feuilles sont bien voisins dans le graphe initial.

Algorithme 12 $ConstructionVoisinageModule(A,r,G)$

```
1:  pour tout u ∈ M,(r,u) ∈ A.E_A faire
2:    ConstructionVoisinageModule(A,u,G)
3:    si r ∈ A.M_PREM alors
4:      pour tout v ∈ M,(r,v) ∈ A.E_A faire
5:        si u < v alors
6:          si (A.f(u),A.f(v)) ∈ G.E alors
7:            A.C ← A.C + (u,v)
8:          fin si
9:        fin si
10:     fin pour
11:   fin si
12: fin pour
```

Afin d'évaluer la qualité de notre approche, nous avons implémenté différentes versions des algorithmes.

Pour le problème de clique maximum, nous avons implémenté :

— MC et les variantes MCQ, MCS_a, MCS_b et $BBMC$. Pour les variantes, nous avons testé les 3 algorithmes de colorations $CardinalitéVoisinagesVoisins$, $CardinalitéVoisinagesVoisins$ et $CardinalitéVoisinagesVoisins$ que nous numéroterons de 1 à 3.
— FMC et son heuristique FMC_h.
— $CliqueMaximumModulaire$, que nous noterons CMM, qui utilise FMC pour le calcul local de clique sur les modules premiers et deux heuristiques CMM_{h1} et CMM_{h2}.

Pour le problème d'énumération de cliques maximales, nous avons implémenté :

— $BronKerbosh$
— $Tomita$
— $Eppstein$ et sa version sans structure $Eppstein_{hybrid}$
— $CliquesMaximalesModulaire$ qui utilise $Tomita$ pour l'énumération locale de cliques maximales sur les modules premiers

1.4.1 Graphes de test

Nous avons testé les différentes méthodes sur plusieurs graphes présentés dans le tableau 1.2. Ces graphes sont de différents types et ont été pris dans les bases de données des projets SNAP [70], DIMACS [28] et NetworkRepository [93].

Chacun des graphes est présenté en version simplifiée, les boucles étant supprimées et les arêtes dirigées étant considérées comme des arêtes non-dirigées.

Nom	**|V|**	**|E|**	$deg_{moy}(G)$	$\Delta(G)$	$\delta(G)$
celegans	453	2 025	8,94	237	1
c-fat500-5	500	23 191	92,76	95	92
c-fat500-10	500	46 627	186,51	188	185
coPapersDBLP	26 836	45 647	3,40	27	1
com-dblp	317 080	1 049 866	6,62	343	1
Hook_1498	59 826	1 146 657	38,33	77	2
notreDame	325 729	1 090 108	6,69	10 721	1
PR02R	40 523	994 913	49,10	87	1
rattus	8 763	39 932	9,11	334	1
worm	3 507	6 531	3,72	523	1

Tableau 1.2 – Graphe de données pour les tests sur la recherche de cliques dans des graphes compressés.

Graphes Compressés

Nous avons compressé chacun de ces graphes par décomposition modulaire. Les statistiques de ces compressions sont présentées dans le tableau 1.3 où :

— temps : représente le temps nécessaire pour compresser le graphe et obtenir l'arbre de décomposition modulaire correspondant
— $|A.C|$: le nombre d'arêtes liant les modules fils de chacun des modules premiers du graphe
— te_c : le taux de compression des arêtes, $te_c = 1 - \frac{|A.C|}{|E|}$
— $|V_{Pc}|$: le nombre de noeuds présents dans le graphe premier aîné P
— tv_{Pc} : le taux de compression des noeuds présents dans le graphe premier aîné, $1 - \frac{|V_{Pc}|}{|V|}$
— $|E_{Pc}|$: le nombre de noeuds présents dans le graphe premier aîné P
— te_{Pc} : le taux de compression des arêtes présentes dans le graphe premier aîné, $1 - \frac{|E_{Pc}|}{|E|}$

Le graphe premier aîné est le graphe construit à partir des modules premiers rencontrés en premier lors d'un parcours de l'arbre à partir de la racine : ce sont des modules premiers qui ne sont pas des descendants de modules premiers. Un exemple est donné où l'on voit un graphe premier aîné en figure 1.20 issu de l'arbre présenté en figure 1.19. Le module premier représenté par le noeud 9 n'est pas pris en compte pour le graphe premier aîné car il a comme ancêtre un module premier.

Dans le cas présenté :

— $|A.C| = 16$
— $te_c = 1 - \frac{16}{75} = 78{,}6\%$
— $|V_{Pc}| = 13$
— $tv_{Pc} = 1 - \frac{13}{17} = 23{,}5\%$
— $|E_{Pc}| = 11$
— $te_{Pc} = 1 - \frac{11}{75} = 85{,}3\%$

1.4.2 Résultats

Dans cette section, nous évaluons nos algorithmes sur les graphes compressés par décomposition modulaire. Nous comparons les résultats obtenus à ceux obtenus par les méthodes présentés dans la littérature.

Pour la clique maximum

Les résultats pour le problème de la clique maximum sont présentés dans le tableau 1.4. Les résultats de chacun des algorithmes sont présentés sur deux lignes :

1. La première ligne donne la taille de la plus grande clique trouvée par l'algorithme
2. La seconde donne le temps d'exécution de l'algorithme

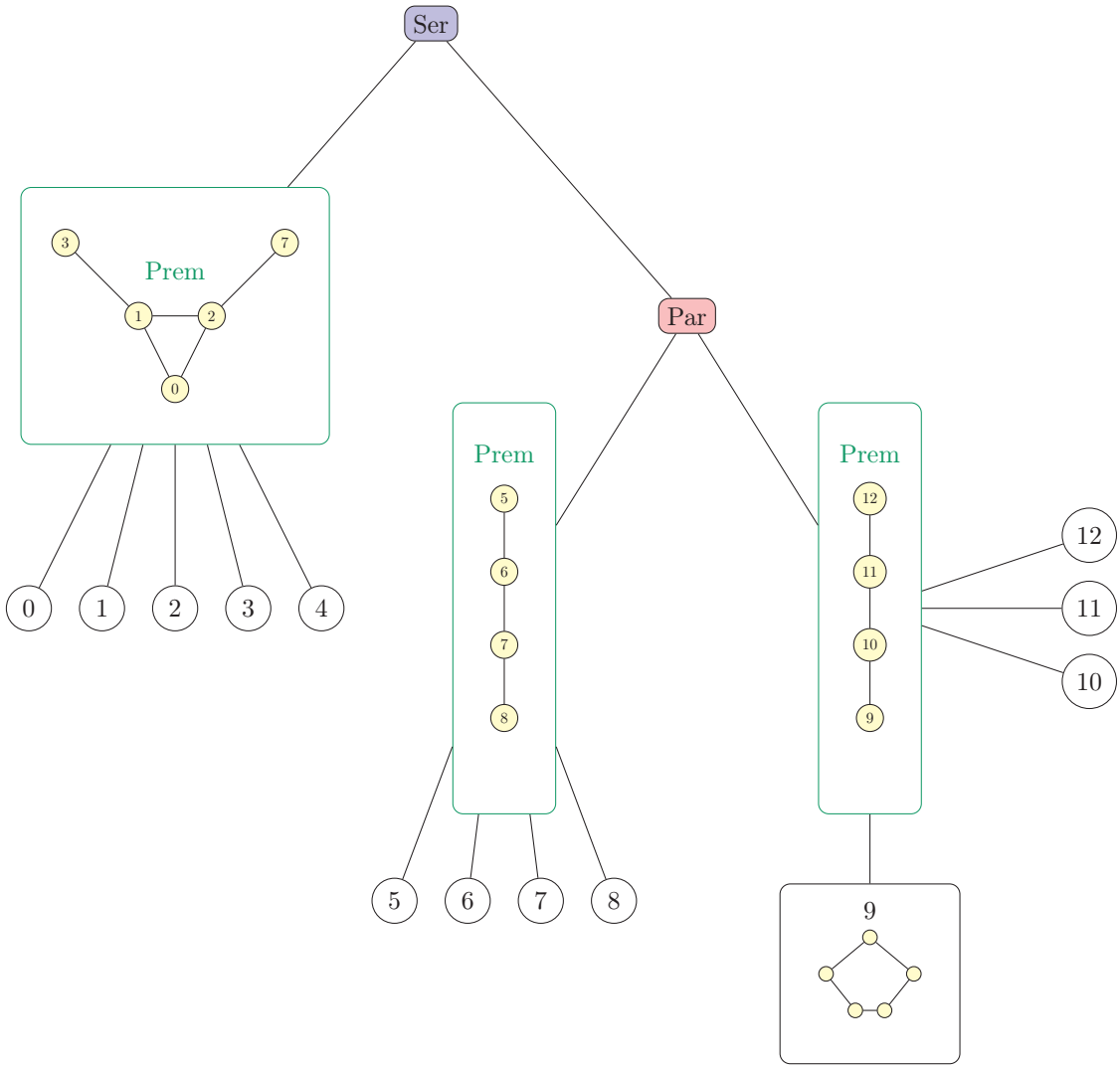


FIGURE 1.19 – Arbre de décomposition modulaire utilisé en exemple pour montrer un graphe premier aîné.

Nom du graphe	**temps (s)**	$\|A.C\|$	$te_c(\%)$	$\|V_{Pc}\|$	$tv_{Pc}(\%)$	$\|E_{Pc}\|$	$te_{Pc}(\%)$
celegans	0,002356	1 779	12,14	413	8,83	1 779	12,14
c-fat500-5	0,007571	16	99,93	16	96,8	16	99,93
c-fat500-10	0,015463	8	99,98	8	98,4	8	99,98
coPapersDBLP	0,467879	3 442	92,45	24 862	7,35	3 442	92,45
com-dblp	1 229	730 952	30,38	223 071	29,65	730 395	30,43
Hook_1498	29,24	120 753	89,46	19 945	66,66	120 753	89,46
notreDame	1 214	769 736	29,38	136 878	57,97	764 693	29,85
PR02R	30,08	425 853	57,19	28 519	29,62	425 853	57,19
rattus	0,555856	37 279	6,64	7 423	15,29	37 279	6,64
worm	0,047741	4 957	24,1	2 128	39,32	4 957	24,1

Tableau 1.3 – Statistiques de la compression de graphe par décomposition modulaire.

Dans le tableau 1.4, les meilleurs résultats sont ceux qui trouvent la plus grande clique en moins de temps

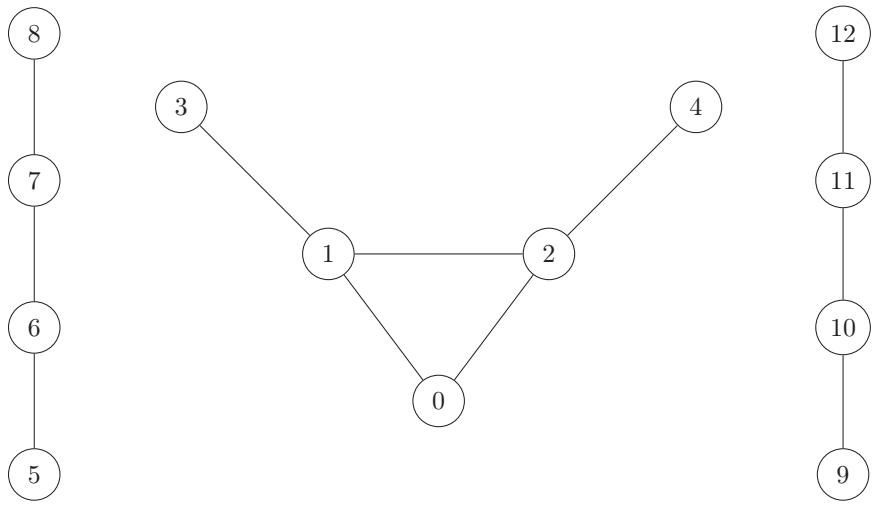


FIGURE 1.20 – Graphe premier aîné calculé à partir d'un arbre de décomposition modulaire.

que les autres. Ces résultats sont présentés en bleu dans le tableau.

La comparaison entre notre algorithme CMM et FMC est détaillée dans la figure 1.21. Nous notons qu'à part pour les graphes Dimacs (les graphes *c-fat500-5* et *c-fat500-10*) qui ont un fort taux de compression, notre algorithme ne donne pas de meilleurs résultats.

Nous notons également que lorsque l'heuristique de FMC_h met plus de temps que l'algorithme FMC, les heuristiques sur le graphe compressé CMM_{h1} et CMM_{h2} mettent également plus de temps que l'algorithme CMM.

En conclusion, nous notons que l'adaptation d'algorithmes pour calculer la taille de la clique maximum sur un graphe compressé par décomposition modulaire n'offre pas d'avantages. Ce résultat s'explique par la nécessité de tester l'ensemble des noeuds du graphe compressé sans distinction de degrés. En effet, tel que présenté dans la figure 1.22, il faut prendre en compte le noeud 4 dans le graphe compressé (qui est en fait une clique de taille 5) afin de trouvé la clique de taille maximale. Cependant, dans le graphe initial, les noeuds composants le noeud 4 auraient été explorés en premier par l'algorithme FMC car ce sont ceux de plus haut degrés dans le graphe.

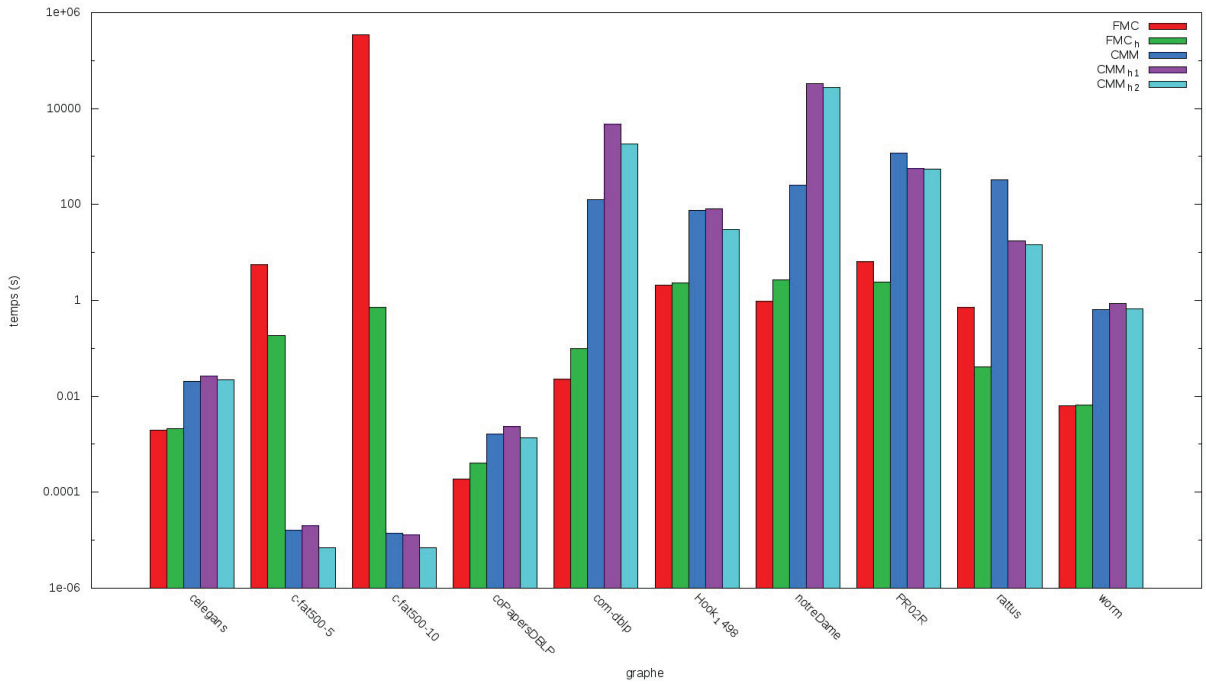


FIGURE 1.21 – Résultat temporels pour la recherche de clique maximum sur des graphes compressés.

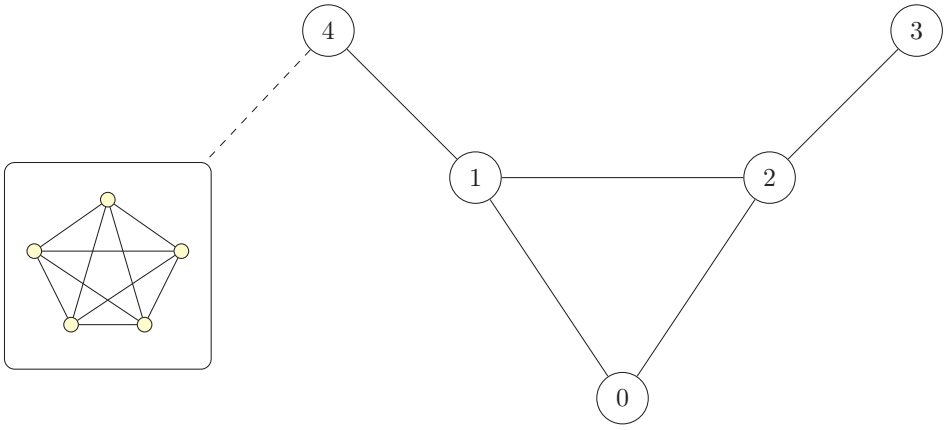


FIGURE 1.22 – Exemple du problème de clique maximum nécessitant l'exploration des noeuds de petits degrés pour trouver la clique de plus grande taille dans un graphe compressé par décomposition modulaire.

Pour l'énumération de cliques maximales

Les résultats pour le problème de la clique maximum sont présentés dans le tableau 1.5. Pour chaque graphe, nous donnons le nombre de cliques trouvées par les algorithmes dans la seconde colonne $|\mathbf{K}|$. Ensuite, pour chacun des algorithmes, nous donnons le temps nécessaire à l'énumération. Le meilleur score est donné en bleu.

Nous remarquons que généralement, l'énumération de cliques donne de meilleurs résultats sur les graphes compressés. Il est intéressant de noter que dans certains cas (pour les graphes *coPapersDBLP*, *com-dblp*, *no-*

treDame et *worm*), l'algorithme *MCE* donne de meilleurs résultats que l'algorithme *Tomita*. En conséquence, il pourrait être intéressant d'adapter *MCE* sur des graphes compressés.

La comparaison entre notre algorithme *TomitaComp* et *Tomita* est détaillée dans la figure 1.23. Nous avons également ajouté une mesure supplémentaire prenant en considération le temps de compression du graphe (en plus du temps mis par notre algorithme) *Comp + TomitaComp*.

Nous notons que bien que notre algorithme donne généralement de meilleurs résultats que *Tomita*, le temps de compression n'est pas négligeable. En conséquence, lorsque ce temps de compression est ajouté au temps mis par l'algorithme, les résultats deviennent moins bons (c'est notamment le cas avec les graphes *celegans*, *com-dblp* et *notreDame*).

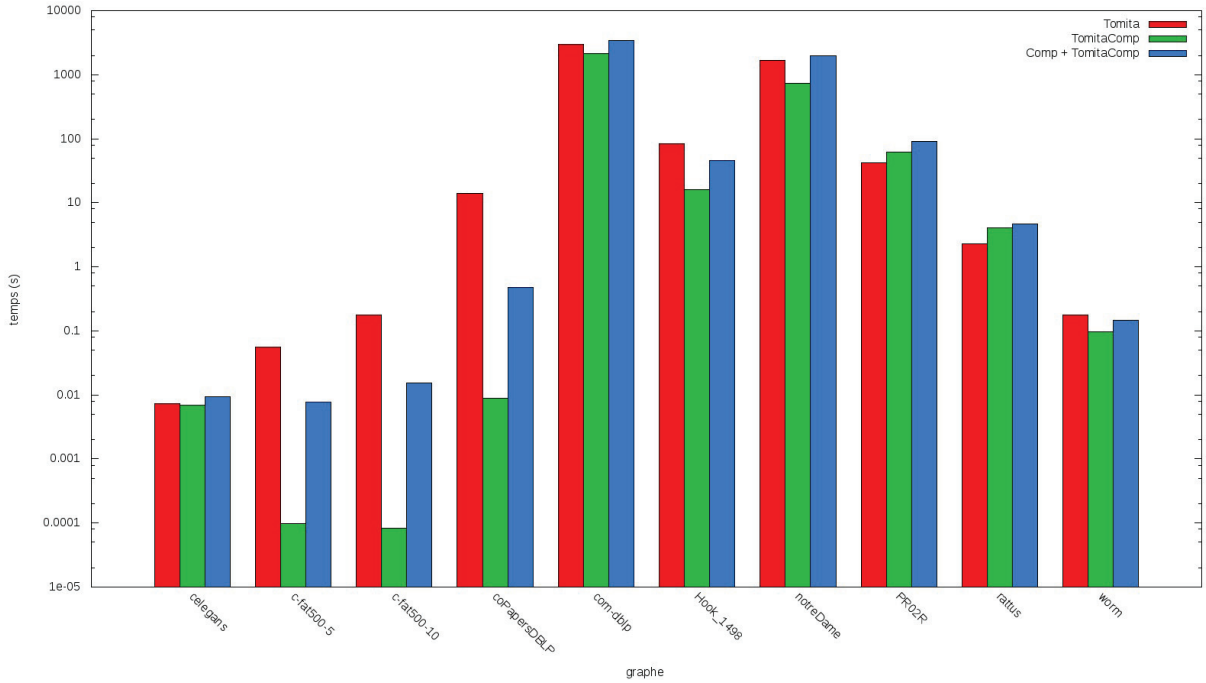


FIGURE 1.23 – Résultat temporels pour l'énumération de cliques maximales sur des graphes compressés.

Afin de collecter plus d'informations, nous avons réalisés des tests supplémentaires pour l'énumération de cliques sur des graphes plus importants. Ces graphes sont présentés dans le tableau 1.6. Les compressions sont présentées dans le tableau 1.7.

Les résultats de l'exécution des algorithmes sont présentés dans le tableau 1.8 et la figure 1.24. Pour ces graphes plus importants, on remarque que le temps d'exécution de l'algorithme sur le graphe compressé donne généralement de meilleurs résultats. Cependant, si l'on prend en considération le temps nécessaire pour compresser les graphes, le temps total devient plus important que celui de l'algorithme sur le graphe non compressé.

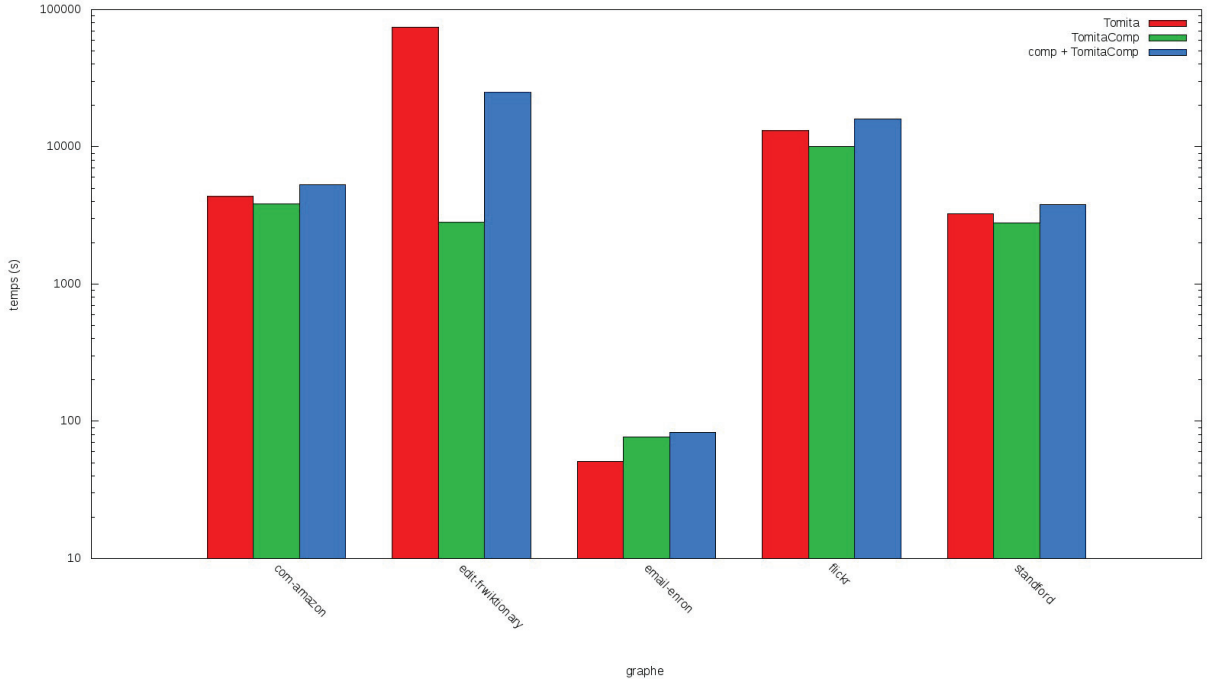


FIGURE 1.24 – Résultats temporels pour l'énumération de cliques maximales sur des graphes compressés.

Algorithme	celegans	c-fat500-5	c-fat500-10	coPapersDBLP	com-dblp	Hook_1498	notreDame	PR02R	rattus	worm
MC	9 0,001883	64 242 766		27 3,253443	114 652	18 18,497061	155 713	23 10,144663	23 0,707162	7 0,114814
$MCQ1$	9 0,003220	64 0,366825	126 1,038696	27 1,876385	114 257,297729	18 36,064568	155 285,150726	23 24,105680	23 0,308356	7 0,054007
$MCQ2$	9 0,003581	64 0,091955	126 1,024399	27 1,620754	114 200,770020	18 31,760515	155 228,324905	23 29,038830	23 0,179666	7 0,052329
$MCQ3$	9 0,003003	64 0,286897	126 1,039571	27 2,167245	114 273,844330	18 38,224083	155 305,081421	23 25,756594	23 0,274418	7 0,065158
MCS_A1	9 0,003023	64 0,343218	126 1,380591	27 1,887291	114 257,345856	18 52,096046	155 285,608215	23 32,332821	23 0,302628	7 0,062424
MCS_A2	9 0,002771	64 0,113196	126 1,360014	27 1,624450	114 200,752747	18 41,754349	155 228,138489	23 36,686756	23 0,176908	7 0,050139
MCS_A3	9 0,004367	64 0,275442	126 1,376538	27 2,175131	114 274,010010	18 55,310303	155 302,834656	23 33,392593	23 0,273256	7 0,066771
MCS_B1	8 0,003784	63 0,035534	126 0,217980	8 2,055029	26 259,428711	12 75,726562	17 298,200073	19 8,473679	23 0,278088	7 0,047455
MCS_B2	8 0,003028	63 0,031675	126 0,200554	6 1,951442	18 202,394302	12 68,187889	40 248,880890	19 10,200466	13 0,179533	7 0,047195
MCS_B3	8 0,002440	63 0,035838	126 0,217808	8 2,373556	26 276,773438	12 77,770447	17 317,739014	16 19,327188	22 0,253458	7 0,061325
$BBMC1$	9 0,023311	64 0,130063	126 0,233935	27 27,850624	114 5 013	18 729	155 5 426	23 916	23 4,501392	7 0,825221
$BBMC2$	9 0,020484	64 0,057249	126 0,290966	27 27,389044	114 5 217	18 607	155 5 158	23 758	23 4,236455	7 0,991203
$BBMC3$	9 0,024554	64 0,102358	126 0,234307	27 28,250706	114 6 097	18 760	155 5 226	23 857	23 4,480955	7 1,051399
FMC	9 0,001997	64 5,587436	126 344 461	26 0,000190	114 0,023029	18 2,111731	155 0,964778	23 6,483014	23 0,729621	7 0,006286
FMC_h	9 0,002089	64 0,184587	126 0,718897	26 0,000410	114 0,099585	18 2,305851	155 2,643887	23 2,443422	23 0,041777	7 0,006702
CMM	9 0,020702	64 0,000016	126 0,000014	27 0,001639	114 126,358925	18 76,335052	155 253,036819	23 1 177	23 326,833527	7 0,655353
CMM_{h1}	9 0,027016	64 0,000020	126 0,000013	27 0,002373	114 4 681	18 80,742821	155 32 890	23 575,333923	23 17,450352	7 0,866367
CMM_{h2}	9 0,022095	63 0,000007	124 0,000007	27 0,001363	110 1 802	15 30,707447	154 27 459	23 538,857849	23 14,742454	7 0,671326

Tableau 1.4 – Résultats temporels pour la recherche de clique maximum sur des graphes compressés. Le temps est donné en secondes.

Graphe	**|K|**	*MCE*	*Tomita*	*Eppstein*	$Eppstein_h$	*TomitaComp*
celegans	668	0,020197	0,007344	0,018796	0,014667	0,006976
c-fat500-5	16	0,162572	0,055711	0,050987	0,043930	0,000097
c-fat500-10	8	0,462186	0,177434	0,134342	0,131667	0,000082
coPapersDBLP	9 910	3,447563	13,937470	17,660326	17,750969	0,008974
com-dblp	257 551	903	2 998	3 389	3 590	2 173
Hook_1498	94 861	43,706188	83,618813	107,354568	107,472374	16,051380
notreDame	495 947	909	1 657	2 616	2 582	735
PR02R	216 741	85,289467	42,117161	51,659836	51,719543	62,114120
rattus	41 118	4,044658	2,268737	2,593526	2,587570	4,077898
worm	5 641	0,100725	0,178407	0,298402	0,294884	0,098185

Tableau 1.5 – Résultats temporels pour l'énumération de cliques maximales sur des graphes compressés. Le temps est donné en secondes.

Nom	**|V|**	**|E|**	$deg_{moy}(G)$	$\Delta(G)$	$\delta(G)$
com-amazon	334 863	925 872	5,53	549	1
edit-frwiktionary	1 905 460	4 794 624	5,03	1 329 427	1
email-enron	36 692	183 831	10,02	1383	1
flickr	757 212	1 371 821	3,62	1912	1
standford	281 903	1 992 636	14,14	38 625	1

Tableau 1.6 – Graphe de données pour les tests d'énumération de cliques dans des graphes compressés.

Nom du graphe	**temps (s)**	$\|A.C\|$	$te_c(\%)$	$\|V_{Pc}\|$	$tv_{Pc}(\%)$	$\|E_{Pc}\|$	$te_{Pc}(\%)$
com-amazon	1 441	815 658	11,90	300 958	10,13	814 884	11,99
edit-frwiktionary	22 238	778 111	83,77	95 702	94,98	778 111	83,77
email-enron	6,747052	155 526	15,40	22 490	38,71	155 448	15,44
flickr	5 845	1 000 346	27,08	456 097	39,77	999 940	27,11
standford	1 023	991 724	50,23	171 361	39,21	955 534	52,05

Tableau 1.7 – Statistiques de la compression de graphe par décomposition modulaire.

Graphe	**|K|**	*Tomita*	*TomitaComp*
com-amazon	470 042	4 346	3 832
edit-frwiktionary	2 753 242	74 819	2 823
email-enron	226 859	51,54	77,10
flickr	1 165 934	13 135	10 037
standford	1 055 936	3 267	2 774

Tableau 1.8 – Résultats temporels pour l'énumération de cliques maximales sur des graphes compressés. Le temps est donné en secondes.

Nous avons enfin calculé le gain de temps observé entre l'algorithme sur le graphe compressé et celui duquel il est tiré (s'appliquant sur le graphe non compressé). Nous avons comparé le rapport observé entre les temps de calcul en fonction du taux de compression du graphe premier aîné. La figure 1.25 présente les résultats en fonction du taux de compression des noeuds et la figure 1.26 présente les résultats en fonction du taux de compression des arêtes. Nous remarquons que l'algorithme sur le graphe compressé semble donner de meilleurs résultats lorsque le taux de compression est supérieur à 40% pour les noeuds et 20% pour les arêtes. Si l'on prend en compte le temps nécessaire à la compression, les taux sont de 70% pour les noeuds et 80% pour les arêtes.

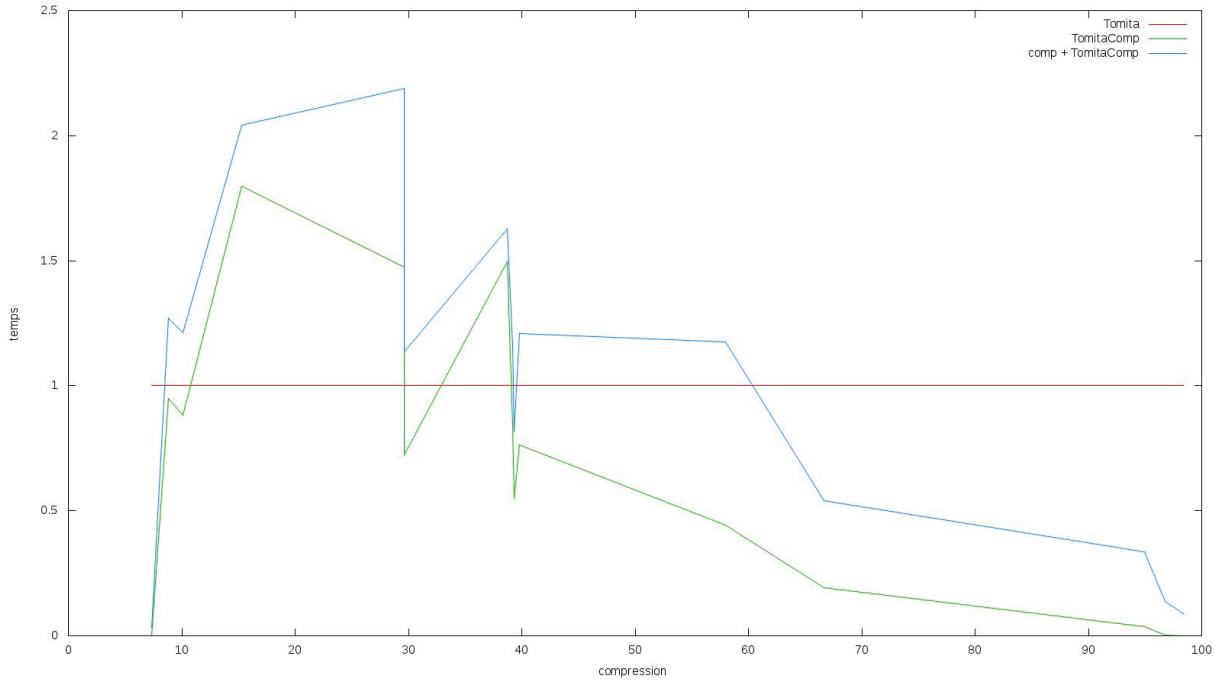


FIGURE 1.25 – Comparaison du temps mis par l'algorithme compressé par rapport au temps initial selon le taux de compression calculé à partir des noeuds.

1.5 Discussion

Notre méthode de calcul de cliques dans un graphe compressé donne de mauvais résultats pour la recherche de la clique maximum et de bons résultats pour le problème d'énumération de cliques maximales.

L'adaptation d'un algorithme de recherche de clique maximum ne donne pas de bons résultats certainement car les méthodes d'élagages usuelles ne peuvent pas être utilisées. En effet, les noeuds de faibles degrés dans le graphe premier issus de la décomposition modulaire peuvent correspondre à des cliques importantes (voir figure 1.22).

L'adaptation d'un algorithme d'énumération de cliques maximales semble donner de bons résultats. Dès que le graphe possède un taux de compression non négligeable, l'algorithme donne généralement de meilleurs résultats (voir tableau 1.8). Cependant, la prise en compte du temps de compression est nécessaire pour relativiser le temps gagné. On peut néanmoins considérer que le gain de temps n'est pas le seul bénéfice espéré. En effet, notre méthode permet d'énumérer les cliques sans avoir besoin de décompresser le graphe, ce qui permet également de travailler sur un graphe plus petit en mémoire.

Dernière remarque, notre méthode n'est utile que sur des graphes pouvant être compressés par décomposition modulaire, ce qui n'est pas le cas de tous les graphes.

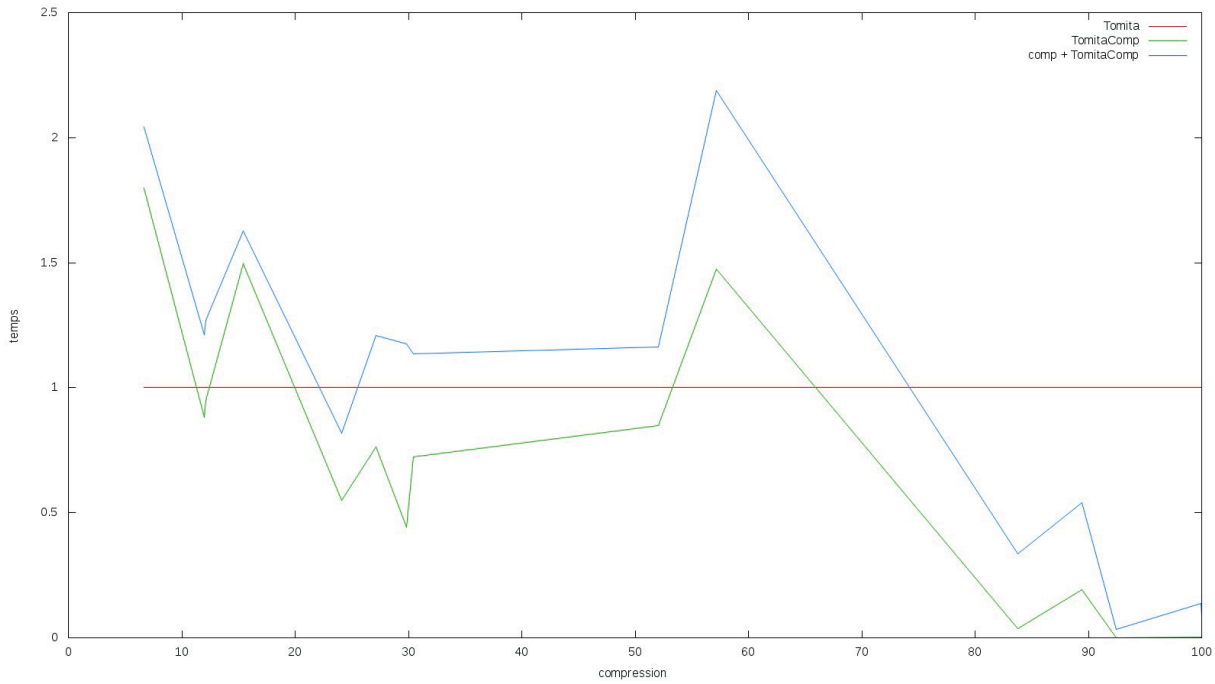


FIGURE 1.26 – Comparaison du temps mis par l'algorithme compressé par rapport au temps initial selon le taux de compression calculé à partir des arêtes.

1.6 Conclusion et ouvertures

Les problèmes d'énumération de recherche de cliques maximums et de cliques maximales 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 chapitre, nous avons proposé et évalué une nouvelle méthode pour résoudre ces problèmes.

Notre méthode 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 sans pertes qui 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 méthode, pour le problème de la clique maximum, n'est pas suffisamment efficace. Cependant, l'énumération de cliques maximales montre de meilleures performances dès que la compression du graphe offre une diminution du nombre d'arêtes de l'ordre de 20%.

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

— Étudier l'impact de la méthode d'exploration de l'arbre de décomposition modulaire, l'ordre d'exploration devant avoir un impact, notamment pour la recherche de clique maximale
— Augmenter l'efficacité de la recherche de clique maximum 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
— Évaluer l'utilisation de la mémoire
— 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

Chapitre 2

Énumération de quasi-cliques

Résumé

Dans ce chapitre nous étudierons les problèmes d'énumération de quasi-cliques. Nous présenterons les modèles de quasi-cliques et les méthodes de recherche de quasi-cliques présents dans la littérature en section 2.2. Dans la section 2.3 nous détaillerons la solution que nous avons développé pour l'énumération de quasi-cliques. Dans la section 2.4 nous proposerons une évaluation de cette solution via plusieurs expérimentations.

2.1 Introduction

Un ensemble de noeuds particulièrement reliés entre eux mais peu reliés avec les autres noeuds du graphe est communément appelé communauté, bien qu'il n'existe pas de formalisme unique pour définir ces communautés. L'intérêt de rechercher ces communautés est de faire apparaître les structures sous-jacentes d'un graphe. Dans le cas de réseaux sociaux les communautés peuvent se rapporter à des groupes d'individus tandis que dans le cas de réseaux routiers les communautés identifierons généralement des zones denses telles que les villes.

La figure 2.1 propose par exemple le partage d'un graphe en 3 communautés distinctes, chacune représentée par une coloration différente des noeuds. Les arêtes à l'intérieur des communautés sont en traits continus tandis que celles liants les communautés sont en traits hachés.

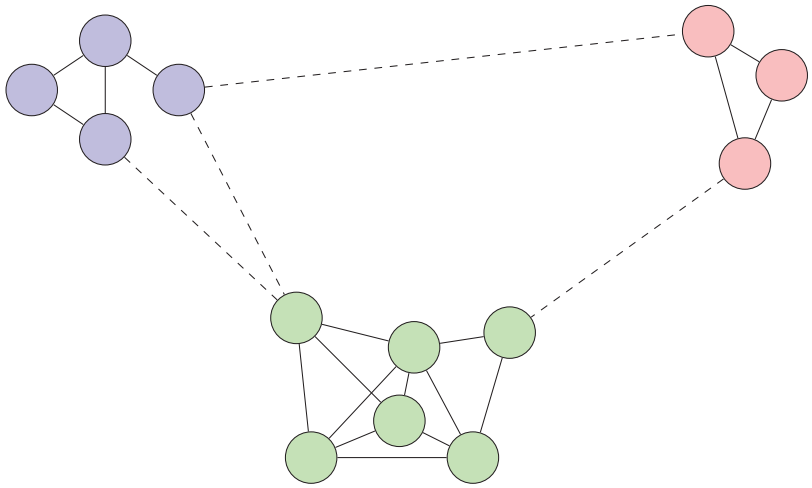


Figure 2.1 – Exemple de communautés dans un graphe.

La recherche des communautés a diverse applications, notamment sur les réseaux biologiques [94] ou sur le World Wide Web [26].

Chaque noeud peut se voir attribuer une ou plusieurs étiquettes correspondant aux communautés auxquelles il appartient. Ces communautés sont généralement appelées *clusters*. On peut mesurer la qualité d'une communauté avec différents marqueurs telle que la densité ou le coefficient de clustering.

Définition 2.1. La densité d'un graphe $G(V,E)$ est une valeur mathématique correspondant au rapport entre le nombre d'arêtes du graphe $|E|$ et le nombre d'arêtes maximales possibles dans le graphe. Dans le cas d'un graphe non-orienté simple :

$$\frac{|V| \cdot (|V| - 1)}{2}$$

La valeur de la densité oscille entre 0 pour un graphe indépendant et 1 pour un graphe complet.

On distingue deux sortes de coefficients de clustering :
— Le *coefficent de clustering global*.
— Le *coefficent de clustering local moyen*

Définition 2.2. Le coefficient de clustering global d'un graphe $G(V,E)$ est une valeur mathématique correspondant au rapport entre le nombre de triangle [1] et le nombre de triplets connectés [2]. Le coefficent de clustering global noté $C_g(G)$ est défini par :

$$C_g(G) = \frac{3 \cdot \textit{nombre de triangles}}{\textit{nombre de triplets connectés}}$$

La valeur du coefficient de clustering global oscille entre 0 pour un graphe sans triangle et 1 pour un graphe dont chaque composante connexe est un sous-graphe complet.

Définition 2.3. Le coefficient de clustering local moyen d'un graphe $G(V,E)$ est une valeur mathématique correspondant à la moyenne des coefficients locaux de chacun des noeuds du graphe. Le coefficient local d'un noeud $u \in V$, noté $c_l(u)$ est défini par le nombre de ses voisins connectés entre eux :

$$\forall v,\, w \in \Gamma(u),\, c_l(u) = \frac{2 \cdot |(v,w) \in E|}{|\Gamma(u)| \cdot |(\Gamma(u) - 1)|}$$

Ainsi le coeffcient de clustering local moyen du graphe, noté $C_l(G)$ est défini par :

$$\forall u \in V,\, C_l(G) = \frac{\sum c_l(u)}{|V|}$$

La valeur du coefficient de clustering local moyen oscille entre 0 pour un graphe sans triangle et 1 pour un graphe dont chaque composante connexe est un sous-graphe complet.

La figure 2.2 présente différents graphes et leurs valeurs de densité ainsi que la valeur de leurs coefficients de clustering global et local moyen.

Les méthodes de communauté permettent donc de définir des sous-graphes fortement connectés tandis que les mesures de densités et de coefficient de clustering permettent d'évaluer la qualité de ces communautés.

La construction de graphe à partir de données réelles se fait par copie de réseaux existants ou par agrégation de données. Par exemple, dans le cadre d'un échange de messages entre utilisateurs, on peut réaliser un graphe liant les personnes échangeant des messages.

Dans ces cas des informations peuvent être manquantes. En effet, lors de la construction des graphes, des informations ont pu être :
— ratées, par exemple si le système de collecte de message était inactif
— ignorées, par exemple si les messages échangés n'ont pas été suffisamment nombreux pour être jugés pertinents

1. Un triangle est un sous-graphe complet composé de 3 noeuds.
2. Un triplet connecté correspond à un sous-graphe connexe composé de 3 noeuds.

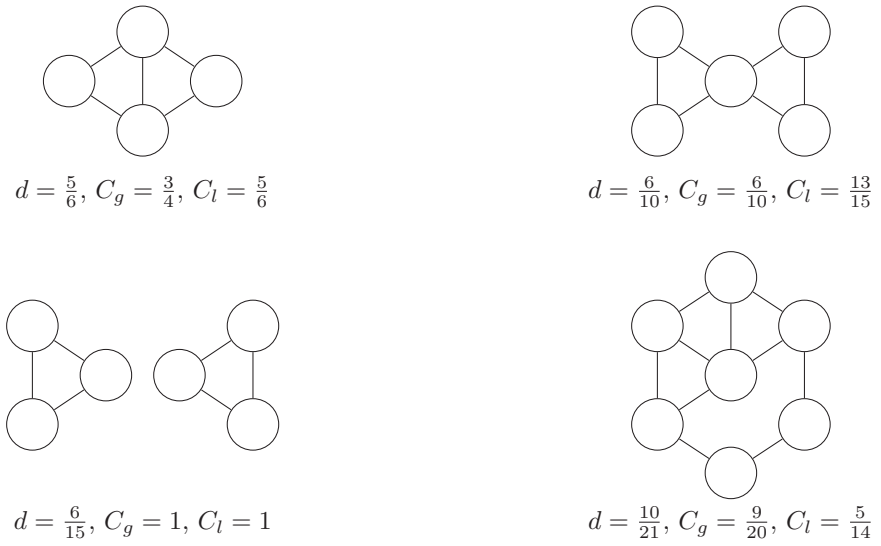


FIGURE 2.2 – Présentation de densités et de coefficients de clustering locaux et globaux sur divers graphes.

Un problème apparaît alors, on peut se retrouver avec des graphes de données qui ne reflètent pas exactement la réalité qu'ils sont supposés représenter. La question est alors de trouver une ou plusieurs techniques permettant de corriger les biais formés lors de la création du graphe.

Nous proposons ici une technique permettant d'évaluer les relations entre l'ensemble des noeuds du graphe via l'évaluation de communautés. Les communautés permettent en effet d'identifier les parties du graphe fortement connectés et donc d'identifier des arêtes absentes qui étaient en réalité présentes dans la situation réelle initiale. Cette idée reposant sur le paradigme *«L'ami de mon ami est mon ami»* : on peut en effet imaginer que les liens absents dans des communautés détectés dans le graphe créé sont potentiellement des liens présents physiquement initialement.

Afin d'évaluer la qualité de chacune des arêtes possibles du graphe nous nous intéressons à un modèle particulier de communautés, les *quasi-cliques*. Les quasi-cliques sont des sous-graphes quasiment complet, c'est à dire auxquels il manque un faible pourcentage d'arêtes.

L'idée repose sur une énumération de l'ensemble des quasi-cliques d'un graphe. Ensuite, pour chaque paire de noeuds du graphe, on regarde le nombre de quasi-cliques communes, c'est à dire dans lesquelles les noeuds sont tous les deux présents.

Nous proposons dans ce chapitre une méthode d'évaluation de potentiels liens dans un graphe via l'évaluation de la co-occurrence de paire de noeuds. Pour cela nous avons développé une méthode d'énumération de quasi-clique dans un graphe.

2.2 État de l'art

2.2.1 Communautés

La recherche de communautés [39], aussi appelée recherche de cluster, est un problème visant à regrouper des noeuds qui partagent dans le graphe des propriétés locales en s'imaginant que les entités représentées par ces noeuds partagent physiquement des données ou jouent des rôles similaires dans le monde représenté par le graphe.

La recherche de communauté est un problème bien étudié dans la littérature notamment pour des graphes simples [35] et pour des graphes dirigés [78].

Une des solutions de partitionnement de graphe a été d'étudier les *coupes* de graphe [64, 34].

Définition 2.4. Une coupe de graphe est une partition des noeuds d'un graphe en deux sous-ensembles disjoints S et T, $S \cap T = \emptyset$. Le cardinal de la coupe représente le nombre d'arêtes liant les deux ensembles :

$$\forall u \in S,\ v \in T,\ |(u,v) \in E|$$

Une coupe est *minimale* si son cardinal est minimal.

Par succession de coupes minimales successives on peut obtenir un graphe subdivisé en sous-ensembles dont les arêtes sont plus nombreuses à l'intérieur des sous-ensembles que celles dirigées vers l'extérieur. Le problème de la coupe minimale est que l'on peut se retrouver avec des sous-ensembles de tailles différentes. Pour éviter ce problème, on peut utiliser des contraintes supplémentaires telle que la balance, le ratio de coupe ou la coupe normalisée [53, 99].

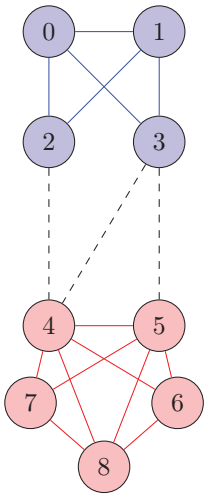
La modularité est une des mesures d'évaluation de la qualité d'un partitionnement introduite initialement par Newman et Girvan pour des graphes simples puis utilisée ensuite pour des graphes pondérés [85, 84]. L'idée est d'évaluer le partitionnement du graphe en fonction du nombre d'arêtes à l'intérieur des sous-ensembles et du nombre d'arêtes liant les sous-ensembles, appelées arêtes extérieures. L'idée étant qu'une bonne partition de graphe maximise le nombre d'arêtes intérieures et minimise le nombre d'arêtes extérieures.

Définition 2.5. La modularité d'une partition de noeuds du graphes $P = \{P_0,P_1,...,P_n\}$ où chacune des partitions est disjointe une à une aux autres, est défini par :

$$\sum_{i=1}^{n} \frac{|e_{ii}|}{|E|} - \left(\frac{|e_{ii}| + |e_{ij}|}{|E|}\right)^2$$

où e_{ii} sont les arêtes intérieurs de la partition P_i et e_{ij} sont les arêtes extérieures. La valeur de la modularité oscille entre -1 et 1. La modularité tendra vers zéro si l'on partitionne aléatoirement un graphe dans lequel les arêtes ont été réparties aléatoirement. La modularité sera négative si la partition favorise les arêtes extérieures et positive si elle favorise les arêtes intérieures.

La figure 2.3 présente un exemple du calcul de modularité pour le partitionnement d'un graphe en deux sous-ensembles P_1 et P_2. La mesure de la modularité se fait par l'addition du calcul de modularité de chaque sous-ensemble. Ainsi le premier sous-ensemble en bleu P_1 possède 5 arêtes intérieures (les arêtes bleues) et 3 arêtes extérieures (les arêtes pointillées) tandis que le sous-ensemble en rouge P_2 possède 9 arêtes intérieures (les arêtes rouges) et 3 arêtes extérieures (les arêtes pointillées).



$$\begin{aligned}
P &= \{P_1,P_2\} \\
P_1 &= \{0,1,2,3\} \\
P_2 &= \{4,5,6,7,8\} \\
M(P) &= \frac{|e_{11}|}{|E|} - \left(\frac{|e_{11}| + |e_{12}|}{|E|}\right)^2 + \\
&\quad \frac{|e_{22}|}{|E|} - \left(\frac{|e_{22}| + |e_{21}|}{|E|}\right)^2 \\
&= \frac{5}{17} - \left(\frac{5+3}{17}\right)^2 + \frac{9}{17} - \left(\frac{9+3}{17}\right)^2 \\
&= 0.1038
\end{aligned}$$

FIGURE 2.3 – Exemple du calcul de modularité.

Trouver la partition optimale avec la modularité maximale est un problème NP-Complet [17]. Le principal problème de l'optimisation de la modularité globale est que cela créé des partitions de taille équivalente, ce qui est rarement le cas dans la réalité, on parle alors de *limite de résolution*.

Afin de palier ce problème des méthodes de résolution de niveaux multiples ont été développées comme, par exemple, l'algorithme de *Louvain* proposé par Blondel et al. [14]. Cet algorithme fonctionne en étapes successives où l'on alterne 2 phases :

1. dans la première phase on attribue à chaque noeud une communauté différente et on regroupe les communautés qui augmentent le score de modularité. Une fois qu'il n'y a plus de gain possible on passe à la deuxième phase.
2. la seconde phase est une phase de compression de graphe (voir 1.2.1) où chaque communauté est regroupée en un noeud. Les arêtes sont regroupées en arêtes pondérées liant les noeuds-communautés (pour les arêtes extérieures) et en boucles pondérées sur les noeuds-communautés (pour les arêtes intérieures).

La figure 2.4 présente le fonctionnement des deux phases de l'algorithme de Louvain[14].

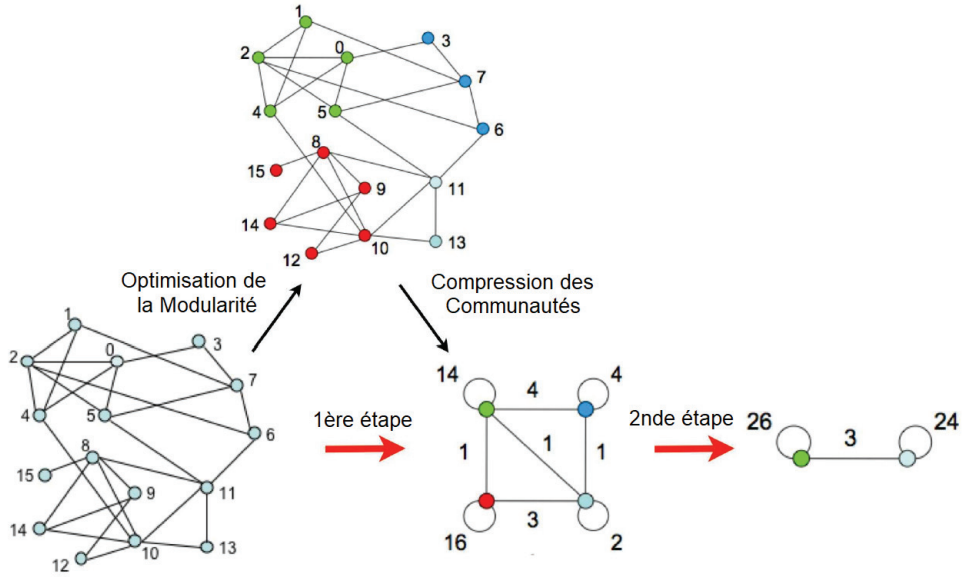


FIGURE 2.4 – Exemple d'optimisation de la modularité selon la méthode de Louvain [14], les poids sur les arêtes représentent le nombre de noeud qui étaient des extrémités d'arêtes avant la compression.

Différentes améliorations de l'algorithme de Louvain ont été proposées telle qu'une diminution du temps de calcul par méthodes d'élagages et d'ordonnancement de l'exploration [100] ou par une méthode générique permettant de changer la fonction de calcul de modularité [22].

Dans des cas appliqués avec des données réelles, les noeuds peuvent cependant appartenir à plusieurs clusters. Par exemple, dans le cas d'un réseau social, un noeud représentant une personne peut être relié à plusieurs clusters pouvant représenter sa famille, son club de sport et son entreprise. Pour représenter ces cas de figure plusieurs méthodes ont été proposées [3, 32] reposant notamment sur les *graphes adjoints* [54].

Définition 2.6. Un graphe adjoint $L(G)$ est un graphe issu de la transformation des arêtes du graphe initial en noeuds. Les noeuds de $L(G)$ sont reliés par une arête si et seulement si les arêtes initiales de G possédaient une extrémité commune.

La détection de communautés dans le graphe adjoint permet de réaliser une coloration d'arête du graphe initial. Ainsi les noeuds du graphe initial sont regroupés par communautés et peuvent appartenir à plusieurs communautés s'ils possèdent des arêtes de différentes couleurs.

La figure 2.5 présente la transformation d'un graphe en graphe adjoint et les multiples communautés qui en découle. Le graphe initial G est transformé en graphe adjoint $L(G)$ où les noeuds possèdent l'identifiant

des deux extrémité de l'arête dans G, ainsi l'arête (1,2) dans G devient le noeud 12 dans $L(G)$. On détermine ensuite deux groupes dans $L(G)$, un bleu et un rouge. Ces groupes dans $L(G)$ donnent une coloration aux arêtes de G. Ainsi, on peut déterminer deux groupes dans le graphe initial G : le bleu, composé des noeuds $\{1,2,3,4,5,7\}$, et le rouge, composé des noeuds $\{5,6,7,8,9\}$. Dans ce cas de figure, les noeuds 5 et 7 appartiennent donc aux deux communautés.

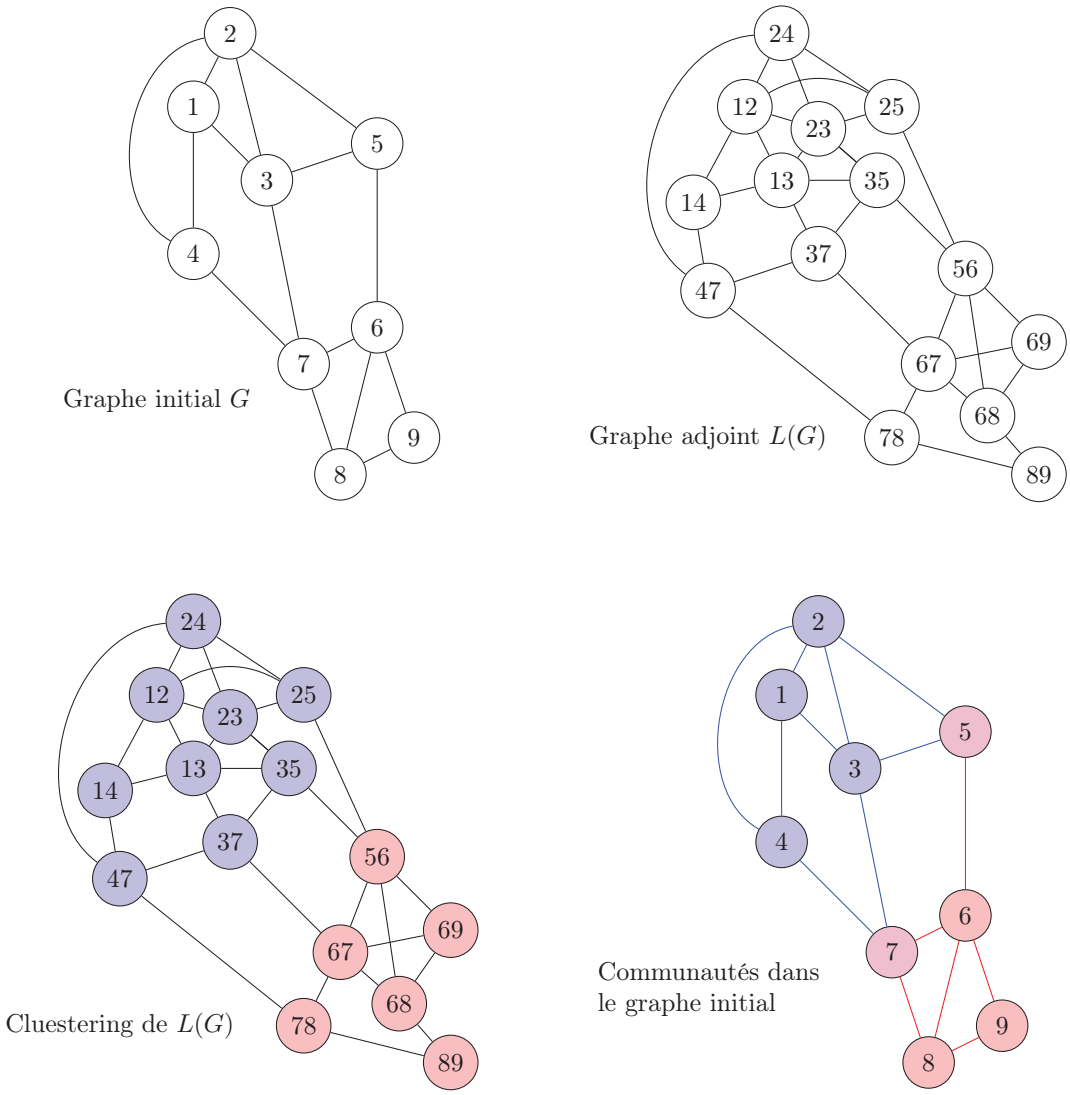


FIGURE 2.5 – Détection de communautés à l'aide de graphe adjoint.

Une fois ces communautés détectées on peut imaginer qu'il existe un lien entre chacun des membres de la communauté, même si le lien n'est peut être pas assez fort pour apparaître dans le graphe. La problème posé est alors comment évaluer la potentialité de l'existence d'un lien entre deux noeuds, même s'il n'est pas présent dans le graphe ?

La détection de communautés peut permettre de détecter des liens potentiellement manquants mais ne permet pas de quantifier la probabilité que ce lien existe réellement.

2.2.2 Quasi-Cliques

Pour répondre à cette problématique nous proposons d'étudier les quasi-cliques d'un graphe. Une quasi-clique est un sous-ensemble de noeuds d'un graphe quasiment complet. Plusieurs formalismes ont été utilisés pour étudier les quasi-cliques. Chaque approche est basée sur un paramètre caractéristique d'une quasi-clique. La définition d'une quasi-clique diffère donc selon la caractéristique étudiée.

Ainsi Matsuda et al. [79] proposent d'étudier les quasi-cliques avec le paramètre λ défini par le plus petit degré des noeuds dans le sous-graphe induit par l'ensemble représentant la quasi-clique. On appellera ce type de quasi-clique les λ-*cliques*.

Définition 2.7. Une λ-clique est un sous-graphe C de G dont le degré minimal dans le graphe induit par C est :

$$\lceil \lambda \cdot (|C| - 1) \rceil$$

Le paramètre λ est défini entre 0 et 1. Lorsque $\lambda = 0$ la quasi-clique n'est pas connexe car elle possède au moins un noeud isolé tandis qu'à $\lambda = 1$ la quasi-clique est une clique.

Abello et al. [1] proposent eux d'étudier les quasi-cliques avec le paramètre γ défini par le ratio entre le nombre d'arêtes présentes dans le sous-graphe induit par l'ensemble représentant la quasi-clique et le nombre théorique maximal d'arêtes si la quasi-clique était une clique. On nommera ce type de quasi-clique les γ-*cliques*.

Définition 2.8. Une γ-clique est un sous-graphe C de G dont le nombre d'arêtes minimales dans le graphe induit par C est :

$$\left\lceil \gamma \cdot \left(\frac{|C| \cdot (|C| - 1)}{2} \right) \right\rceil$$

Le paramètre γ est défini entre 0 et 1. Si $\gamma = 0$ la quasi-clique est un stable alors qu'à $\gamma = 1$ la quasi-clique est une clique.

Enfin Brunato et al. [19] proposent d'utiliser les deux paramètres, λ et γ où une quasi-clique doit avoir un certain nombre d'arêtes et chaque noeud doit avoir un degré suffisant dans le sous-graphe induit par l'ensemble des noeuds composant la quasi-clique. Ces quasi-cliques sont appelées λ-γ-*cliques*.

Définition 2.9. Une λ-γ-clique est un sous-graphe C de G dont le nombre d'arêtes minimales dans le graphe induit par C est :

$$\left\lceil \gamma \cdot \left(\frac{|C| \cdot (|C| - 1)}{2} \right) \right\rceil$$

Ainsi tout noeud dans le graphe induit par C est, au minimum, de degré : :

$$\lceil \lambda \cdot (|C| - 1) \rceil$$

On peut démontrer que la valeur du paramètre λ minore celle du paramètre γ.

Proposition 2.10. Pour toute λ-γ-clique, la valeur du paramètre λ est inférieure ou égale à celle du paramètre γ.

Preuve. Soit une λ-γ-clique C :

$$\forall u \in C, |\Gamma(u) \cap C| \geq \lceil \lambda \cdot (|C| - 1) \rceil$$

Ainsi, dans le sous-graphe induit par C, le nombre d'arêtes minimum est de :

$$|C| \cdot \frac{\lceil \lambda \cdot (|C| - 1) \rceil}{2}$$

Soit

$$\left\lceil \lambda \cdot \frac{|C| \cdot (|C| - 1)}{2} \right\rceil$$

Ce qui implique :

$$\gamma \geq \lambda$$

□

La figure 2.6 présente différentes quasi-cliques et les valeurs respectives des paramètres λ et γ. Le noeud en rouge est le noeud de plus petit degré, il permet de délimiter la valeur du paramètre λ tandis que les arêtes hachées permettent d'évaluer la valeur du paramètre γ en visualisant les arêtes manquantes.

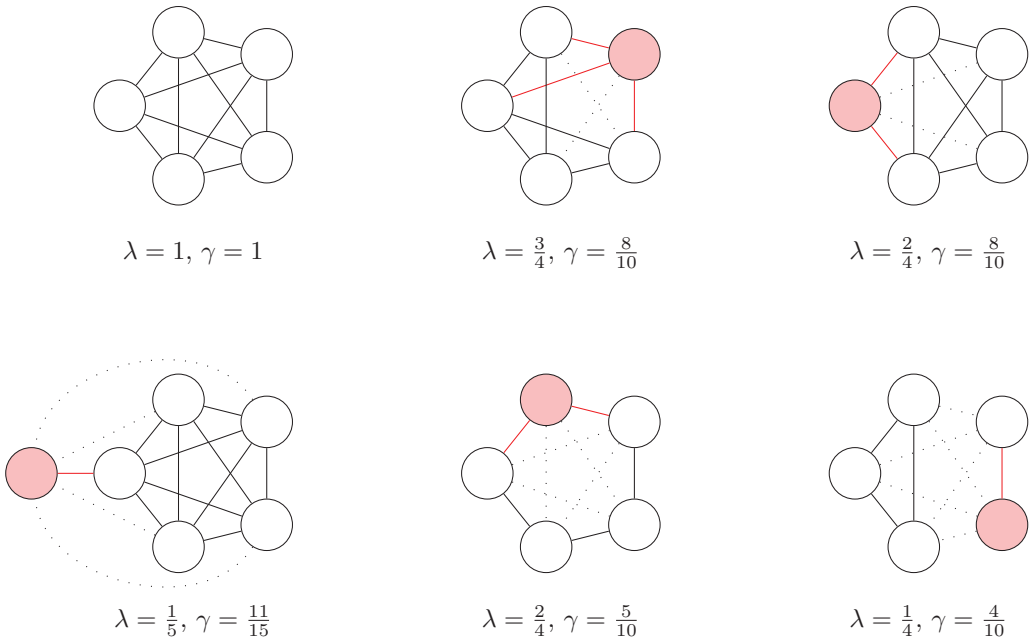


FIGURE 2.6 – Présentation de différentes quasi-cliques et des valeurs des paramètres λ et γ.

2.2.3 Énumération de quasi-cliques maximales

En définissant des valeurs minimales pour les paramètres λ et γ, nommées respectivement λ_{min} et/ou γ_{min}, on peut définir des *quasi-cliques maximales*.

Définition 2.11. Une λ-clique C est maximale s'il n'existe pas d'ensemble de noeuds $L \in P - C$ de telle sorte que :

$$\lambda(C + L) \geq \lambda_{min}$$

Une γ-clique C est maximale s'il n'existe pas d'ensemble de noeuds $L \in P - C$ de telle sorte que :

$$\gamma(C + L) \geq \gamma_{min}$$

Une λ-γ-clique C est maximale s'il n'existe pas d'ensemble de noeuds $L \in P - C$ de telle sorte que :

$$\lambda(C + L) \geq \lambda_{min}$$
$$\textbf{ET } \gamma(C + L) \geq \gamma_{min}$$

On peut alors chercher à énumérer tout ou partie des quasi-cliques maximales d'un graphe en accord avec les valeurs des paramètres λ_{min} et/ou γ_{min}. Ce problème est appelé *MQCE* pour l'anglais *Maximal Quasi-Clique Enumeration*.

Plusieurs algorithmes ont été proposés pour résoudre ce problème. Ainsi Uno [114] propose de lister des γ-cliques en temps polynomial à partir d'un arbre d'exploration du voisinage d'une clique courante.

Yang et al. [119] proposent d'énumérer des γ-cliques par aggrégation de noeuds en utilisant le paradigme *Map-Reduce*

Définition 2.12. Map-Reduce[27] est un paradigme de programmation permettant de découper un problème en plusieurs sous-problèmes plus faciles à résoudre. Cela permet de distribuer le calcul sur plusieurs unités de traitement.

L'algorithme de Yang et al. permet de lister des quasi-cliques maximales mais n'a aucune garantie de trouver l'ensemble des quasi-cliques maximales[119, Chapitre IV.B].

Brunato et al. [19] proposent d'adapter deux algorithmes *RLS* et *DLS-MC* de recherche locale de clique afin de rechercher des λ-γ-cliques.

Algorithme RLS

L'algorithme *RLS*, pour l'anglais *Reactive Local Search*, proposé par Battiti [9, 8] est un algorithme qui permet de rechercher localement des cliques par ajout et suppression de noeuds à une clique courante.

Brunato et al. adaptent RLS et les fonctions d'ajout et de suppression de noeuds pour respecter les contraintes imposées par les valeur des paramètres λ_{min} et γ_{min}.

Pour ajouter un noeud supplémentaire à la quasi-clique courante on définit un ensemble de *noeuds critiques*. Un noeud critique est un noeud dont il est nécessaire d'ajouter un voisin si on augmente la taille de la clique courante.

Définition 2.13. Un noeud critique dans une quasi-clique implique que tout nouveau noeud ajouté à la quasi-clique doit être voisin du noeud critique (afin de satisfaire le paramètre λ).

$$\exists u \in C,\ |\Gamma(u) \cap C| < \lceil \lambda_{min} \cdot |C| \rceil \Rightarrow u \in Crit(C)$$

La figure 2.7 présente une quasi-clique courante (bleue) dans un graphe. Pour $\lambda_{min} = \gamma_{min} = \frac{2}{3}$ les noeuds critiques sont les noeuds 1 et 7 dont la valeur du paramètre λ au sein de la quasi-clique est égale à $\frac{2}{3}$ et auxquels il est donc nécessaire d'ajouter un voisin pour que le degré reste suffisant.

Afin de satisfaire la contrainte de densité γ_{min} les noeuds candidats pour l'agrandissement de la quasi-clique doivent ajouter un nombre e_{min} suffisant d'arêtes.

$$e_{min} = \left\lceil \gamma_{min} \cdot \frac{|C| \cdot (|C| + 1)}{2} \right\rceil - |E_C|$$

Dans l'exemple de la figure 2.7, $e_{min} = 2$.

Il est également nécessaire que les noeuds ajoutés aient un degré minimum avec la quasi-clique courante pour satisfaire la contrainte λ_{min}.

L'ensemble des noeuds éligibles *Add* pouvant être ajoutés à la quasi-clique courante doivent donc satisfaire les contraintes imposées :

$$\begin{aligned} &\forall u \in V - C,\ v \in Crit, \\ &|\Gamma(u) \cap C| \geq max(\lceil \lambda_{min} \cdot |C| \rceil, e_{min}) \\ &\qquad \textbf{ET } \exists (u,v) \in E \Rightarrow u \in Add \end{aligned}$$

Ainsi dans l'exemple de la figure 2.7, avec $\lambda_{min} = \gamma_{min} = \frac{2}{3}$, le seul noeud satisfaisant les contraintes d'ajout est le noeud 2.

Pour supprimer un noeud de la quasi-clique courante il convient d'être sûr que la suppression des arêtes qui le lient aux autres noeuds de la quasi-clique courante permette de conserver la satisfaction des contraintes imposées par les paramètres λ_{min} et γ_{min}. Dans cette optique, un ensemble de noeuds $RCrit$ est défini, où un noeud appartient à l'ensemble $RCrit$ si la suppression d'un de ses voisins dans la clique courante ne permet pas de satisfaire la contrainte imposée par le paramètre λ_{min}.

$$\forall u \in V, \Gamma(u) - 1 < \lceil \lambda_{min} \cdot (|C| - 2) \rceil \Rightarrow u \in RCrit$$

La suppression d'un noeud doit également éviter de supprimer trop d'arêtes dans la quasi-clique pour permettre de respecter les contraintes imposées par le paramètre γ_{min}. Ainsi le nombre maximal d'arêtes pouvant être supprimées est défini par la valeur e_{max}.

$$e_{max} = |E_C| - \left\lceil \gamma_{min} \cdot \frac{(|C| - 1) \cdot (|C| - 2)}{2} \right\rceil$$

Dans l'exemple de la figure 2.7, $e_{max} = 3$.

L'ensemble des noeuds éligibles *Rem* pouvant être ajoutés à la quasi-clique courante doivent donc satisfaire les contraintes imposées :

$$\begin{gathered} \forall u \in C, \\ |\Gamma(u) \cap C| \leq e_{max} \\ \textbf{ET } \Gamma(u) \cap RCrit = \emptyset \Rightarrow u \in Rem \end{gathered}$$

Ainsi dans l'exemple de la figure 2.7, avec $\lambda_{min} = \gamma_{min} = \frac{2}{3}$, les noeuds satisfaisants les contraintes de retrait sont les noeuds 1 et 7.

Afin d'éviter que RLS tombe toujours sur les mêmes quasi-cliques, une recherche tabou [40, 41] est utilisée. Le principe de la recherche tabou repose sur l'utilisation d'une mémoire enregistrant les états de la quasi-clique courante dans les étapes précédentes et interdit à la quasi-clique courante de revenir temporairement vers ces états en maintenant une liste de noeuds interdits.

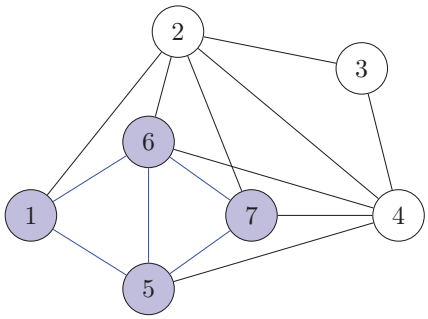


FIGURE 2.7 – Présentation d'une λ-γ-clique courante [19] pour les étapes d'énumération de quasi-cliques maximales. La quasi-clique courante est en bleue.

Algorithme DLS-MC

L'algorithme *DLS-MC*, pour l'anglais *Dynamic Local Search for Maximum Clique*, proposé par Pullan et Hoos [92] est un algorithme qui permet de rechercher localement des cliques par ajout de noeuds et phases de plateau, où l'on conserve la taille d'une clique courante en ajoutant un noeud u puis en supprimant un noeud v.

Afin de conserver les valeurs des paramètres λ et γ de la quasi-clique courante au dessus des valeurs minimales, des ensembles de noeuds sont définis. Ainsi les noeuds $PAdd$ pouvant être ajoutés dans une phase de plateau sont les noeuds dont le voisinage à la quasi-clique est suffisant :

$$\begin{gathered} \forall u \in V - C, \\ |\Gamma(u) \cap C| \geq \lceil \lambda_{min} \cdot (|C| - 1) \rceil \Rightarrow u \in PAdd \end{gathered}$$

L'ensemble des noeuds pouvant être retirés dépend alors du noeud choisi u. On détecte alors l'ensemble des noeuds critiques $PCrit(u)$ pour le retrait :

$$\begin{aligned} &\forall v \in C + u, \\ |\Gamma(v) \cap C| - 1 < &\lceil \lambda_{min} \cdot (|C| - 1) \rceil \\ &\textbf{ET } (v,u) \notin E \Rightarrow v \in PCrit(u) \end{aligned}$$

L'ensemble $PCrit(u)$ contient les noeuds qui ne satisferont pas la contrainte de degré dans la quasi-clique courante C imposée par λ_{min} si u est ajouté à la quasi-clique courante.

Le noeud pouvant être retiré doit également respecter les contraintes imposées par le facteur de densité γ_{min}. Ainsi on défini $pe_{max}(u)$, le nombre maximal d'arêtes pouvant être retiré si le noeud u est ajouté à la clique :

$$pe_{max}(u) = |E_C| + |\Gamma(u) \cap C| - \left\lceil \gamma_{min} \cdot \frac{|C| \cdot (|C| - 1)}{2} \right\rceil$$

Ainsi les noeuds pouvant être retirés si u est ajouté sont les noeuds v appartenant à l'ensemble $PRem(u)$ permettant à la quasi-clique $C + u - v$ de satisfaire les contraintes imposées par les paramètres λ_{min} et γ_{min} :

$$\begin{aligned} &\forall v \in C,\ w \in PCrit(u), \\ |\Gamma(v) \cap (C + u)| \leq &\ pe_{max}(u) \\ &\textbf{ET } (v,w) \notin E \Rightarrow v \in PRem(u) \end{aligned}$$

Ainsi tout noeud u considéré par une phase de plateau comme pouvant être ajouté doit avoir un ensemble $PRem(u)$ non vide.

L'exemple donné dans la figure 2.8 présente un graphe et une quasi-clique courante pour une phase de plateau. Si l'on cherche des λ-γ-cliques avec $\lambda_{min} = \gamma_{min} = \frac{2}{3}$ on remarque que la quasi-clique courante en bleue $C = \{0,1,2,3\}$ est maximale, on ne peut pas ajouter de noeuds tout en continuant à satisfaire les contraintes imposées par les paramètres λ_{min} et γ_{min}.

On peut alors envisager une phase de plateau où l'on ajoute puis retire un noeud à la quasi-clique courante. Ainsi l'ensemble $PAdd$ est composé des noeuds $\{4,5,6\}$, les noeuds 7 et 8 ne satisfaisant pas les contraintes du paramètre λ_{min}. Le noeud 4 impose cependant un ensemble de retrait vide, $PRem(4) = \emptyset$ car les noeuds 2 et 3 sont critiques, $2,3 \in PCrit(4)$ et l'ensemble des noeuds de la quasi-clique courante C a au moins l'un de ces deux noeuds en voisin. Cependant les listes de retrait des noeuds 5 et 6 ne sont pas vides, $PRem(5) = \{1,3\}$, $PRem(6) = \{0,2\}$. Ainsi une phase plateau à partir de la quasi-clique courante C peut amener la création des quasi-cliques $\{0,1,2,5\}$, $\{0,2,3,5\}$, $\{0,1,3,6\}$ et $\{1,2,3,6\}$.

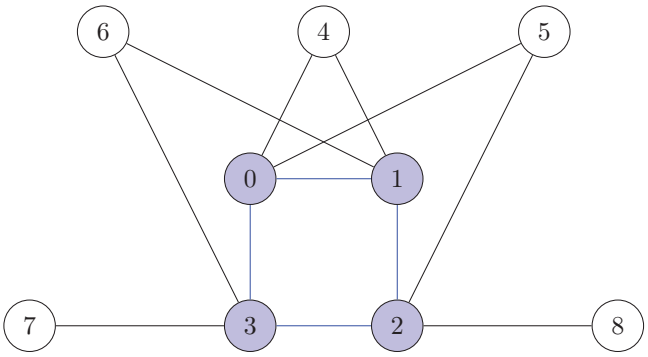


FIGURE 2.8 – Présentation d'une phase de plateau pour une λ-γ-clique courante [19]. La quasi-clique courante est en bleue.

Afin d'éviter que DLS-MC recherche toujours les mêmes quasi-cliques, les noeuds se voient attribuer une pénalité. Ainsi à la fin des phases d'ajout et de plateau, la pénalité des noeuds appartenant à la clique courante C est augmentée de 1. Alternativement, après un certain délais p_d, les pénalités sont réduites de 1. Les noeuds pouvant être sélectionnés lors de phases d'ajout ou de plateau sont donc sélectionnés en fonction de cette pénalité, ils sont ajoutés si sa valeur est faible, retirés si elle est élevée. Enfin, en fonction de la valeur de p_d, on repart d'une situation différente :

— Si $p_d = 1$, cela veut dire que l'on n'a pas de mémoire pour la pénalité, la valeur étant augmentée puis directement diminuée, on sélectionne aléatoirement un noeud $u \in V$, et on repart d'une clique courante $\Gamma(u) \cap C$.

— Si $p_d > 1$, on repart du dernier noeud ajouté lors de la phase de plateau (ou d'ajout s'il n'y avait pas de phase de plateau possible), l'idée d'éviter les noeuds étant dans la quasi-clique courante C.

2.3 Algorithme d'énumération de Quasi-Cliques

Dans cette section nous étudierons les problèmes d'énumération de λ-γ-cliques maximales.

2.3.1 Présentation

Nous proposons un algorithme récursif d'exploration de graphe permettant l'énumération de λ-γ-cliques maximales. Notre idée repose sur un agrandissement constant d'une quasi-clique courante par la sélection de candidats. Notre méthode repose sur la gestion d'ensemble candidats, d'une mémoire des noeuds déjà explorés et d'élagage de ceux à visiter via l'utilisation de pivot [18, 111] (voir 1.2.4 pour plus de détails). Nous proposons une méthode distribuée afin d'éviter des calculs aboutissant à des doublons. Enfin nous montrerons que notre méthode permet de lister avec certitude l'ensemble des λ-γ-cliques ne comportant pas certains sous-graphes.

2.3.2 Distribution du calcul

Afin d'éviter les doublons et d'optimiser le temps de calcul Svendsen et al. [105] ont proposé une version distribuée de l'énumération de cliques par Tomita [111] en utilisant le paradigme Map-Reduce. L'idée repose, pour l'énumération de cliques, sur une sélection de candidats possibles pour l'agrandissement de la clique courante en fonction de l'identifiant et du degré des noeuds par rapport au noeud initial de la clique.

Ainsi, pour chaque noeud $u \in V$, ses voisins sont considérés comme l'ensemble *candidats* et sont répartis dans le sous-ensemble $Cand$ ou dans le sous-ensemble $Fini$ selon s'ils sont ou non des candidats déjà considérés pour l'agrandissement :

$$\begin{aligned}
&\forall u \in V, v \in \Gamma(u) \\
|\Gamma(u)| < |\Gamma(v)| \ \textbf{OU} \ (|\Gamma(u)| = |\Gamma(v)| \ \textbf{ET} \ u < v) &\Rightarrow v \in Cand(u) \\
|\Gamma(u)| > |\Gamma(v)| \ \textbf{OU} \ (|\Gamma(u)| = |\Gamma(v)| \ \textbf{ET} \ u > v) &\Rightarrow v \in Fini(u) \\
candidats(u) &= Cand(u) \cup Fini(u), \\
Cand(u) \cap Fini(u) &= \emptyset
\end{aligned}$$

PECO, pour l'anglais *Parallel Enumeration of Cliques using Ordering*, est présenté dans l'algorithme 13.

Algorithme 13 $PECO(V)$

1: **pour tout** $u \in V$ **faire**
2: $\quad Tomita(u, Fini(u) \cup Cand(u), Cand(u))$
3: **fin pour**

La figure 2.9 présente un graphe pour lequel la répartition des voisins par noeud, dans les ensembles *Cand* ou *Fini*, est présentée dans le tableau 2.1. On remarque l'intérêt de répartir les *Cand* par degrés les plus petits, cela permet de distribuer la découverte des cliques sur plusieurs noeuds. Par exemple, dans la figure 2.9, si les noeuds étaient répartis par degrés les plus grands, les noeud 2 et 5 trouveraient 6 des 7 cliques à eux deux.

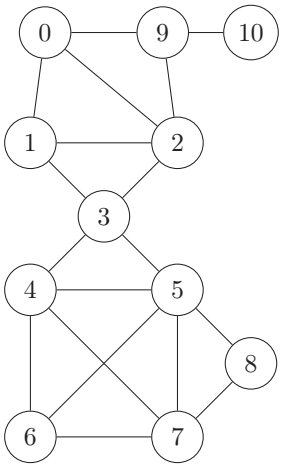


FIGURE 2.9 – Graphe exemple pour la répartition initiale des candidats, permettant une distributivité de l'énumération des cliques.

noeud	*Cand*	*Fini*	Cliques
0	{1,2,9}	∅	{{0,1,2},{0,2,9}}
1	{2,3}	{0}	{{1,2,3}}
2	{3}	{0,1,9}	∅
3	{4,5}	{1,2}	{{3,4,5}}
4	{5,7}	{3,6}	∅
5	∅	{3,4,6,7,8}	∅
6	{4,5,7}	∅	{{6,5,4,7}}
7	{5}	{4,6,8}	∅
8	{5,7}	∅	{{8,5,7}}
9	{2}	{0,10}	∅
10	{9}	∅	{{9,10}}

Tableau 2.1 – Répartition des candidats pour la distributivité et cliques trouvées par chacun des noeuds.

2.3.3 Adaptation d'algorithmes d'énumérations de cliques

Notre méthode d'énumération des λ-γ-cliques repose donc sur l'adaptation des algorithmes introduits par Svendsen et al. [105] et Tomita [111] avec les méthodes de sélections de noeuds pouvant être ajoutés à la quasi-clique courante selon les contraintes, imposées par les valeurs des paramètres λ_{min} et γ_{min}, introduites par Brunato et al.[19].

La conservation de candidats

Lors de l'agrandissement d'une quasi-clique il faut conserver en candidats les noeuds qui permettent toujours l'agrandissement de la quasi-clique en accord avec les contraintes imposées par les paramètres λ_{min} et γ_{min}.

Le graphe illustré en figure 2.10 présente ce cas de figure : si l'on cherche des λ-γ-cliques, avec $\lambda_{min} = \frac{2}{3}$ et $\gamma_{min} = \frac{2}{3}$, il convient de conserver le noeud 3 en candidat possible lors de l'ajout du noeud 2 à la quasi-clique courante {0,1} malgré le fait qu'il ne soit pas un voisin de ce noeud. En effet, le noeud 3 est initialement un candidat valable pour l'agrandissement de la clique formée par les noeuds 0 et 1. Dans le cas de la recherche de cliques on aurait éliminé le noeud 3 car ce n'est pas un voisin du noeud ajouté 2. Or la $\frac{2}{3}$-$\frac{5}{5}$-clique {0,1,2,3} est une clique valide en accord avec λ_{min} et γ_{min}.

L'élimination de candidats

En contre-exemple, certains noeuds voisins du dernier noeud ajouté et présents en candidats ne doivent pas être conservés car ils ne permettent pas la satisfaction d'une clique plus grande. Le graphe présenté en figure 2.11 permet de comprendre ce cas de figure. Si l'on cherche à énumérer des λ-γ-cliques, avec $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{5}{6}$, il convient d'éliminer le noeud 4 lors de l'ajout du noeud 3 pour la prochaine étape car il ne permet pas de conserver la contrainte de densité imposée par le paramètre γ_{min}. En effet $\gamma(\{0,1,2,3,4\}) = \frac{8}{10} < \lambda_{min} = \frac{5}{6}$. L'élimination de ce noeud permettant de détecter la quasi-clique {0,1,2,3} comme une quasi-clique maximale, car elle n'aura plus de noeuds candidats.

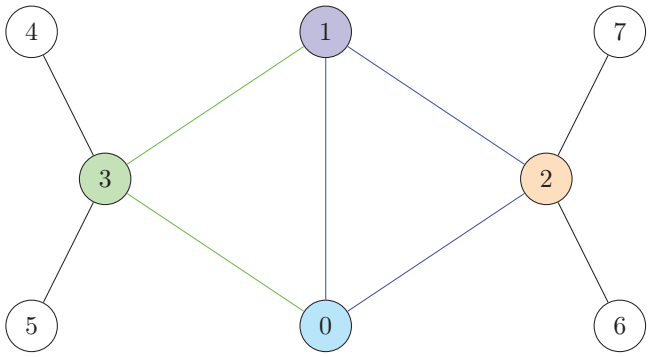


FIGURE 2.10 – Graphe exemple pour la conservation des candidats pour l'énumération de λ-γ-cliques, avec $\lambda_{min} = \frac{2}{3}$ et $\gamma_{min} = \frac{2}{3}$.

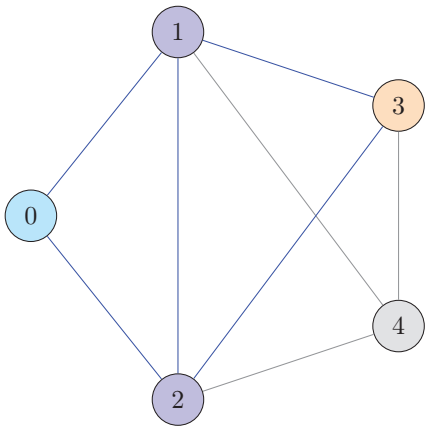


FIGURE 2.11 – Graphe exemple pour l'élimination des candidats pour l'énumération de λ-γ-cliques, avec $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{5}{6}$.

La découverte de nouveaux candidats

Il convient également de regarder, dans le voisinage du dernier noeud ajouté, quels sont les noeuds à ajouter dans les ensembles $Cand$ ou $Fini$. La figure 2.12 présente deux graphes présentant la découverte de candidats et leur attribution aux ensembles $Cand$ (à gauche) et $Fini$ (à droite).

En effet, si l'on part respectivement du noeud 0 à gauche et 3 à droite, et que l'on cherche à énumérer des λ-γ-cliques avec $\lambda_{min} = \frac{2}{3}$ et $\gamma_{min} = \frac{2}{3}$, il convient de regarder dans le voisinage du dernier noeud ajouté quels sont les candidats valides pour l'agrandissement de la quasi-clique courante. Dans le cas présenté figure 2.12 à gauche, lors de l'ajout du noeud 2, le noeud 3 (respectivement 0 à droite) doit être pris en considération car il permet de former une quasi-clique composée des noeuds $\{0,1,2,3\}$ avec $\lambda = \frac{2}{3}$ et $\gamma = \frac{5}{6}$. Afin d'éviter de lister deux fois la même quasi-clique il faut attribuer le nouveau noeud détecté à un ensemble, $Cand$ ou $Fini$. Étant donné que $|\Gamma(0)| = |\Gamma(3)|$ et $0 < 3$, le noeud 3 est ajouté à l'ensemble $Cand$ pour le noeud initial 0 et le noeud 0 est ajouté à l'ensemble $Fini$ pour le noeud initial 3. Ainsi seul le noeud initial 0 permet de détecter la quasi-clique $\{0,1,2,3\}$.

Le sous-graphe C_5

Une problématique se pose avec le *graphe cycle* C_5 présenté en figure 2.13. En effet, C_5 forme une quasi-clique avec $\lambda = \frac{1}{2}$ et $\gamma = \frac{1}{2}$ tout en étant composé de quasi-cliques de taille 4 dont la valeur λ est inférieure à la quasi-clique finale : $\lambda = \frac{1}{3}$.

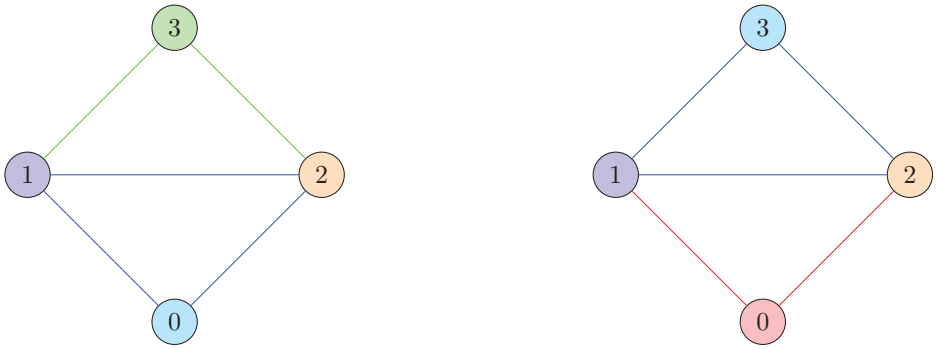


FIGURE 2.12 – Graphes exemples pour la découverte de nouveaux candidats dans l'énumération de λ-γ-cliques, avec $\lambda_{min} = \frac{2}{3}$ et $\gamma_{min} = \frac{2}{3}$.

Définition 2.14. Un graphe cycle C_x est un graphe composé de x noeuds {1,2,...,x} et x arêtes {(1,2),...,($x-1$,x),(x,1)}. Les λ-γ-cliques formées par de tels graphes ont des valeurs $\lambda = \frac{2}{x-1}$ et $\gamma = \frac{2 \cdot x}{x \cdot (x-1)}$.

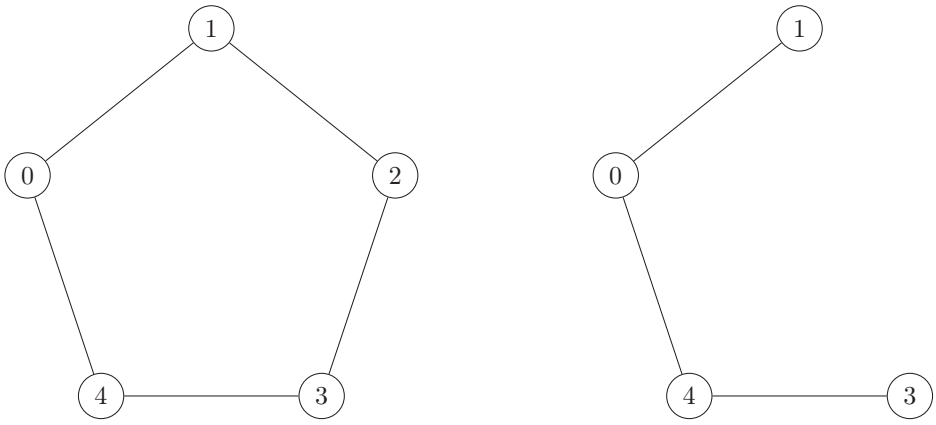


FIGURE 2.13 – Le graphe C_5 formant une λ-γ-clique, avec $\lambda = \frac{1}{2}$ et $\gamma = \frac{1}{2}$. À droite est présenté un sous-graphe de C_5 formant une quasi-clique avec $\lambda = \frac{1}{3}$ et $\gamma = \frac{3}{6}$.

Par conséquent l'exploration d'un graphe pour l'énumération de λ-γ-cliques, avec $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{1}{2}$ (présentée en figure 2.14) ne trouve que des quasi-cliques de taille 3 avec $\lambda = \frac{1}{2}$ et $\gamma = \frac{2}{3}$:

Si l'on part du noeud initial 0 *(a)*, il y a initialement deux candidats : les noeuds 1 et 4 *(b)*. Si l'on agrandit la clique avec le noeud 1 *(c)*, on forme une clique composée des noeuds {0,1}, on conserve le noeud 4 en candidat et on ajoute le noeud 2 en candidat potentiel.

Le problème se pose lors de l'ajout du noeud 2 *(d)*. En effet, les noeuds 3 et 4 sont éliminés car ils ne permettent pas, *individuellement*, de satisfaire la contrainte de degré minimal imposée par λ_{min}. On va alors déclarer la quasi-clique {0,1,2} comme quasi-clique maximale avec les contraintes imposées par λ_{min} et γ_{min}. Ce qui fait que l'étape suivante *(e)* va voir l'exploration effectuer un retour en arrière, et placer le noeud 2 dans l'ensemble $Fini$.

Par la suite on va ajouter le noeud 4 *(f)*, et se trouver dans la même situation qu'auparavant, à savoir l'élimination des noeuds voisins car ils ne permettent pas individuellement de satisfaire les contraintes.

La suite de l'exploration est constitué des étapes suivantes :

— le retour à la clique {0,1} et l'ajout du noeud 4 à l'ensemble $Fini$ *(g)*

— le retour à la situation initiale avec le noeud 1 dans l'ensemble $Fini$ *(h)*

— l'ajout du noeud 4 avec la conservation du noeud 1 et la découverte du noeud 3 *(i)*

— l'élimination des noeuds 1 et 2 et la découverte de la quasi-clique maximale {0,4,3} *(j)*
— le retour à la clique {0,4} et l'ajout du noeud 3 à l'ensemble $Fini$ *(k)*
— le retour à la situation initiale avec l'ensemble $Cand = \emptyset$ et la fin de l'exploration pour le noeud initial 0 *(l)*

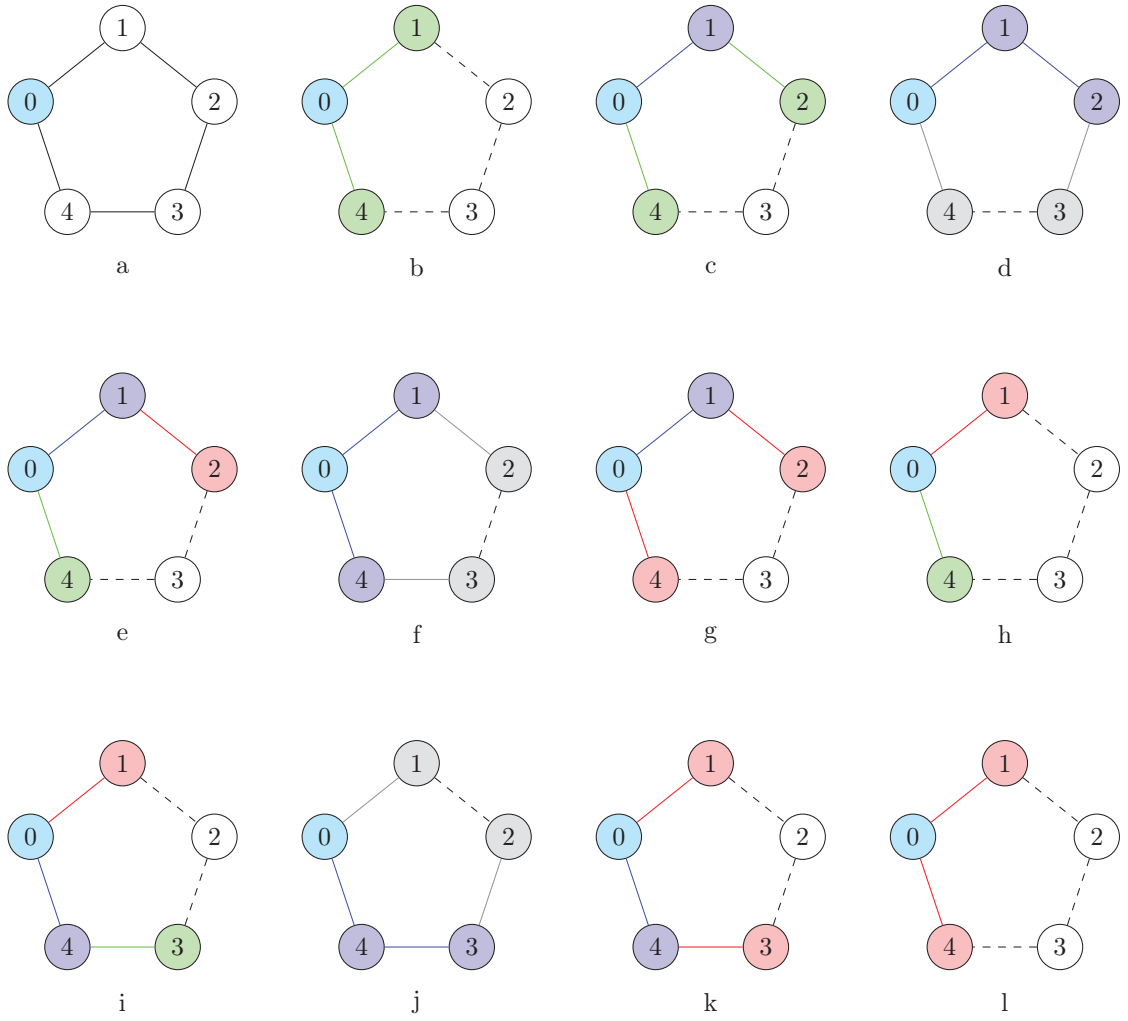


FIGURE 2.14 – Exploration du graphe C_5 à partir du noeud 0 lors de l'énumération de λ-γ-clique, avec $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{1}{2}$.

Ce problème est dû au fait que, temporairement, le sous-graphe ne respecte pas la contrainte de degré minimal imposée par λ_{min}. Ce cas de figure est également valable pour les algorithmes *RLS* et *DLS-MC* adaptés par Brunato et al. [19].

Afin de résoudre ce problème il convient de prendre en compte le voisinage des noeuds candidats entre eux pour la conservation ou la découverte des candidats.

Comme le présente la figure 2.15 il ne suffit pas de prendre en compte uniquement un candidat voisin pour l'évaluation des candidats : en effet, pour la quasi-clique courante {0,1,2,3,4,5} et l'étude du candidat 6, il convient de prendre en compte la totalité du voisinage entre la quasi-clique, le noeud candidat 6 et l'ensemble des autres candidats.

La quasi-clique courante forme une λ-γ-clique avec $\lambda = \frac{3}{5}$ et $\gamma = \frac{11}{15}$. L'ajout du noeud 6 forme une λ-γ-clique avec $\lambda = \frac{2}{6}$ et $\gamma = \frac{13}{21}$, ce qui ne permet pas de satisfaire les contraintes imposées par le paramètre de

voisinage minimal, $\lambda_{min} = \frac{1}{2}$. Cependant si l'on prend en compte l'ensemble restant des candidats possibles {7,8,9} on remarque que cela forme une λ-γ-clique avec $\lambda = \frac{5}{9}$ et $\gamma = \frac{20}{36}$, ce qui satisfait les contraintes. En conséquence le noeud 6 doit être pris en compte lors des étapes suivantes.

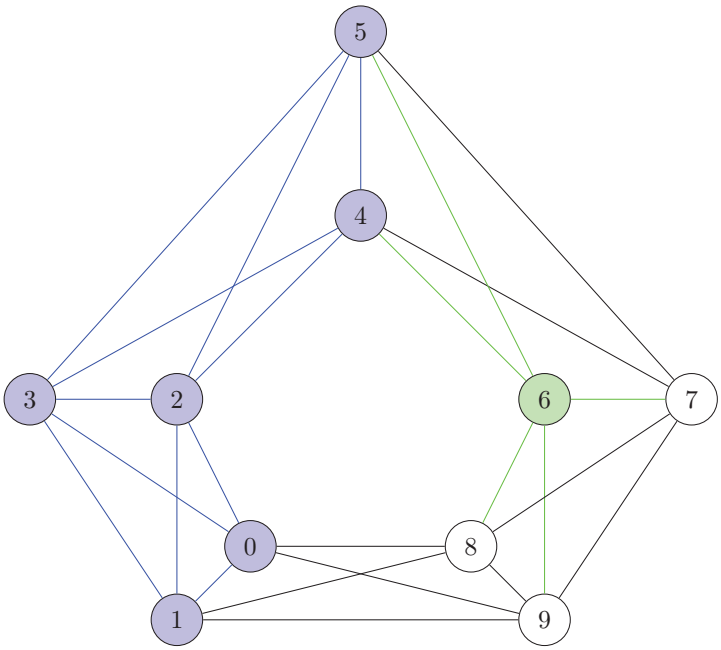


FIGURE 2.15 – Graphe exemple nécessitant l'étude de plus d'un candidat voisin pour la conservation ou la découverte de candidats lors de la recherche de λ-γ-clique, avec $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{1}{2}$.

Cependant la prise en compte de la totalité des candidats n'est pas toujours valable. Un exemple est présenté en figure 2.16. En partant de la λ-γ-clique courante {0,1,2} avec $\lambda = \frac{1}{2}$ et $\gamma = \frac{2}{3}$ et en étudiant le candidat 3 il convient de ne pas étudier l'ensemble des candidats voisins.

En effet, l'ajout du seul noeud 3 à la quasi-clique courante ne permet pas de satisfaire la contrainte de voisinage minimum. La quasi-clique formée {0,1,2,3} ayant une valeur $\lambda = \frac{1}{3}$, il convient d'étudier les candidats avec leur voisinage.

Cependant l'ajout de la totalité des autres candidats {4,5} forme une λ-γ-clique courante {0,1,2} avec $\lambda = \frac{2}{5}$ et $\gamma = \frac{8}{15}$, ce qui ne satisfait pas non plus la contrainte $\lambda_{min} = \frac{1}{2}$.

Il faut alors prendre en compte seulement une partie de l'ensemble du voisinage des candidats du noeud 3 à savoir le noeud 4 ou le noeud 5, ce qui permet de former des λ-γ-cliques avec $\lambda = \frac{2}{4}$ et $\gamma = \frac{5}{10}$.

Afin de déterminer quels sous-ensembles doivent être choisis pour la conservation ou non d'un candidat, nous proposons d'étudier les candidats voisins par le degré maximum qu'ils peuvent apporter lors de leur ajout à la quasi-clique.

La figure 2.17 présente ce cas de figure : si l'on considère la quasi-clique courante {0,1,2,3,4,5} et le candidat potentiel 6, on remarque que la λ-γ-clique courante C respecte les contraintes $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{1}{2}$ car $\lambda_C = \frac{3}{5}$ et $\gamma_C = \frac{11}{15}$. L'ajout du noeud 6 ne permet pas le respect des contraintes : $\lambda_{C+6} = \frac{2}{6}$ et $\gamma_{C+6} = \frac{13}{21}$. En prenant en compte l'ensemble des candidats restants {7,8,9,10,11}, on remarque que quel que soit le noeud ajouté à la quasi-clique $C + 6$, cela forme une λ-γ-clique avec $\lambda = \frac{3}{8}$ et $\gamma = \frac{16}{28}$. On peut donc prendre en compte une quasi-clique $C + 6 + x$, $x \in$ {7,8,9,10,11}.

Deux options sont alors possibles :

— $x \in$ {7,10} : n'importe quel noeud dans l'ensemble {8,9,11} permet de former une λ-γ-clique avec $\lambda = \frac{4}{8}$ et $\gamma = \frac{20}{36}$, ce qui respecte les contraintes λ_{min} et γ_{min}. Cependant si le noeud appartient à

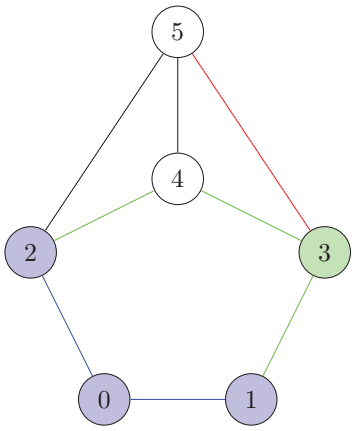


FIGURE 2.16 – Graphe exemple nécessitant de ne pas prendre en compte l'ensemble des candidats voisins pour la conservation ou la découverte de candidats lors de la recherche de λ-γ-clique, avec $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{1}{2}$.

$\{7,10\} - x$, cela forme une λ-γ-clique avec $\lambda = \frac{3}{8}$ et $\gamma = \frac{20}{36}$, ce qui ne permet pas le respect de la contrainte de voisinage λ_{min}.

— $x \in \{8,9,11\}$: n'importe quel noeud restant dans l'ensemble des $candidats - 6 - x$ permet de former une λ-γ-clique avec $\lambda = \frac{4}{8}$ et $\gamma = \frac{20}{36}$, ce qui respecte les contraintes λ_{min} et γ_{min}.

On remarque ainsi que le tri des candidats restants en fonction de leur apport permet de déterminer, et ce plus tôt que s'ils étaient considérés aléatoirement, si un noeud candidat doit être gardé ou non. Cela permet même un élagage car on détermine qu'une quasi-clique n'est pas maximale, sans tester la totalité des combinaisons de noeuds appartenant à l'ensemble des candidats. Dans notre exemple, $\{0,1,2,3,4,5,6,7,8\}$ permet de déterminer que C n'est pas maximale, alors qu'elle n'est elle-même pas maximale au regard de λ_{min} et γ_{min}, car elle peut être incluse dans une quasi-clique plus grande (notamment dans $\{0,1,2,3,4,5,6,7,8,9,10\}$).

En conséquence, nous arrivons à la situation où, après l'ajout du noeud u, un candidat c doit être conservé ou ajouté à la quasi-clique courante C si :

$$\begin{aligned} &\forall v_i \in (Cand(u) \cup Fini(u)) \cap \Gamma(c), \\ &\exists i \in [1,|(Cand(u) \cup Fini(u)) \cap \Gamma(c)|] \Rightarrow \lambda_{(C+u+c+\{v_1,\ldots,v_i\})} \geq \lambda_{min} \\ &\qquad\qquad \textbf{ET } \gamma_{(C+u+c+\{v_1,\ldots,v_i\})} \geq \gamma_{min} \end{aligned}$$

Les sous-graphe C_6 et plus

La solution proposée ne permet cependant pas de trouver des quasi-cliques avec des λ_{min} plus petits comme dans la figure 2.18. En effet, en regardant la classe des graphes cycles C_x, on remarque qu'il est nécessaire de prendre en considération des noeuds ou des arêtes qui ne sont pas voisins de la quasi-clique courante. Si l'on cherche à énumérer des λ-γ-cliques maximales satisfaisants les contraintes imposées par λ_{min} par ajout de noeuds successifs à la quasi-clique courante, on est amené à prendre en considération des éléments n'appartenant pas au voisinage direct de la quasi-clique courante :

— Pour le graphe C_6 : lors de l'ajout d'un 3ème noeud pour la recherche d'une λ-γ-clique, avec $\lambda = \frac{2}{5}$ et $\gamma = \frac{6}{15}$, on remarque qu'il est nécessaire de prendre en compte des noeuds voisins des noeuds candidats n'appartenant ni à l'ensemble de la quasi-clique courante ni à l'ensemble des candidats. Dans l'exemple donné dans la figure 2.18, il faut prendre en compte le noeud 3 afin de sélectionner les noeuds 2 et 4 comme candidats pour l'agrandissement de la quasi-clique à l'étape suivante (indifféremment que l'on vienne d'ajouter le noeud 1 ou le noeud 5). En effet, la quasi-clique courante satisfait les contraintes, $\frac{1}{2} \geq \lambda_{min}$ et $\frac{2}{3} \geq \gamma_{min}$, mais l'ajout du noeud 2 ou 4 ne satisfait pas la contrainte de voisinage

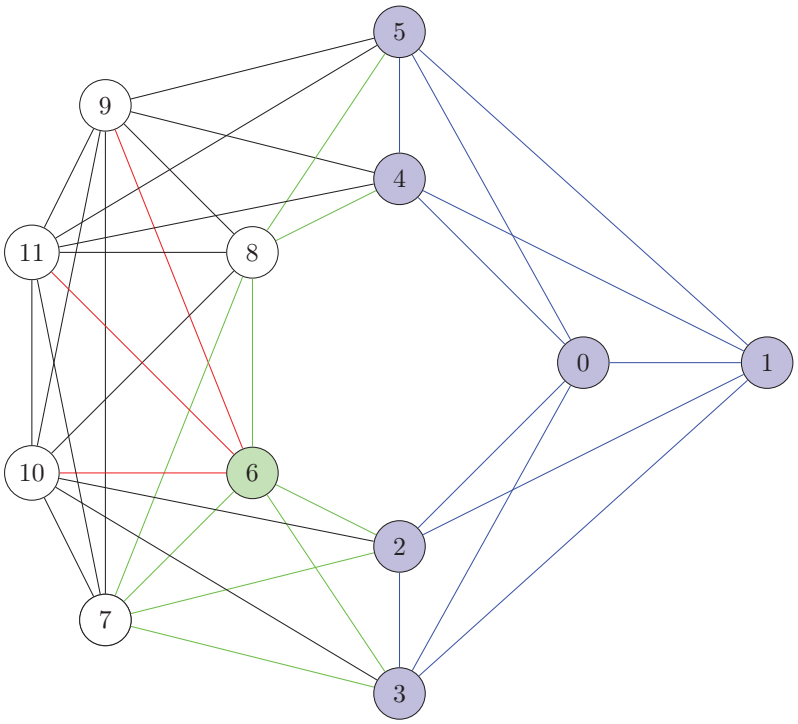


FIGURE 2.17 – Graphe exemple nécessitant de prendre en compte un sous-ensemble des candidats voisins pour la conservation ou la découverte de candidats lors de la recherche de λ-γ-clique, avec $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{1}{2}$.

minimal, $\frac{1}{3} < \lambda_{min}$. Par conséquent, afin de sélectionner les noeuds 2 et 4 comme candidats il faut prendre en compte des noeuds appartenant à leurs voisinages communs mais pas ceux appartenant à la quasi-clique courante.

— Pour le graphe C_7 : lors de l'ajout d'un 4ème noeud pour la recherche d'une λ-γ-clique, avec $\lambda = \frac{2}{6}$ et $\gamma = \frac{7}{21}$ on remarque qu'il est nécessaire de prendre en compte des noeuds voisins des noeuds candidats n'appartenant ni à l'ensemble de la quasi-clique courante ni à l'ensemble des candidats. Dans l'exemple donné dans la figure 2.18, il faut prendre en compte le noeud 4 afin de sélectionner les noeuds 3 et 5 comme candidats pour l'agrandissement de la quasi-clique à l'étape suivante (indifféremment que l'on vienne d'ajouter le noeud 2 ou le noeud 6). En effet, la quasi-clique courante satisfait les contraintes, $\frac{1}{3} \geq \lambda_{min}$ et $\frac{3}{6} \geq \gamma_{min}$, mais l'ajout du noeud 3 ou 5 ne satisfait pas la contrainte de voisinage minimal, $\frac{1}{4} < \lambda_{min}$. Par conséquent, afin de sélectionner les noeuds 3 et 5 comme candidats il faut prendre en compte des noeuds appartenant à leurs voisinages communs mais pas ceux appartenant à la quasi-clique courante.

— Pour le graphe C_8 : lors de l'ajout d'un 4ème noeud pour la recherche d'une λ-γ-clique, avec $\lambda = \frac{2}{7}$ et $\gamma = \frac{8}{28}$ on remarque qu'il est nécessaire de prendre en compte des arêtes entre les noeuds voisins des noeuds candidats n'appartenant ni à l'ensemble de la quasi-clique courante ni à l'ensemble des candidats. Dans l'exemple donné dans la figure 2.18, il faut prendre en compte l'arête entre les noeuds 4 et 5 afin de sélectionner les noeuds 3 et 6 comme candidats pour l'agrandissement de la quasi-clique à l'étape suivante (indifféremment que l'on vienne d'ajouter le noeud 2 ou le noeud 7). En effet, la quasi-clique courante satisfait les contraintes, $\frac{1}{3} \geq \lambda_{min}$ et $\frac{3}{6} \geq \gamma_{min}$ mais l'ajout du noeud 3 ou 6 ne satisfait pas la contrainte de voisinage minimal, $\frac{1}{4} < \lambda_{min}$. De plus, la prise en compte des noeuds voisins de 3 ou 6, les noeuds 4 et 5, ne permet pas à elle seule de définir l'existence d'un cycle. Il convient également de prendre en compte le voisinage des noeuds 4 et 5 entre eux pour satisfaire la contrainte de cycle. Par conséquent, afin de sélectionner les noeuds 3 et 6 comme candidats il faut

prendre en compte des arêtes liants des noeuds appartenant à leurs voisinages communs mais pas à ceux de la quasi-clique courante.

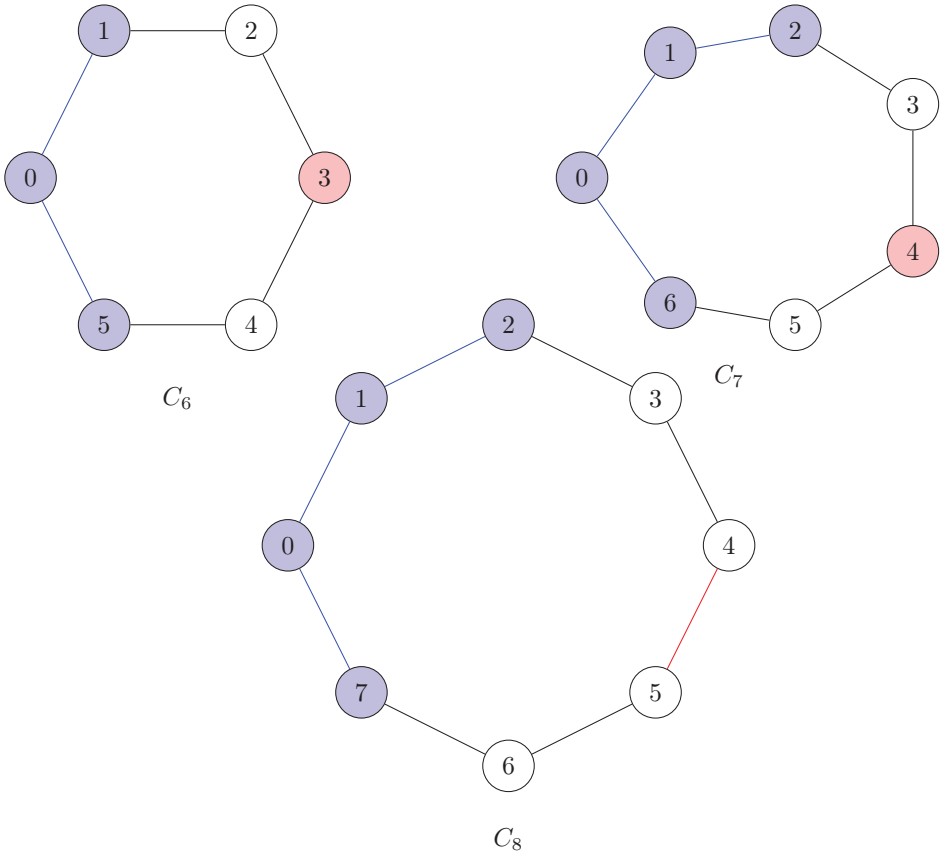


FIGURE 2.18 – Les graphes C_x formant une λ-γ-clique, avec $\lambda = \frac{2}{x-1}$ et $\gamma = \frac{2 \cdot x}{x \cdot (x-1)}$. En bleu est représentée la quasi-clique de plus grand taille trouvable par ajouts successifs de noeuds sans prise en compte des éléments en rouge.

Ces cas de figures imposent de prendre en considération des noeuds ou arêtes toujours plus distantes de la quasi-clique courante avec des λ_{min} tendant vers 0. Dans la suite de ce chapitre, sauf mention contraire, nous ne prendrons pas en compte de telles situations et prendrons en considération uniquement les arêtes liant les candidats entre eux.

La prise en compte des candidats non voisins

Un dernier cas est à prendre en compte lors de l'évaluation des candidats. Il est possible qu'un autre candidat non-voisin du candidat c étudié soit nécessaire pour sélectionner le candidat local.

Un exemple est présenté dans la figure 2.19. On cherche à énumérer des λ-γ-clique, avec $\lambda_{min} = \frac{3}{4}$ et $\gamma_{min} = \frac{8}{10}$, pour la quasi-clique courante $C = \{0,1,2\}$. Le dernier noeud ajouté est le noeud 2 et l'on évalue le noeud candidat 3. Si l'on ajoute directement 3 à la quasi-clique courante on obtient une quasi-clique avec $\lambda_{C+c} = \frac{2}{3}$, ce qui ne satisfait pas la contrainte imposée par le paramètre λ_{min}. Cependant l'ajout du noeud candidat 4, qui n'est pourtant pas voisin de 3 permet de satisfaire les contraintes, $\lambda_{C+c+4} = \frac{3}{4}$ et $\gamma_{C+c+4} = \frac{8}{10}$.

En conséquence nous arrivons à la situation où un candidat c doit être conservé ou ajouté, après l'ajout du noeud u à la quasi-clique courante C si :

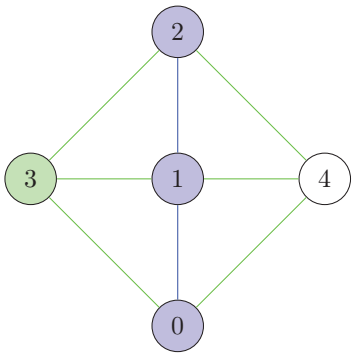


FIGURE 2.19 – Graphe exemple nécessitant de prendre en compte l'ensemble des candidats non-voisins pour la conservation ou la découverte de candidats lors de la recherche de λ-γ-clique, avec $\lambda_{min} = \frac{3}{4}$ et $\gamma_{min} = \frac{8}{10}$.

$$\begin{aligned} &\forall v_i \in Cand(u) \cup Fini(u), \\ \exists i \in [1,|Cand(u) \cup Fini(u)|] &\Rightarrow \lambda_{(C+u+c+\{v_1,\ldots,v_i\})} \geq \lambda_{min} \\ &\textbf{ET } \gamma_{(C+u+c+\{v_1,\ldots,v_i\})} \geq \gamma_{min} \end{aligned}$$

Détermination du sous-ensemble des candidats ajoutés lors de l'évaluation d'un candidat

Cette situation nous amène un nouveau problème, qui consiste à déterminer quels candidats doivent être sélectionnés pour l'évaluation d'un autre candidat.

Un exemple est présenté en figure 2.20. On cherche à énumérer des λ-γ-cliques, avec $\lambda_{min} = \frac{3}{4}$ et $\gamma_{min} = \frac{8}{10}$, pour la quasi-clique courante $C = \{0,1,2\}$. Le dernier noeud ajouté est le noeud 2 et l'on évalue le noeud candidat 3.

On remarque que l'ordre de prise en compte des candidats restants $candidats - c = \{4,5\}$ influe sur la sélection du pivot restant :

— En effet, 4 permet de satisfaire les contraintes pour la quasi-clique $\{0,1,2,3,4\}$, $\frac{3}{4} = \lambda_{min}$ et $\frac{8}{10} = \gamma_{min}$.

— Cependant, 5 ne permet pas de satisfaire les contraintes, la quasi-clique $\{0,1,2,3,5\}$ ne satisfaisant pas la contrainte de voisinage, $\frac{2}{4} < \lambda_{min}$, pas plus que la quasi-clique $\{0,1,2,3,4,5\}$, $\frac{3}{5} < \lambda_{min}$.

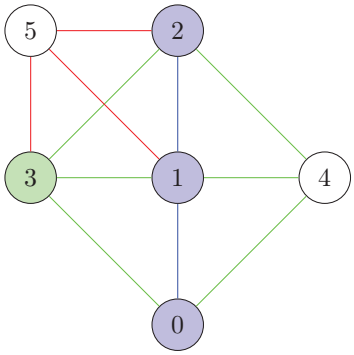


FIGURE 2.20 – Graphe exemple nécessitant de prendre en compte l'ordre des candidats pour la conservation ou la découverte de candidats lors de la recherche de λ-γ-clique, avec $\lambda_{min} = \frac{3}{4}$ et $\gamma_{min} = \frac{8}{10}$.

Nous posons le problème de décision associé *(EQC)* de l'*existence d'une quasi-clique* satisfaisant les contraintes des paramètres de voisinage et de densité.

Définition 2.15. Pour un graphe G contenant deux sous-ensembles disjoints de noeuds K et C et les paramètres λ_{min} et γ_{min}, le problème de l'existence d'une quasi-clique consiste à déterminer s'il existe un sous-ensemble $C^* \in C$ tel que G contient une quasi-clique $K + C^*$ satisfaisant les contraintes λ_{min} et γ_{min}.

Dans ce problème :
— K est la quasi-clique courante contenant le dernier noeud ajouté et le candidat c évalué
— C est l'ensemble des candidats privé du candidat évalué ($candidats - c$)
— λ_{min} est la contrainte de voisinage minimum
— γ_{min} est la contrainte de densité

Il convient donc de résoudre, pour chacun des candidats $c \in C$ le problème de l'existence d'une quasi-clique. Ce problème est cependant difficile.

Proposition 2.16. Le problème de décision de l'existence d'une quasi-clique présente dans un graphe est un problème NP-Complet.

Preuve. Le problème de décision de clique prend en entrée un graphe G et un entier k et il doit déterminer si G contient une clique de cardinal au moins égal à k. C'est un problème NP-Complet [59].

C'est une version particulière du problème de l'existence d'une quasi-clique où $\lambda_{min} = \gamma_{min} = 1$.

Cette version particulière étant NP-Complet, le problème de quasi-clique est NP-Complet dans le cas général. □

Afin d'être certain que le candidat peut être préservé, il faut donc résoudre pour chaque candidat un problème NP-Complet.

Le choix du pivot et la sélection des candidats

Afin d'élaguer certaines étapes de calcul nous avons gardé l'utilisation d'un pivot p qui maximise l'ensemble $Cand \cap \Gamma(p)$[111].

La sélection d'un noeud pivot p pour une clique de taille $|C|$ permet d'éliminer ses voisins des noeuds candidats de l'exploration de la clique car ils seront toujours présents dans l'ensemble des candidats pour la clique de taille supplémentaire $|C + p|$. Dans le cas des quasi-cliques, un exemple est présenté en figure 2.21.

En considération du graphe initial *(a)*, si l'on part du noeud 0 *(b)*, nos candidats sont les noeuds 1 et 2 qui appartiennent tous les deux au sous-ensemble $Cand$. Si l'on sélectionne le noeud 1 comme pivot dans l'ensemble des candidats, on élimine pour l'agrandissement de la quasi-clique courante {0} le noeud 2 car il est voisin du noeud 1 *(b_p)*.

Lors de l'ajout du noeud 1 à la quasi-clique courante, on recalcule les candidats et les noeuds 3 et 4 sont ajoutés à l'ensemble $Cand$ *(c)*. Lors du calcul du pivot pour la quasi-clique {0,1}, on est forcé de prendre le noeud 2, car il maximise l'intersection entre son voisinage et les candidats. Par conséquent, on élimine les noeuds 3 et 4 de l'exploration pour la quasi-clique {0,1} *(c_p)*.

Ensuite, lors de l'ajout du noeud 2, on conserve les noeuds 3 et 4 dans l'ensemble des $Cand$ *(d)*. Le calcul du pivot nous amène à indistinctement choisir le noeud 3 ou 4 comme pivot, ce qui préserve l'autre noeud de l'élimination pour l'exploration des agrandissements possibles de la quasi-clique {0,1,2} *(d_p)*.

Lors de l'ajout du noeud 3, on élimine le noeud 4 de l'ensemble des candidats. Ce dernier ne permettant pas de satisfaire les contraintes, on se retrouve alors avec une quasi-clique maximale *(e)*. On effectue donc un retour en arrière en transférant le noeud 3 dans l'ensemble des $Fini$ *(f)*. L'ajout du noeud 4 élimine le noeud 3 des $Fini$ et permet de découvrir une nouvelle quasi-clique maximale *(g)*. On effectue alors un premier retour en arrière *(h)* et l'ensemble Cand est dépourvu de noeuds. On effectue donc un second retour en arrière *(i)* pour se retrouver avec la quasi-clique {0,1} et le noeud 2 placé dans l'ensemble des $Fini$.

Les étapes *(j)* (ajout du noeud 3), *(k)* (retour en arrière), *(l)* (ajout du noeud 4), *(m)* (retour en arrière) sont éliminées par le calcul du pivot lors de l'étape *(c_p)*. En effet, lors de cette étape on avait éliminé l'exploration des quasi-cliques {0,1,3} et {0,1,4} par l'utilisation du pivot 2, voisin aux noeud 3 et 4.

Selon le même principe, les étapes *(o)*, *(p)*, *(q)*, *(r)* et *(s)* sont évitées par le calcul du pivot lors de l'étape *(b_p)*(car on avait éliminé l'exploration de la quasi-clique {0,2}).

L'utilisation d'un pivot permet donc d'élaguer les solutions explorées afin de réduire les temps de calculs.

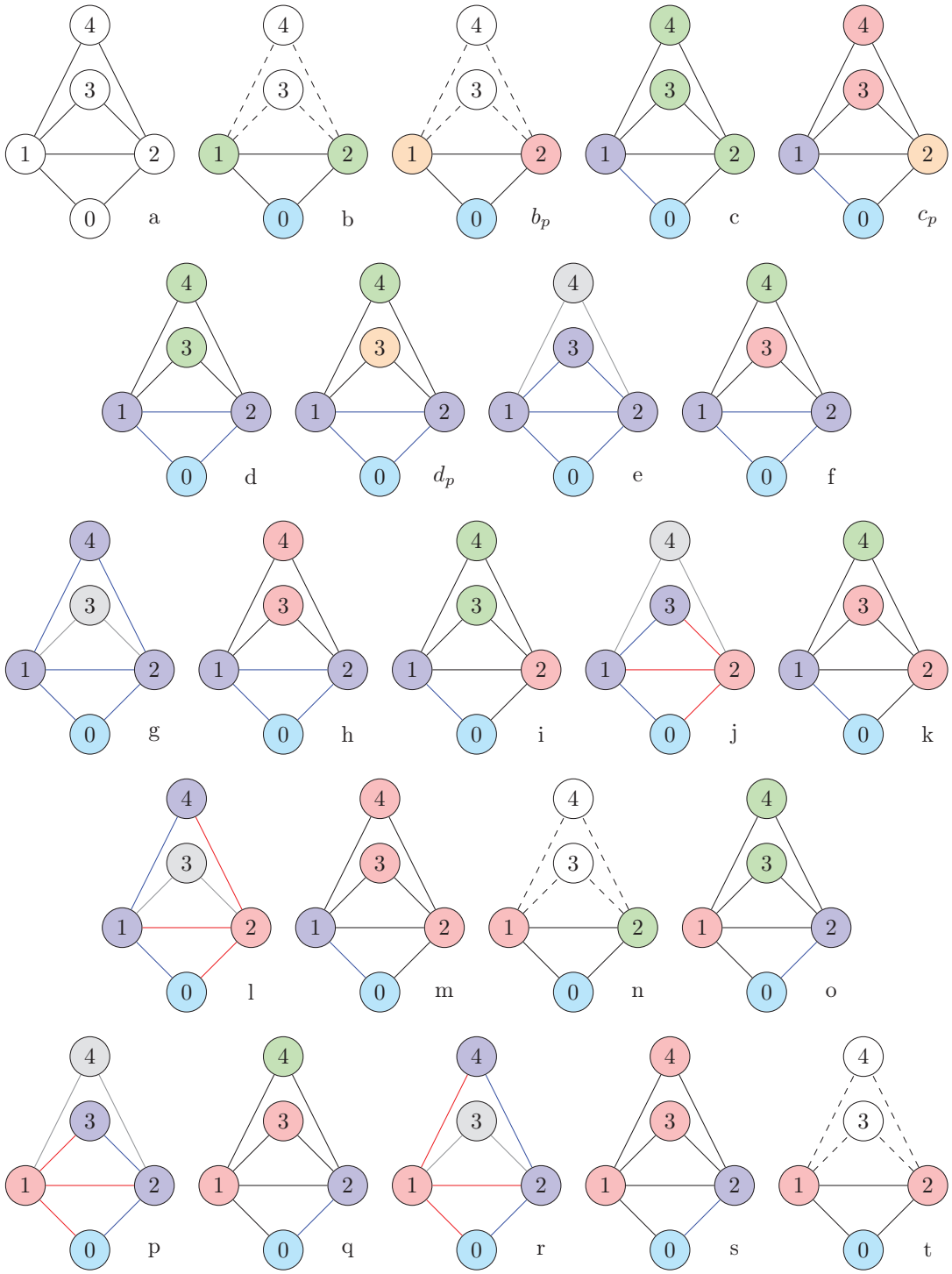


FIGURE 2.21 – Exploration d'un graphe et élagage par utilisation de pivot à partir du noeud 0 lors de l'énumération de λ-γ-clique, avec $\lambda_{min} = \frac{2}{3}$ et $\gamma_{min} = \frac{2}{3}$.

Cependant, certains des voisins du pivot p doivent être conservés dans les ajouts possibles à la quasi-clique courante C. C'est notamment le cas lorsque des noeuds ne satisfont plus les contraintes λ_{min} ou γ_{min}

lorsqu'ils sont ajoutés à la quasi-clique $C + p$ alors qu'ils satisfont les contraintes lorsqu'ils sont ajoutés à C.

La figure 2.22 montre un exemple : on cherche à énumérer des λ-γ-cliques, avec $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{5}{6}$.

En effet, si l'on prend en compte la quasi-clique courante $C = \{0,1,2\}$, on se retrouve avec les noeuds 3 et 4 comme candidats pour former des λ-γ-cliques, avec $\lambda = \frac{2}{3}$ et $\gamma = \frac{5}{6}$.

Si le noeud 3 est sélectionné comme pivot p, on est censé éliminer le noeud 4 et le prendre en considération uniquement lorsque le noeud 3 est ajouté à la quasi-clique courante. Toutefois la quasi-clique $C + 3 + 4$ ne satisfait pas la contrainte de densité minimale, $\frac{8}{10} < \gamma_{min}$. Il convient donc de préserver le noeud 4 des noeuds éliminés dans le voisinage du pivot.

Ainsi, parmi l'ensemble $Pres$ des noeuds préservés de l'élimination du voisinage du pivot se trouvent ceux satisfaisants la contrainte :

$$\begin{aligned} &\forall u \in Cand(C) \cap \Gamma(p), \\ &\qquad \lambda_{C+p+u} < \lambda_{min} \\ &\textbf{OU } \gamma_{C+p+u} < \gamma_{min} \Rightarrow u \in Pres \end{aligned}$$

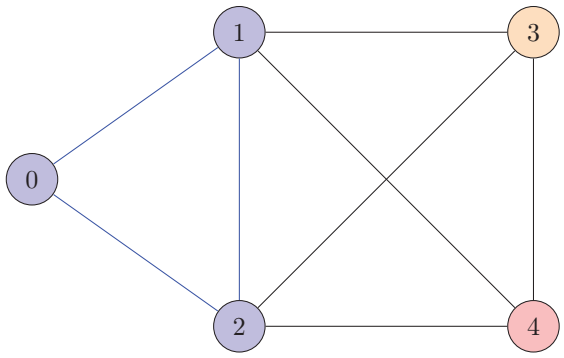


FIGURE 2.22 – Graphe exemple pour la conservation des voisins du pivot par ajout direct dans l'énumération de λ-γ-cliques, avec $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{5}{6}$.

Il faut également prendre en compte la découverte de nouveaux candidats avant d'éliminer les noeuds voisins du pivot.

La figure 2.23 présente un exemple : on cherche à énumérer des λ-γ-cliques, avec $\lambda_{min} = \frac{2}{3}$ et $\gamma_{min} = \frac{2}{3}$.

Si l'on part du noeud initial 0 et que le pivot choisi dans le voisinage est le noeud 1 (les noeuds 2 et 3 sont équivalents), on est censé éliminer les noeuds 2 et 3 (pour l'agrandissement de la quasi-clique $\{0\}$). Or la suppression de ces deux noeuds à cette étape empêche la découverte de la quasi-clique maximale $\{0,2,3,5\}$.

Par conséquent, il convient de garder comme noeuds ceux dont l'ajout à la quasi-clique courante permet la découverte de noeuds candidats n'appartenant pas à l'ensemble des noeuds candidats découverts par le pivot si ce dernier est ajouté à la quasi-clique courante.

Dans notre exemple figure 2.23, à partir de la quasi-clique courante bleue $\{0\}$ et avec la sélection du pivot 1, on découvre en candidats potentiels les noeuds 4 et 6. Cela permet à terme de former respectivement les quasi-cliques maximales $\{0,1,3,4\}$ et $\{0,1,2,6\}$.

Si l'on élimine les noeuds voisins du pivot (noeuds 2 et 3) pour l'agrandissement de la quasi-clique courante à une clique de taille 2, on ne trouve pas la quasi-clique $\{0,2,3,5\}$.

Toutefois, il est nécessaire de pas éliminer les noeuds en prenant en compte des candidats potentiellement ajoutés. Dans le cas présent la préservation des noeuds 2 et 3 est faite parce qu'ils permettent de découvrir le noeud candidat 5.

Ainsi, pour l'agrandissement de la quasi-clique C, on ajoute également à l'ensemble $Pres$ des noeuds préservés de l'élimination du voisinage du pivot ceux satisfaisants la contrainte :

$$\forall u \in Cand(C) \cap \Gamma(p),$$
$$\exists v,\ v \in (candidats(C+u) - candidats(C+p)) \Rightarrow u \in Pres$$

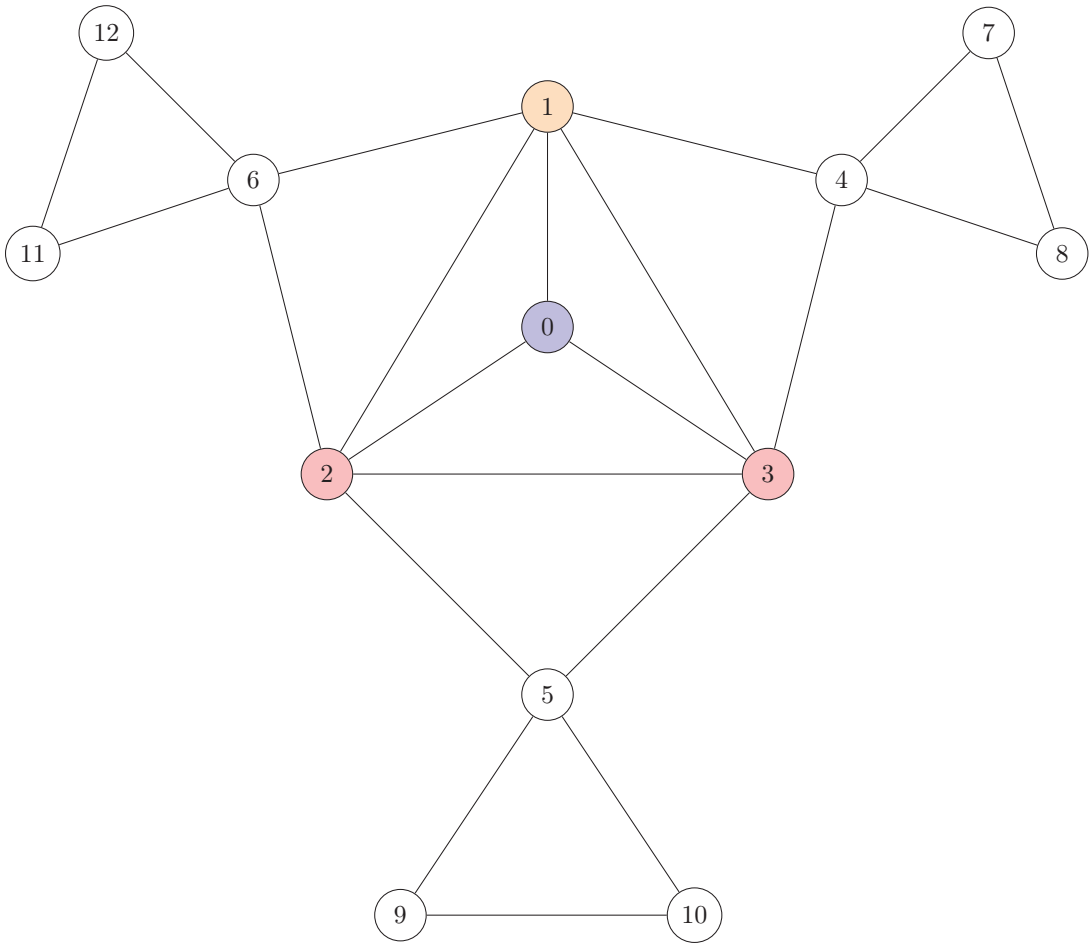


FIGURE 2.23 – Graphe exemple pour la conservation des voisins du pivot par découverte de potentiels candidats dans l'énumération de λ-γ-cliques, avec $\lambda_{min} = \frac{2}{3}$ et $\gamma_{min} = \frac{2}{3}$.

Il faut également éviter d'éliminer trop de candidats en accord avec les paramètres λ_{min} et γ_{min}. En effet, il se peut qu'on élimine de l'exploration trop de candidats dans le voisinage pour une quasi-clique courante qui, ajoutés ensembles, permettent l'élaboration d'une autre quasi-clique.

La figure 2.24*(a)* présente un exemple de cette situation pour la recherche de λ-γ-cliques, avec $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{11}{15}$ et la quasi-clique courante $C = \{0,1,2,3\}$. Dans ce cas de figure les noeuds 4, 5, 6 et 7 appartiennent à l'ensemble $Cand$. Le noeud 4 est celui maximisant $Cand \cap \Gamma(p)$, il est donc sélectionné comme pivot. Par conséquent, les noeuds 5, 6 et 7 doivent être éliminés de l'exploration à cette étape car ils sont voisins du pivot.

La conséquence de l'élimination des noeuds 5, 6 et 7 est que les λ-γ-cliques maximales $\{0,1,2,3,5,6\}$, $\{0,1,2,3,5,7\}$ et $\{0,1,2,3,6,7\}$, avec $\lambda = \frac{3}{5}$ et $\gamma = \frac{11}{15}$, ne seront pas trouvées. En effet, il est possible que l'ajout d'un sous-ensemble éliminé par le pivot permet de former une quasi-clique maximale.

Afin d'éviter que cette situation se réalise on cherche à éviter d'éliminer trop de candidats en calculant la taille de la quasi-clique maximale pouvant être théoriquement atteinte. Ce calcul est réalisé à partir de la

quasi-clique courante C, du pivot p, des candidats éliminés $Elim$ dans le voisinage du pivot et des valeurs des paramètres λ_{min} et γ_{min}.

Il est nécessaire de calculer, pour l'ensemble $C+p+Elim$, la taille maximale théorique d'une quasi-clique respectant les paramètres λ_{min} et γ_{min}.

Pour le paramètre de voisinage minimal λ, on peut calculer la taille de la quasi-clique maximale à l'aide de la valeur du paramètre λ_{min} et du sous-graphe induit par le sous-ensemble $C+p+Elim$. Pour cela on regarde quelle est la valeur du plus petit voisinage $\Gamma_{min}(C+p,Elim)$ pour les noeuds $u \in C+p$ dans le sous-graphe induit par l'ensemble $(C+p) \cup Elim$.

On définit la taille maximale théorique pouvant être atteinte par un sous-ensemble $S = C+p+Elim$ à partir de la quasi-clique courante $C+p$ par $m(S)$:

$$\begin{aligned}
\forall u \in C+p,\ \Gamma_{min} &= min\left(|\Gamma(u) \cap ((C+p) \cup Elim)|\right) \\
\frac{\Gamma_{min}}{m-1} &= \lambda_{min} \\
m-1 &= \frac{\Gamma_{min}}{\lambda_{min}} \\
m &= \left\lfloor \Gamma_{min} \cdot \lambda_{min}^{-1} \right\rfloor + 1
\end{aligned}$$

Dans l'exemple donné figure 2.24*(a)* c'est le noeud 0 qui possède le plus petit degré $\Gamma_{min} = 3$.

$$\begin{aligned}
m &= \left\lfloor 3 \cdot \frac{2}{1} \right\rfloor + 1 \\
&= 6+1 \\
&= 7
\end{aligned}$$

La quasi-clique maximale trouvée peut donc avoir une taille maximale de 7, 0 devant être voisin avec aux moins la moitié des 6 autres noeuds composants la quasi-clique selon λ_{min}. La taille du sous-ensemble $|C+p+Elim|$ étant de 8 il convient de retirer un élément de $Elim$. En retirant le noeud 7 de $Elim$ on trouvera les quasi-cliques {0,1,2,3,5,7} et {0,1,2,3,6,7}.

En continuant dans cet exemple, on remarque que la figure *(b)* satisfait la contrainte de taille minimale selon le paramètre λ_{min}.

Pour le paramètre de densité minimal γ, on peut calculer la taille de la quasi-clique maximale à l'aide de la valeur du paramètre γ_{min} et du sous-graphe induit par le sous-ensemble $(C+p) \cup Elim$. Pour cela on regarde quel est le nombre d'arêtes e dans le sous-graphe $(C+p) \cup Elim$.

On définit la taille maximale théorique pouvant être atteinte par un sous-ensemble $S = (C+p) \cup Elim$ par $n(S)$:

$$\begin{aligned}
\forall u,v \in (C+p) \cup Elim,\ e &= |(u,v) \in E| \\
n \cdot (n-1) &= \frac{2 \cdot e}{\gamma_{min}} \\
n^2 - n &= 2 \cdot e \cdot \gamma_{min}^{-1}
\end{aligned}$$

On se retrouve avec une équation du second degré où :

$$\begin{aligned}
a &= 1 \\
b &= -1 \\
c &= -(2 \cdot e \cdot \gamma_{min}^{-1})
\end{aligned}$$

Ainsi :

$$\begin{aligned}\Delta &= b^2 - 4ac \\ &= 1 - 4c\end{aligned}$$

c étant négatif,

$$\Delta > 1$$

On a donc comme solutions :

$$\begin{aligned}n &= \frac{-b \pm \sqrt{\Delta}}{2a} \\ &= \frac{1 \pm \sqrt{\Delta}}{2}\end{aligned}$$

Δ étant supérieur à 1, on ne prend pas en compte la solution négative, ainsi :

$$\begin{aligned}n &= \left\lfloor \frac{1 + \sqrt{\Delta}}{2} \right\rfloor \\ &= \left\lfloor \frac{1 + \sqrt{1 + 4 \cdot (2 \cdot e \cdot \gamma_{min}^{-1})}}{2} \right\rfloor\end{aligned}$$

Dans l'exemple donné figure 2.24*(b)* le nombre d'arêtes du graphe induit par le sous-ensemble $C+p+Elim$ est $e = 15$.

$$\begin{aligned}n &= \left\lfloor \frac{1 + \sqrt{1 + 4 \cdot (30 \cdot \frac{15}{11})}}{2} \right\rfloor \\ &= \left\lfloor \frac{1 + \sqrt{\frac{1811}{11}}}{2} \right\rfloor \\ &= \lfloor 6{,}92 \rfloor \\ &= 6\end{aligned}$$

La quasi-clique maximale trouvée peut donc avoir une taille maximale de 6 pour satisfaire la contrainte de densité minimale λ_{min}. La taille du sous-ensemble $|C + p + Elim|$ étant de 7 il convient donc de retirer un élément de $Elim$. En retirant le noeud 6 de $Elim$ on trouvera la quasi-clique $\{0,1,2,3,5,6\}$.

Ainsi on ajoute à l'ensemble $Pres$ (noeuds préservés de l'élimination du voisinage du pivot $Elim$ pour l'agrandissement de la quasi-clique C) ceux satisfaisant la contrainte :

$$\begin{aligned}&\forall u_i \in Elim,\ \exists i_{max} \in [0,|Elim|] \Rightarrow S = C + p + Elim - \{u_1,...,u_{i_{max}}\} \\ &\text{tel que } |S| \leq min(m(S),n(S)) \Rightarrow \{u_1,...,u_{i_{max}}\} \in Pres\end{aligned}$$

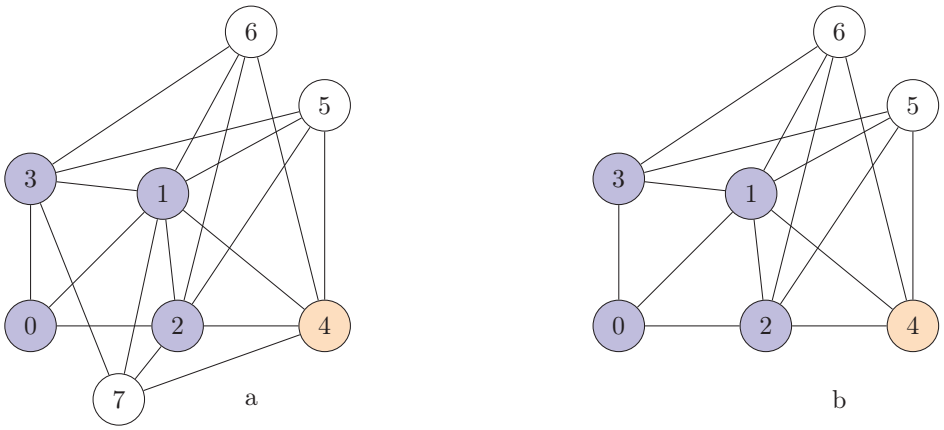


FIGURE 2.24 – Graphe exemple pour la conservation des voisins du pivot par préservation de quasi-clique maximale dans l'énumération de λ-γ-cliques, avec $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{11}{15}$.

La connectivité

Un dernier cas est à prendre en compte, les quasi-cliques non-connexes. Un exemple est présenté en figure 2.25. Si l'on cherche à énumérer l'ensemble des λ-γ-cliques, avec $\lambda_{min} = \frac{1}{3}$ et $\gamma_{min} = \frac{2}{6}$, on doit théoriquement trouver la quasi-clique {0,1,2,3,4,9}. Toutefois l'ensemble {2,3,9} n'étant pas connexe il n'est pas possible de découvrir comme nouveaux candidats les noeuds 2, 3 ou 9.

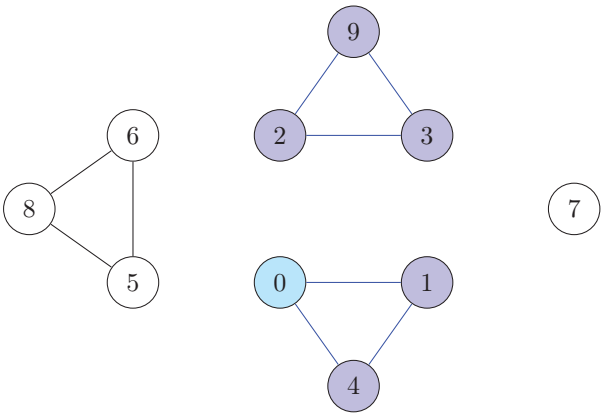


FIGURE 2.25 – Exemple de quasi-clique non-connexe pour l'énumération de λ-γ-cliques, avec $\lambda_{min} = \frac{1}{3}$ et $\gamma_{min} = \frac{2}{6}$.

Une solution permettant de prendre en compte ce cas de figure serait de considérer l'ensemble des noeuds n'appartenant pas à la quasi-clique pour l'étude des candidats.

Ainsi lorsque l'on ajoute le noeud $u = 1$ au cas présenté figure 2.25 et que l'on considère l'ensemble des autres noeuds candidats {2,3,4,5,6,8,9} (le noeud 7 est éliminé car il a un plus petit degré que le noeud source 0), alors :

Pour $c = 4$:

$$\lambda_{C+u+c} = \frac{2}{2} > \lambda_{min}$$
$$\textbf{ET } \gamma_{C+u+c} = \frac{3}{3} > \gamma_{min}$$

Pour les autres noeuds $c \in \{2,3,5,6,8,9\}$ (en exemple $c = 2$) :

$$\lambda_{C+u+c} = \frac{0}{2} < \lambda_{min}$$
$$\textbf{ET } \gamma_{C+u+c} = \frac{1}{3} = \gamma_{min}$$

L'ajout direct n'est pas suffisant, il convient donc de prendre en considération le voisinage de c parmi les candidats :

$$\Gamma(c) \cap candidates = \{3,9\}$$

Ainsi :

$$\forall i \in [1,|\Gamma(c) \cap candidates|],\ i \in [1,2]$$

Avec $i = 1$:

$$\lambda_{C+u+c+v_1} = \frac{1}{3} = \lambda_{min}$$
$$\textbf{ET } \gamma_{C+u+c+v_1} = \frac{2}{6} = \gamma_{min}$$

Ainsi pour $v_1 = 3$ ou $v_1 = 9$, les contraintes des paramètres λ_{min} et γ_{min} sont respectées, ce qui est suffisant pour préserver le noeud 2

$$2 \in Cand$$

Nous remarquons que l'étude de tels cas de figures est extrêmement contraignante : il faut vérifier à chaque étape l'ensemble des noeuds du graphes en les considérant comme candidats, et pour chacun résoudre un problème NP-Complet. C'est pourquoi, dans la suite de ce chapitre, sauf mention contraire, nous ne prendrons en considération que les quasi-cliques connexes et ne testerons que les candidats voisins à la quasi-clique courante.

2.3.4 Détails Algorithmiques

Dans cette section nous donnerons les algorithmes permettant d'énumérer l'ensemble des λ-γ-cliques pour des paramètres λ_{min} et γ_{min}.

La fonction principale

La fonction principale de notre méthode, que nous nommerons *PEQCO* pour *Parallel Enumeration of Quasi-Cliques using Ordering*, est présentée dans l'algorithme 14. Cet algorithme permet de classer pour chacun des voisins d'un noeud s'il est dans la liste $Cand$ ou $Fini$ *(lignes 2 à 8)*, ce qui garanti l'absence de doublons dans le résultat final. De plus cela permet de distribuer le calcul sur plusieurs machines, chacune exécutant l'algorithme pour un noeud initial différent.

Une fois les noeuds voisins classés, on exécute la fonction d'énumération $\lambda\gamma Tomita$ *(ligne 9)* avec comme quasi-clique courante l'ensemble composé du noeud initial.

Algorithme 14 $PEQCO(V, \lambda_{min}, \gamma_{min})$

1: **pour tout** $u \in V$ **faire**
2: **pour tout** $v \in \Gamma(u)$ **faire**
3: **si** $|\Gamma(u)| < |\Gamma(v)|$ **OU** $(|\Gamma(u)| = |\Gamma(v)|$ **ET** $u < v)$ **alors**
4: $Cand \leftarrow Cand + v$
5: **sinon**
6: $Fini \leftarrow Fini + v$
7: **fin si**
8: **fin pour**
9: $\lambda\gamma Tomita(\{u\}, Cand, Fini, \lambda_{min}, \gamma_{min}, u)$
10: **fin pour**

La sous-fonction récursive

La fonction $\lambda\gamma Tomita$ est la sous-fonction principale de notre méthode. C'est elle qui va se charger de déterminer si la quasi-clique courante est maximale. Si ce n'est pas le cas, elle va calculer parmi les candidats quels noeuds sont à explorer et pour ces noeuds quels sont leurs propres candidats. Cette fonction est récursive et permet d'augmenter noeud par noeud la taille de la quasi-clique.

Dans un premier temps on regarde s'il reste des noeuds candidats dans l'ensemble $Cand \cup Fini$. Si cet ensemble est vide c'est qu'il n'est pas possible d'ajouter d'autres noeuds à la quasi-clique courante en respectant les contraintes λ_{min} et γ_{min}. Cela indique que la quasi-clique courante est une quasi-clique maximale. Dans ce cas on retourne la quasi-clique courante comme résultat satisfaisant les contraintes *(lignes 1 à 3)*.

Si la quasi-clique n'est pas maximale, on calcule alors un noeud pivot p pour lequel on regarde quels candidats parmi ses voisins sont à préserver avec la fonction $PreservationVoisinsPivot$. Ainsi on défini un ensemble Ext composé des candidats non-explorés $Cand$ auquel on retire les voisins non préservés du pivot *(lignes 4 et 5)*.

Enfin, pour chacun des noeuds de l'ensemble $u \in Ext$, on calcule les nouveaux candidats et on appelle la fonction $\lambda\gamma Tomita$ sur une nouvelle quasi-clique courante $C + u$. Une fois toutes les quasi-cliques possibles explorées à partir de $C + u$, on enlève u de la liste des candidats non-explorés $Cand$ pour l'ajouter à celle des candidats explorés $Fini$ *(lignes 6 à 11)*.

La sélection des candidats

Lors de l'ajout d'un noeud u on doit calculer pour chacun des noeuds voisins de la quasi-clique C ceux sont à prendre en considération pour l'étape suivante. La fonction permettant de déterminer les candidats sélectionnés est présentée en algorithme 16.

Ainsi dans un premier temps on définit 3 ensembles :

— La liste des candidats à étudier qui sont les noeuds voisins de la clique courante C *(ligne 1)*
— La liste des noeuds à garder, initialement vide *(ligne 2)*
— La liste des noeuds dont on est sûr qu'ils ne peuvent pas être ajoutés, initialement vide *(ligne 3)*

Ensuite, pour chacun des noeuds $v \in Candidats$ qui n'ont pas été ajoutés, on va tester si leur ajout à la quasi-clique courante permet la satisfaction des contraintes λ_{min} et γ_{min}. Dans ce cas on ajoute le noeud v à l'ensemble $CandidatsAcceptes$ *(ligne 6 et 7)*.

Algorithme 15 $\lambda\gamma Tomita(C, Cand, Fini, \lambda_{min}, \gamma_{min}, initNode)$

1: **si** $Cand \cup Fini = \emptyset$ **alors**
2: retourner C comme quasi-clique maximale
3: **fin si**
4: choisir $p \in Cand \cup Fini$ qui maximise la taille $|Cand \cap \Gamma(p)|$
5: $Ext \leftarrow (Cand - (\Gamma(p) - PreservationVoisinsPivot(p, C, Cand, Fini, \lambda_{min}, \gamma_{min}, initNode))$
6: **pour tout** $u \in Ext$ **faire**
7: $(validCand, validFini) \leftarrow SelectionCandFinis(C + u, Fini - u, \lambda_{min}, \gamma_{min}, initNode)$
8: $\lambda\gamma Tomita(C + u, validCand, validFini, \lambda_{min}, \gamma_{min}, initNode)$
9: $Cand \leftarrow Cand - u$
10: $Fini \leftarrow Fini + u$
11: **fin pour**

Si l'ajout direct du noeud ne permet pas de satisfaire les contraintes on doit regarder si l'ajout d'autres noeuds candidats à la quasi-clique $C+v$ permet de satisfaire les contraintes λ_{min} et γ_{min}. Pour cela on utilise la fonction $AjoutAutresCandidats$ qui retourne un ensemble $Ret \in Candidats$ de telle sorte que l'ensemble $(C + v) \cup Ret$ satisfasse les contraintes *(ligne 9)*.

On regarde donc l'ensemble Ret retourné par la fonction $AjoutAutresCandidats$. Si cet ensemble est vide c'est qu'aucun candidat ne permet de satisfaire les contraintes et donc que le noeud v n'est pas un candidat valide. Dans ce cas on ajoute v à l'ensemble $CandidatsElimines$, ce qui permettra aux noeuds suivants de ne pas le prendre en considération *(ligne 13)*. Sinon, si l'ensemble Ret n'est pas vide, on ajoute cet ensemble et le noeud v à la l'ensemble des $CandidatsAcceptes$ *(ligne 11)*. Ceci permet d'éviter aux noeuds présents dans Ret d'être eux-mêmes évalués si cela n'a pas déjà été fait *(ligne 5)*.

Finalement on regarde pour chacun des noeuds de l'ensemble $CandidatsAcceptes$ s'ils appartiennent à l'ensemble $Cand$ ou $Fini$ *(ligne 20 à 26)* en prenant notamment en compte le fait que le noeud n'a pas encore été visité *(ligne 21)*.

L'ajout de candidats

Il est parfois nécessaire de vérifier que, pour un candidat u et une quasi-clique C dont la quasi-clique $C + u$ ne satisfait pas les contraintes λ_{min} et γ_{min}, l'ajout d'autres noeuds candidats permet de satisfaire les contraintes. La fonction $AjoutAutresCandidats$, présentée en algorithme 17, permet de détecter si un ensemble $Ret \in Candidate$ permet de satisfaire les contraintes.

Pour cela on trie les candidats par la valeur du paramètre de voisinage qu'ils apportent s'ils sont ajoutés à la quasi-clique courante. Cela permet de favoriser les noeuds donnant potentiellement plus de chances de trouver des quasi-cliques satisfaisant les contraintes plus rapidement *(ligne 1)*.

Puis, pour chacun de ces noeuds $v \in Candidats$, on teste si leur ajout à la quasi-clique permet de satisfaire les contraintes λ_{min} et γ_{min} et, le cas échéant, on retourne le noeud v *(ligne 9 et 10)*. Sinon on teste si l'ajout d'autres noeuds parmi les candidats restants permet de trouver une quasi-clique satisfaisant les contraintes en appelant récursivement la fonction d'ajout de candidats *(lignes 11 à 16)*.

Afin d'éviter de tester plusieurs fois les mêmes combinaisons de candidats, on retire ceux pour lesquels on a pas trouvé de quasi-clique satisfaisante *(ligne 3)*.

Enfin on regarde également s'il est possible de trouver des quasi-cliques plus grandes à partir de la quasi-clique courante $C + v$ tout en respectant les contraintes de voisinage *(ligne 4)* et de densité *(ligne 5)*. Si cela n'est théoriquement pas possible, il n'est pas nécessaire d'ajouter de nouveaux noeuds et on peut arrêter l'exploration *(ligne 6 à 8)*.

La préservation des candidats voisins du pivot

Il faut également éviter d'éliminer certains candidats voisins du pivot p. La fonction $PreservationVoisins$-$Pivot$, présentée en algorithme 18, permet de déterminer quel ensemble de noeuds parmi le voisinage du pivot est à prendre en considération pour la quasi-clique courante C.

Algorithme 16 *SelectionCandFinis*(C, $Fini$, λ_{min}, γ_{min}, $initNode$)

1: $Candidats \leftarrow \Gamma(C)$
2: $CandidatsAcceptes \leftarrow \emptyset$
3: $CandidatsElimines \leftarrow \emptyset$
4: **pour tout** $v \in Candidats$ **faire**
5: **si** $v \notin CandidatsAcceptes$ **alors**
6: **si** $\lambda_{C+v} \geq \lambda_{min}$ **ET** $\gamma_{C+v} \geq \gamma_{min}$ **alors**
7: $CandidatsAcceptes \leftarrow CandidatsAcceptes + v$
8: **sinon**
9: $Ret \leftarrow AjoutAutresCandidats(C + v, Candidats - v - CandidatsElimines, \lambda_{min}, \gamma_{min})$
10: **si** $Ret \neq \emptyset$ **alors**
11: $CandidatsAcceptes \leftarrow (CandidatsAcceptes + v) \cup Ret$
12: **sinon**
13: $CandidatsElimines \leftarrow CandidatsElimines + v$
14: **fin si**
15: **fin si**
16: **fin si**
17: **fin pour**
18: $CandN \leftarrow \emptyset$
19: $FiniN \leftarrow \emptyset$
20: **pour tout** $v \in CandidatsAcceptes$ **faire**
21: **si** $v \notin Fini$ **ET** $(|\Gamma(initNode)| < |\Gamma(v)|$ **OU** $(|\Gamma(initNode)| = |\Gamma(v)|$ **ET** $u < v))$ **alors**
22: $CandN \leftarrow CandN + v$
23: **sinon**
24: $FiniN \leftarrow FiniN + v$
25: **fin si**
26: **fin pour**
27: retourner $(CandN, FiniN)$

Algorithme 17 *AjoutAutresCandidats*(C, $Candidats$, λ_{min}, γ_{min})

1: trier l'ensemble des $u \in Candidats$ par la valeur λ_{C+u}
2: **pour tout** $v \in Candidats$ **faire**
3: $Candidats \leftarrow Candidats - v$
4: $m \leftarrow \lfloor \Gamma_{min}(C + v, Candidats) \cdot \lambda_{min}^{-1} \rfloor + 1$
5: $n \leftarrow \left\lfloor \frac{1+\sqrt{1+4\cdot(2\cdot e\cdot\gamma_{min}^{-1})}}{2} \right\rfloor$
6: **si** $|C + v| \geq min(m,n)$ **alors**
7: retourner $\emptyset$
8: **fin si**
9: **si** $\lambda_{C+v} \geq \lambda_{min}$ **ET** $\gamma_{C+v} \geq \gamma_{min}$ **alors**
10: retourner $\{v\}$
11: **sinon**
12: $Ret \leftarrow AjoutAutresCandidats(C + v, Candidats, \lambda_{min}, \gamma_{min})$
13: **si** $Ret \neq \emptyset$ **alors**
14: retourner $\{v\} \cup Ret$
15: **fin si**
16: **fin si**
17: **fin pour**
18: retourner $\emptyset$

Pour cela on regarde parmi les candidats voisins du pivot qui n'ont pas encore été explorés $Cand \cap \Gamma(p)$ s'ils permettent de satisfaire les contraintes pour la quasi-clique courante C mais pas pour la quasi-clique à laquelle on a ajouté le pivot *(lignes 4 et 5)*. Sinon on regarde parmi ces noeuds s'ils permettent d'apporter des

candidats différents du pivot *(lignes 7 à 10)*. Si une de ces situations est rencontrée, le candidat est préservé de l'élimination et est ajouté à l'ensemble $CandidatsPreserves$, qui est initialement vide *(ligne 1)*.

Enfin, on vérifie parmi les noeuds restants $(Cand \cap \Gamma(p)) - CandidatsPreserves$ que l'on n'élimine pas trop de candidats par rapport à la quasi-clique courante à l'aide de la fonction $PreservationTropElimination$. On retourne finalement un ensemble correspondant à l'union entre le retour de cette fonction et les voisins apportant de nouveaux candidats *(ligne 13)*.

Algorithme 18 $PreservationVoisinsPivot(p, C, Cand, Fini, \lambda_{min}, \gamma_{min}, initNode)$

1: $CandidatsPreserves \leftarrow \emptyset$
2: $(PivotCand, PivotFini) \leftarrow SelectionCandFinis(C + p, Fini, \lambda_{min}, \gamma_{min}, initNode)$
3: **pour tout** $u \in (Cand \cap \Gamma(p)$ **faire**
4: **si** $(\lambda_{C+p+u} < \lambda_{min}$ **OU** $\gamma_{C+p+u} < \gamma_{min})$ **ET** $\lambda_{C+u} \geq \lambda_{min}$ **ET** $\gamma_{C+u} \geq \gamma_{min}$ **alors**
5: $CandidatsPreserves \leftarrow CandidatsPreserves + u$
6: **sinon**
7: $(NoeudCand, NoeudFini) \leftarrow SelectionCandFinis(C + u, Fini, \lambda_{min}, \gamma_{min}, initNode)$
8: **si** $NoeudCand - PivotCand \neq \emptyset$ **alors**
9: $CandidatsPreserves \leftarrow CandidatsPreserves + u$
10: **fin si**
11: **fin si**
12: **fin pour**
13: retourner $CandidatsPreserves \cup$
$PreservationTropElimination(C + p, (Cand \cap \Gamma(p)) - CandidatsPreserves, \lambda_{min}, \gamma_{min})$

La préservation de l'élimination de trop nombreux candidats voisins du pivot

Il faut également éviter d'éliminer trop de candidats voisins du pivot p. La fonction $PreservationTrop-Elimination$, présentée en algorithme 19, permet de déterminer quel sous-ensemble de noeuds parmi le voisinage du pivot est à prendre en considération afin d'éviter de rater l'énumération d'une quasi-clique maximale.

Pour cela on commence par trier l'ensemble des candidats éliminés $u \in CandidatsElimines$ par la taille de leur voisinage avec la clique $|\Gamma(u) \cap C|$ *(ligne 2)*. L'idée étant d'éliminer en premier ceux apportant les meilleurs valeurs λ et γ et qui se retrouvent probablement dans plusieurs quasi-cliques maximales.

Ensuite on ajoute le noeud ayant le meilleur degré à la liste des noeuds préservés *Ret* et on l'enlève du sous-graphe induit *(lignes 6 à 12)*. Cette action se fait sous contrainte : elle est réalisée tant que les noeuds éliminés forment, avec la quasi-clique courante, un sous-graphe induit dont la taille est plus grande que ce qui est permis par la contrainte de voisinage *(lignes 3 et 9)* ou par la contrainte de densité *(lignes 4 et 10)*.

Finalement on retourne l'ensemble des noeuds préservés *Ret (ligne 13)*.

2.3.5 Exactitude de l'algorithme d'énumération de quasi-cliques maximales

Dans cette section nous démontrerons que notre méthode permet d'énumérer l'ensemble des quasi-cliques maximales sans doublons en respectant les contraintes imposées par les paramètres λ_{min} et γ_{min}. Pour cela nous considérons une étape e comme étant une situation fixe avec une quasi-clique courante C. Nous montrerons que, parce que C est construite par succession d'ajouts de noeuds (un par un), notre méthode liste l'ensemble des λ-γ-cliques maximales satisfaisant les contraintes.

Pour commencer, nous montrerons que pour une étape $e + 1$, nous calculons, à l'étape e, l'ensemble des candidats voisins de la quasi-clique courante à nécessairement étudié pour éviter de rater l'énumération d'une quasi-clique.

Pour commencer, nous montrerons que pour réaliser une étape $e + 1$, il est nécessaire de déterminer, à l'étape e, l'ensemble des candidats voisins de la quasi-clique courante. Ces candidats devant nécessairement être étudiés pour éviter de rater l'énumération d'une quasi-clique.

Algorithme 19 *PreservationTropElimination*(C, $CandidatsElimines$, λ_{min}, γ_{min})

1: $Ret \leftarrow \emptyset$
2: trier l'ensemble des $u \in CandidatsElimines$ par la valeur $|\Gamma(u) \cap C|$
3: $m \leftarrow \lfloor \Gamma_{min}(C \cup (CandidatsElimines - Ret)) \cdot \lambda_{min}^{-1} \rfloor + 1$
4: $n \leftarrow \left\lfloor \frac{1+\sqrt{1+4\cdot(2\cdot e\cdot\gamma_{min}^{-1})}}{2} \right\rfloor$
5: $i \leftarrow 1$
6: **tant que** $|C \cup (CandidatsElimines - Ret)| > min(m,n)$ **ET** $i \leq |CandidatsElimines|$ **faire**
7: $u_i \leftarrow$ le *ième* élément de l'ensemble $CandidatsElimines$
8: $Ret \leftarrow Ret + u_i$
9: $m \leftarrow \lfloor \Gamma_{min}(C \cup (CandidatsElimines - Ret)) \cdot \lambda_{min}^{-1} \rfloor + 1$
10: $n \leftarrow \left\lfloor \frac{1+\sqrt{1+4\cdot(2\cdot e\cdot\gamma_{min}^{-1})}}{2} \right\rfloor$
11: $i \leftarrow i + 1$
12: **fin tant que**
13: retourner Ret

Lemme 2.17. La fonction *SelectionCandFinis* liste l'ensemble des candidats à garder pour l'étape suivante $E + 1$ lorsque l'on vient d'ajouter un noeud à la clique courante C à l'étape e

Preuve. Soit :
- C_G l'ensemble de toutes les quasi-cliques maximales de G respectant les contraintes
- C la quasi-clique courante à l'étape e
- $Candidat$ l'ensemble des candidats pour accroître la quasi-clique C, $Candidat = \Gamma(C)$
- u un noeud candidat qui ne serait pas conservé par la fonction *PreservationVoisinsPivot* et où :
  - $u \in \Gamma(p)$
  - $(C + u) \cup S \in C_G$

Soit $S = \emptyset$, ce qui implique que $C + u$ satisfait les contraintes λ_{min} et γ_{min}, ce qui est en contradiction avec les lignes 6 et 7 de l'algorithme 16.

Soit $S \neq \emptyset$,

Ainsi soit $S \subseteq Candidats$, ce qui est pris en compte par la sous-fonction *AjoutAutresCandidats* présentée en algorithme 17, qui assure de tester toutes les combinaisons possibles pour chacun des candidats.

Sinon cela implique $S \nsubseteq Candidats$, on définit alors $T = S - Candidats$, ce qui implique $\overline{T} = S \cap Candidats$. Dans ce cas,

Si $\Gamma(T) \cap \overline{T} = \emptyset$, cela implique une quasi-clique non connexe. Cette situation n'est considérée dans notre cas.

Sinon cela implique $\Gamma(T) \cap \overline{T} \neq \emptyset$.

Si $(C + u) \cup \overline{T}$ satisfait les contraintes, cette situation est prise en compte par la sous-fonction présentée en algorithme 17,

Sinon cela implique de prendre en considération un ensemble de noeuds qui n'est pas voisin de la quasi-clique courante C, ce cas n'étant pas traité dans le cadre de notre proposition.

□

Ensuite nous montrerons qu'à une étape e l'ensemble des noeuds candidats à nécessairement testé le sont effectivement. Cette situation est assurée car nous évitons d'éliminer des candidats dans le voisinage du pivot qui permettent l'énumération de quasi-cliques qui ne seraient pas trouvées parce qu'elles ne contiennent pas le pivot.

Lemme 2.18. La fonction *PreservationVoisinsPivot* liste l'ensemble des voisins du pivot à garder à l'étape e pour accroître la quasi-clique courante.

Preuve. Soit :
- C_G l'ensemble de toutes les quasi-cliques maximales de G respectant les contraintes
- C la quasi-clique courante à l'étape e
- p le pivot sélectionné qui maximise la taille de l'ensemble $|Cand \cap \Gamma(p)|$

— $Candidat$ l'ensemble des candidats valables pour accroître la quasi-clique C
— u un noeud candidat qui ne serait pas conservé par la fonction $PreservationVoisinsPivot$ et où :
  — $u \in \Gamma(p)$
  — $(C + u) \cup S \in C_G$, $p \notin S$

Soit $S = \emptyset$, ce qui implique que $C+u$ satisfait les contraintes mais pas $C+u+p$, ce qui est en contradiction avec le test conditionnel d'ajout direct par l'algorithme 18.

Soit $S - \Gamma(p) = \emptyset$, ce qui implique que l'on élimine trop de noeuds, ce qui est en contradiction avec la préservation d'un minimum de candidats par l'algorithme 19, qui assure de conserver suffisamment de noeuds pour trouver toutes les quasi-cliques maximales.

Soit $S - \Gamma(p) \neq \emptyset$, ce qui implique qu'un ensemble $T \subseteq S - \Gamma(p)$ est nécessaire.

Ainsi si $T \cap Candidat \neq \emptyset$, cela implique qu'on passe par un noeud $v \in T$ pour découvrir la quasi-clique maximale, et qu'on n'a pas de raison de conserver le noeud u à l'étape e.

Sinon si $T \cap Candidat = \emptyset$ et que l'on considère $v \in T$:

Soit $T \cap \Gamma(u) \neq \emptyset$, cela implique qu'on a un noeud candidat v dans le voisinage de u qui n'est pas dans le voisinage du pivot p, ce qui est pris en compte par l'algorithme 19.

Soit $T \cap \Gamma(u) = \emptyset$, ce qui implique que la quasi-clique n'est pas connexe, ce qui n'est pas considéré dans notre cas.

□

Ensuite nous montrerons que la quasi-clique courante s'accroissant noeud par noeud, il n'est possible pas de rater l'énumération d'une quasi-clique. En effet, cette situation ne pourrait arriver que si l'on ne prendrait pas en considération un noeud de la quasi-clique à une étape e.

Théorème 2.19. *Pour un graphe G, la fonction $\lambda\gamma Tomita$ énumère l'ensemble des quasi-cliques maximales et sans doublons satisfaisant les contraintes λ_{min} et γ_{min} de G.*

Preuve. Soit :
— C_G l'ensemble de toutes les quasi-cliques maximales de G respectant les contraintes
— $C \in C_G$,$C = \{u_1,...,u_n\}$, une quasi-clique qui n'est pas énumérée par la fonction $\lambda\gamma Tomita$

Étant donné que $\lambda\gamma Tomita$ accroît la quasi-clique noeud par noeud, si C n'est pas trouvée par $\lambda\gamma Tomita$ cela implique $\exists u_e \in C$ qui n'est pas conservé à l'étape e :

Ainsi,

Soit u_e n'est pas conservé lors du calcul des candidats à l'étape $e - 1$, ce qui entre en contradiction avec le lemme 2.17.

Soit u_e n'est pas évalué à l'étape e, car, étant voisin du pivot, il a été éliminé sans être conservé, ce qui entre en contradiction avec le lemme 2.18.

□

Finalement nous montrerons que l'ensemble des quasi-clique trouvées sont maximales,

Proposition 2.20. Pour un graphe G, la fonction $\lambda\gamma Tomita$ énumère uniquement des quasi-cliques maximales.

Preuve. Soit :
— C_G l'ensemble de toutes les quasi-cliques maximales de G respectant les contraintes
— $C \in C_G$ une quasi-clique qui est énumérée par la fonction $\lambda\gamma Tomita$ mais qui n'est pas maximale

Si C n'est pas maximale, cela implique $\exists S$ tel que $C \cup S$ satisfait les contraintes λ_{min} et γ_{min}.

Ainsi soit $S \cap \Gamma(C) = \emptyset$, ce qui implique que la quasi-clique $C \cup S$ n'est pas connexe, ce qui n'est pas pris en considération dans notre cas.

Sinon cela implique qu'il existe un sous-ensemble $T = S \cap \Gamma(C)$ et

Soit $C \cup T$ satisfait les contraintes et n'est donc pas conservé ce qui entre en contradiction avec le théorème 2.19.

Soit $C \cup T$ ne satisfait pas les contraintes et cela implique de prendre en considération des noeuds n'appartenant pas au voisinage de la clique courante, situation qui n'est pas prise en considération dans notre cas.

□

Nous montrerons également que l'ensemble des quasi-clique trouvées ne comporte pas de doublons,

Proposition 2.21. Pour un graphe G, la fonction $\lambda\gamma Tomita$ énumère uniquement des quasi-cliques maximales sans doublons.

Preuve. Soit :
— C_G l'ensemble de toutes les quasi-cliques maximales de G respectant les contraintes
— C_1,$C_2 \in C_G$ deux quasi-cliques maximales énumérées par $PEQCO$ tel que $C_1 \equiv C_2$

Si $C_1 \equiv C_2$, cela implique que C_1 et C_2 ont été construites à partir du même noeud initial u, sinon il y a contradiction avec la répartition initiale des candidats entre les ensembles $Cand$ et $Fini$.

Ainsi si $C_1 \equiv C_2$ et C_1 et C_2 sont construites à partir du même noeud initial u, cela implique :
— $C_1 = \{S,u,T\}$
— et $C_2 = \{S,x,X,v,Y,u,Z\}$ construite après C_1 où :
  — $u \in Fini$ pour $C = \{S,x\}$
  — $u \notin Fini$,$u \notin Cand$ pour $C = \{S,x,X,v\}$

Ainsi, cela implique que $\exists w \in Y$ qui permet d'ajouter u à la liste des $Cand$ et donc que $u \in \Gamma(w)$. Or, si w est sélectionné, cela implique qu'il appartient à l'ensemble des $Cand$. S'il appartient à l'ensemble des $Cand$ et qu'il est voisin de u cela implique que u est également conservé à l'aide du noeud w par la sous-fonction $AjoutAutresCandidats$ présentée en algorithme 17. Si u est conservé, il y a contradiction avec le fait que $\exists v \Rightarrow u \notin Fini$.

□

En conclusion, l'algorithme $PEQCO$ énumère l'ensemble des quasi-cliques maximales sans doublons.

2.3.6 Élagages et heuristique pour l'énumération de quasi-cliques maximales

Dans cette section nous proposerons différentes méthodes d'élagages afin d'accélérer la vitesse d'exécution de notre algorithme $PEQCO$. En effet, de par la complexité du modèle de λ-γ-cliques, il convient de résoudre un problème NP-Complet pour chaque noeud $u \in \Gamma(C)$ après l'ajout d'un noeud à la quasi-clique courante. Ainsi, à moins que $P = NP$, il n'existe pas d'algorithme polynomial permettant de résoudre ce problème.

Pour améliorer le temps d'exécution nous proposons d'effectuer certaines coupes dans l'arbre d'exploration de la construction des quasi-cliques, notamment par l'utilisation d'algorithmes polynomiaux. Les conséquences étant qu'en fonction des élagages utilisés on peut se retrouver au final avec des quasi-cliques non-maximales, avec des doublons ou bien avec des quasi-cliques ne respectant pas les contraintes λ_{min} ou γ_{min}. Ainsi il est nécessaire de proposer des opérations de calculs supplémentaires (après l'énumération) pour corriger certaines de ces erreurs. Ces opérations de calculs étant au pire d'ordre $O(|C_G|^2)$ [3], elles restent dans un temps polynomial et sont donc négligeables par rapport au gain de temps espéré par l'utilisation des heuristiques.

La combinatoire des autres candidats et l'ordonnancement

Afin d'éviter de tester l'ensemble des combinaisons nous testons si un noeud u satisfait les contraintes par ajout direct ou par ajout successif de noeuds v_i présents dans l'ensemble des candidats, $v \in Candidats$. Pour maximiser le nombre de noeuds conservés nous trions les candidats par la taille de l'intersection entre leur voisinage et la quasi-clique courante, $|Gamma(v) \cap C|$.

La conséquence est que l'on peut éliminer des candidats qui sont valides. Un tel cas est présenté en figure 2.26. Si l'on souhaite énumérer l'ensemble des λ-γ-cliques, avec $\lambda_{min} = \frac{2}{5}$ et $\gamma_{min} = \frac{9}{15}$, on est censé découvrir la quasi-clique $\{0,1,2,3,5,6\}$ à partir du noeud initial 0.

Cependant, quand on cherche à sélectionner les candidats valables pour la quasi-clique courante $C = \{0,1,2,3\}$, on ne trouve que le noeud 4 comme candidat valable. Les noeuds 5 ou 6 n'étant pas conservés pour deux raisons : ils ont uniquement besoin l'un de l'autre pour former une quasi-clique valide et l'on teste systématiquement le noeud 4 avant les autres car il possède plus d'arêtes communes avec C.

3. Pour trouver des quasi-cliques non-maximales ou des doublons parmi toutes les cliques retournées on peut utiliser une comparaison une à une de chacune des quasi-cliques.

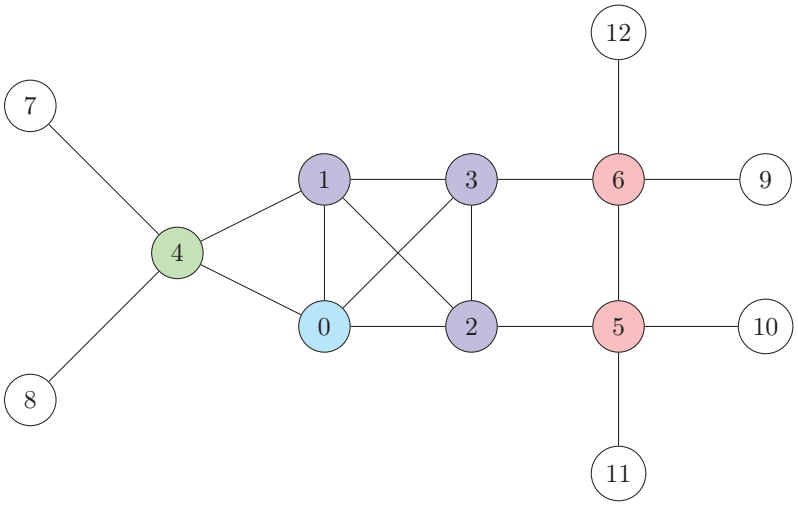


FIGURE 2.26 – Élimination de candidats valables dû à l'ordre d'exploration des autres candidats pour l'énumération de λ-γ-cliques, avec $\lambda_{min} = \frac{2}{5}$ et $\gamma_{min} = \frac{9}{15}$.

Élagages supplémentaires

Nous proposons de traiter différemment les potentiels candidats. Pour éviter d'avoir à en ajouter un trop grand nombre, nous effectuons certains élagages :

- on ne teste en candidats sélectionnés pour la prochaine étape que :
  - les voisins du dernier noeud ajouté
  - les candidats sélectionnés à l'étape précédente
- pour chacun des candidats u on ne prend en considération que les candidats voisins $v \in \Gamma(u) \cap Candidats$ pour tenter de satisfaire les contraintes
- on sépare la satisfaction des contraintes :
  1. on conserve d'abord les noeuds satisfaisants la contrainte de densité γ
  2. puis on teste parmi ceux conservés les noeuds qui satisfont la contrainte de voisinage λ
- on évalue la satisfaction des contraintes que localement. Un noeud est conservé pour les étapes suivantes lorsqu'il satisfait les contraintes à court terme (par ajout direct) ou lorsqu'il semble satisfaire les contraintes à l'aide des candidats dans son voisinage

Ces élagages peuvent entraîner la non-découverte ou la non-identification de quasi-cliques maximales.

La figure 2.27 présente un exemple. Si l'on cherche à énumérer les λ-γ-cliques, avec comme contraintes $\lambda_{min} = \frac{2}{5}$ et $\gamma_{min} = \frac{7}{15}$ à partir du noeud initial 0, en ne cherchant des nouveaux candidats que dans le voisinage du dernier noeud ajouté, on peut ne pas détecter certaines des quasi-cliques maximales.

En effet, en accord avec les contraintes, on est censé trouvé une unique quasi-clique {0,1,2,3,4,5}. Cependant, étant donné que l'heuristique ne traite que les candidats déjà présents dans l'ensemble et ceux qui sont voisins du dernier noeud ajouté, on ne détectera pas cette quasi-clique qui est maximale mais seulement des quasi-cliques qui sont inclus dans celle-ci :

Si l'on commence par ajouter le noeud 1 à la quasi-clique courante *(b)*, on détecte le noeud 3 comme candidat potentiel (son ajout direct permet de satisfaire les contraintes). Ensuite lorsque l'on ajoute le noeud 2 *(c)* on considère comme candidat possible le noeud 3, déjà présent dans la liste des candidats, et le noeud 4, voisin du noeud 2. Seul le noeud 3 est conservé car il permet de satisfaire la contrainte de voisinage. Le problème arrive ensuite lorsque l'on ajoute le noeud 3 : le noeud 4 n'étant pas voisin du noeud 3 et n'ayant pas été sélectionné à l'étape précédente comme noeud candidat, il n'est pas pris en considération comme candidat potentiel. Or, le fait de ne pas le prendre en considération ne permet pas de sélectionner le noeud 5, voisin du noeud 3 comme candidat. Ainsi, comme le noeud 3 n'est pas sélectionné comme candidat, la quasi-clique courante est considérée comme maximale *(d)*.

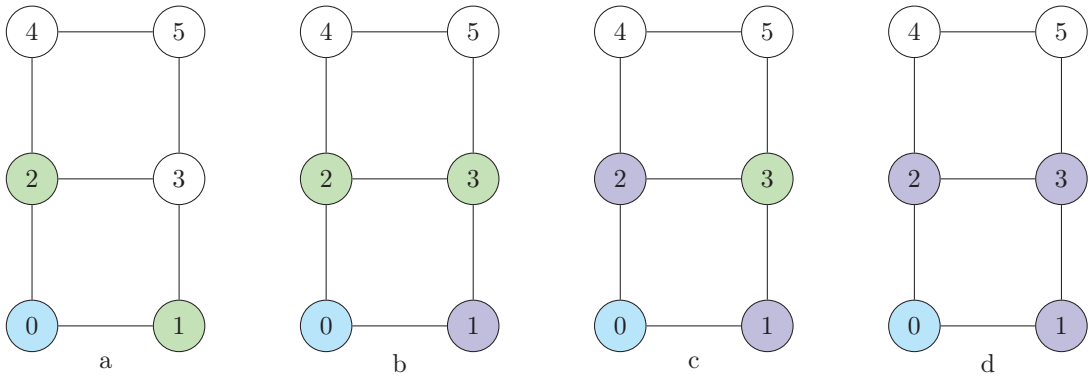


FIGURE 2.27 – Exploration d'un graphe avec modification de la sélection des candidats à partir du noeud 0 lors de l'énumération de λ-γ-clique, avec $\lambda_{min} = \frac{2}{5}$ et $\gamma_{min} = \frac{7}{15}$.

Un autre exemple est montré en figure 2.28. Si l'on cherche à énumérer les λ-γ-cliques, avec comme contraintes $\lambda_{min} = \frac{3}{5}$ et $\gamma_{min} = \frac{3}{5}$ à partir du noeud initial 0, la séparation et l'étude locale de la satisfaction des contraintes entraîne la non-découverte d'une quasi-clique maximale composée des noeuds {0,3,5,6}.

En effet, en accord avec les contraintes, lorsque la quasi-clique est composée des noeuds {0,3,5,6}, les noeuds 1, 2 et 4 seront conservés comme candidats. Ils satisfont en effet la contrainte γ par ajout direct, étant connectés à 2 noeuds de la quasi-clique courante. Ensuite, selon l'heuristique ils satisfont la contrainte λ. Cette erreur s'explique par le fait que la quasi-clique maximale espérée à partir de la quasi-clique courante et des candidats est de taille 6. Dans ce cas de figure, en accord avec la contrainte $\lambda_{min} = \frac{3}{5}$, les candidats doivent avoir 3 voisins parmi les ensembles de la quasi-clique courante et des candidats. Ceci étant bien le cas pour chacun des noeuds, ils sont conservés.

Finalement, lorsque les candidats sont ajoutés individuellement à la quasi-clique courante, plus aucun noeud n'est conservé comme candidat et les quasi-cliques courantes ne satisfaisant pas les contraintes, elles ne sont pas enregistrées comme quasi-cliques maximales. Cette situation est évitée par l'algorithme car il prend en compte les ajouts individuels successifs des noeuds. Dans ce cas de figure, les contraintes ne peuvent être satisfaites, il identifiera alors la quasi-clique courante comme une quasi-clique maximale.

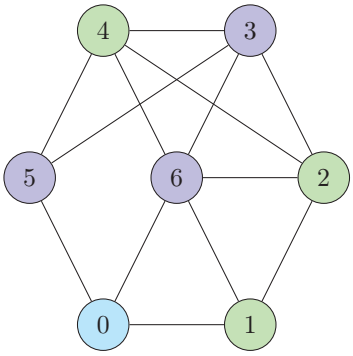


FIGURE 2.28 – Exploration d'un graphe avec modification de la sélection des candidats à partir du noeud 0 lors de l'énumération de λ-γ-clique, avec $\lambda_{min} = \frac{3}{5}$ et $\gamma_{min} = \frac{3}{5}$.

Le fait de seulement prendre en compte le voisinage entraîne également la non énumération de certaines quasi-cliques maximales comme cela avait été présenté en figure 2.19.

L'utilisation du diamètre

Nous avons également ajouté un élagage supplémentaire via l'utilisation d'un diamètre dans la quasi-clique. En sélectionnant uniquement les quasi-cliques de diamètre maximal de 2 on peut diminuer le nombre de candidats étudiés. Pour cela, on ne conserve à chaque étape que les noeuds v après l'ajout d'un noeud u à la quasi-clique courante. Ceux-ci doivent :

— être voisins du noeud u, $v \in \Gamma(u)$

— avoir au moins un voisin commun avec le noeud u, $\Gamma(v) \cap \Gamma(u) \neq \emptyset$

Cet élagage n'a aucun impact sur la qualité dès lors que la contrainte de voisinage est définie comme devant être supérieure à la moitié des noeuds de la quasi-clique.

Proposition 2.22. Une contrainte de voisinage $\lambda_{min} \geq \frac{1}{2}$ implique un diamètre maximal de 2.

Preuve. Soit une quasi-clique courant C tel que $\lambda_C \geq \frac{1}{2}$,

Si le diamètre de C est de 3 cela implique :

$$\exists u,v \in C \text{ tel que}$$
$$u \notin \Gamma(v) \textbf{ ET } \Gamma(u) \cap \Gamma(v) = \emptyset$$

Or :

$$|\Gamma(u)| \geq \left\lceil \frac{|C|}{2} \right\rceil$$
$$|\Gamma(v)| \geq \left\lceil \frac{|C|}{2} \right\rceil$$
$$|\Gamma(u)| + |\Gamma(v)| \geq |C| \text{ car les voisinages sont disjoints}$$

Ceci implique que ni u ni v n'appartiennent à la quasi-clique courante, ce qui est faux. □

De la gestion du pivot

Afin d'améliorer nos temps de calculs, les élagages appliqués à la sélection de nouveaux candidats le sont également pour la sélection de voisins du pivot.

2.4 Expérimentations

Afin de tester les différents algorithmes et paramètres nous avons mis en place différentes expérimentations permettant d'évaluer :

— la qualité des différents algorithmes

— la rapidité des différents algorithmes

— l'impact des valeurs des paramètres λ et γ

Graphes de test

Nous avons généré des graphes à l'aide de la bibliothèque NetworkX [52].

Les graphes sont de différents types :

— Aléatoire [30, 38]

— Régulier [104, 65]

— Power-Law [56]

— Petits mondes [115, 86]

Nom	**|V|**	**|E|**	$deg_{moy}(G)$	$\Delta(G)$	$\delta(G)$
erdos_20_0.2	20	32	3,20	6	1
erdos_20_0.4	20	77	7,70	11	5
erdos_20_0.6	20	105	10,50	13	7
erdos_20_0.8	20	155	15,50	19	13
erdos_50_0.2	50	210	8,40	13	4
erdos_50_0.4	50	482	19,28	28	14
erdos_50_0.6	50	707	28,28	36	22
erdos_50_0.8	50	975	39,00	46	33
erdos_100_0.2	100	961	19,22	27	9
erdos_100_0.4	100	1974	39,48	53	29
erdos_100_0.6	100	2936	58,72	74	41
erdos_100_0.8	100	3973	79,46	88	69

Tableau 2.2 – Graphes de données aléatoires pour les tests sur l'énumération de quasi-cliques.

Les graphes aléatoires sont présentés dans le tableau 2.2. Nous avons généré des graphes avec 20, 50 ou 100 noeuds et des probabilité d'existence d'une arête entre deux noeuds de 20, 40, 60 ou 80%.

Les graphes réguliers sont présentés dans le tableau 2.3. Nous avons généré des graphes avec 20, 50 ou 100 noeuds et un nombre d'arêtes par noeud compris entre 3 et 10.

Nom	**|V|**	**|E|**	$deg_{moy}(G)$	$\Delta(G)$	$\delta(G)$
regular_20_3	20	30	3,00	3	3
regular_20_4	20	40	4,00	4	4
regular_20_5	20	50	5,00	5	5
regular_20_6	20	60	6,00	6	6
regular_20_7	20	70	7,00	7	7
regular_20_8	20	80	8,00	8	8
regular_20_9	20	90	9,00	9	9
regular_20_10	20	100	10,00	10	10
regular_50_3	50	75	3,00	3	3
regular_50_4	50	100	4,00	4	4
regular_50_5	50	125	5,00	5	5
regular_50_6	50	150	6,00	6	6
regular_50_7	50	175	7,00	7	7
regular_50_8	50	200	8,00	8	8
regular_50_9	50	225	9,00	9	9
regular_50_10	50	250	10,00	10	10
regular_100_3	100	150	3,00	3	3
regular_100_4	100	200	4,00	4	4
regular_100_5	100	250	5,00	5	5
regular_100_6	100	300	6,00	6	6
regular_100_7	100	350	7,00	7	7
regular_100_8	100	400	8,00	8	8
regular_100_9	100	450	9,00	9	9
regular_100_10	100	500	10,00	10	10

Tableau 2.3 – Graphes de données réguliers pour les tests sur l'énumération de quasi-cliques.

Les graphes Power-Law sont présentés dans les tableaux 2.4, 2.5 et 2.6. Nous avons généré des graphes avec 20, 50 ou 100 noeuds ; des arêtes ajoutées aléatoirement (entre 3 et 10 par noeud) ; une probabilité de créer un triangle après l'ajout aléatoire d'une arête égale à 20, 40, 60 ou 80%.

Les graphes petits mondes sont présentés dans les tableaux 2.7, 2.8 et 2.9. Nous avons généré des graphes avec 20, 50 ou 100 noeuds ; des arêtes ajoutées aléatoirement (entre 3 et 10 par noeud) ; une probabilité de créer un triangle après l'ajout aléatoire d'une arête égale à 20, 40, 60 ou 80%.

Nom	**|V|**	**|E|**	$deg_{moy}(G)$	$\Delta(G)$	$\delta(G)$
power_law_20_3_0.2	20	51	5,10	11	3
power_law_20_3_0.4	20	50	5,00	14	1
power_law_20_3_0.6	20	51	5,10	12	3
power_law_20_3_0.8	20	48	4,80	14	2
power_law_20_4_0.2	20	62	6,20	16	4
power_law_20_4_0.4	20	63	6,30	15	4
power_law_20_4_0.6	20	64	6,40	14	3
power_law_20_4_0.8	20	64	6,40	13	4
power_law_20_5_0.2	20	75	7,50	14	5
power_law_20_5_0.4	20	73	7,30	15	3
power_law_20_5_0.6	20	74	7,40	17	4
power_law_20_5_0.8	20	73	7,30	15	3
power_law_20_6_0.2	20	80	8,00	17	3
power_law_20_6_0.4	20	82	8,20	18	2
power_law_20_6_0.6	20	81	8,10	16	4
power_law_20_6_0.8	20	82	8,20	17	3
power_law_20_7_0.2	20	88	8,80	17	5
power_law_20_7_0.4	20	88	8,80	18	4
power_law_20_7_0.6	20	88	8,80	18	1
power_law_20_7_0.8	20	90	9,00	19	3
power_law_20_8_0.2	20	92	9,20	18	4
power_law_20_8_0.4	20	90	9,00	18	3
power_law_20_8_0.6	20	94	9,40	18	3
power_law_20_8_0.8	20	95	9,50	18	3
power_law_20_9_0.2	20	97	9,70	19	5
power_law_20_9_0.4	20	96	9,60	18	4
power_law_20_9_0.6	20	97	9,70	18	1
power_law_20_9_0.8	20	98	9,80	18	3
power_law_20_10_0.2	20	93	9,30	18	4
power_law_20_10_0.4	20	94	9,40	18	3
power_law_20_10_0.6	20	99	9,90	19	3
power_law_20_10_0.8	20	100	10,00	19	4

Tableau 2.4 – Graphes de données Power-Law de taille 20 pour les tests sur l'énumération de quasi-cliques.

Nom	**|V|**	**|E|**	$deg_{moy}(G)$	$\Delta(G)$	$\delta(G)$
power_law_50_3_0.2	50	141	5,64	20	3
power_law_50_3_0.4	50	141	5,64	18	3
power_law_50_3_0.6	50	141	5,64	24	3
power_law_50_3_0.8	50	140	5,60	24	2
power_law_50_4_0.2	50	182	7,28	22	3
power_law_50_4_0.4	50	182	7,28	21	4
power_law_50_4_0.6	50	183	7,32	22	3
power_law_50_4_0.8	50	184	7,36	30	2
power_law_50_5_0.2	50	220	8,80	21	5
power_law_50_5_0.4	50	222	8,88	30	5
power_law_50_5_0.6	50	220	8,80	32	5
power_law_50_5_0.8	50	220	8,80	26	3
power_law_50_6_0.2	50	258	10,32	24	5
power_law_50_6_0.4	50	254	10,16	29	5
power_law_50_6_0.6	50	257	10,28	38	5
power_law_50_6_0.8	50	260	10,40	34	5
power_law_50_7_0.2	50	294	11,76	32	7
power_law_50_7_0.4	50	289	11,56	35	6
power_law_50_7_0.6	50	294	11,76	34	6
power_law_50_7_0.8	50	296	11,84	36	5
power_law_50_8_0.2	50	332	13,28	36	7
power_law_50_8_0.4	50	327	13,08	36	7
power_law_50_8_0.6	50	328	13,12	38	7
power_law_50_8_0.8	50	330	13,20	35	2
power_law_50_9_0.2	50	360	14,40	38	8
power_law_50_9_0.4	50	357	14,28	34	8
power_law_50_9_0.6	50	362	14,48	39	7
power_law_50_9_0.8	50	361	14,44	38	4
power_law_50_10_0.2	50	390	15,60	37	2
power_law_50_10_0.4	50	383	15,32	40	8
power_law_50_10_0.6	50	389	15,56	38	7
power_law_50_10_0.8	50	394	15,76	39	5

Tableau 2.5 – Graphes de données Power-Law de taille 50 pour les tests sur l'énumération de quasi-cliques.

Nom	**|V|**	**|E|**	$deg_{moy}(G)$	$\Delta(G)$	$\delta(G)$
power_law_100_3_0.2	100	291	5,82	33	3
power_law_100_3_0.4	100	288	5,76	32	3
power_law_100_3_0.6	100	290	5,80	32	3
power_law_100_3_0.8	100	290	5,80	27	2
power_law_100_4_0.2	100	383	7,66	29	4
power_law_100_4_0.4	100	383	7,66	39	4
power_law_100_4_0.6	100	383	7,66	33	4
power_law_100_4_0.8	100	382	7,64	32	4
power_law_100_5_0.2	100	470	9,40	38	3
power_law_100_5_0.4	100	472	9,44	44	5
power_law_100_5_0.6	100	474	9,48	38	2
power_law_100_5_0.8	100	473	9,46	49	5
power_law_100_6_0.2	100	559	11,18	44	5
power_law_100_6_0.4	100	557	11,14	53	5
power_law_100_6_0.6	100	551	11,02	53	5
power_law_100_6_0.8	100	555	11,10	55	5
power_law_100_7_0.2	100	640	12,80	48	6
power_law_100_7_0.4	100	639	12,78	49	6
power_law_100_7_0.6	100	642	12,84	50	6
power_law_100_7_0.8	100	646	12,92	53	7
power_law_100_8_0.2	100	725	14,50	52	8
power_law_100_8_0.4	100	722	14,44	52	7
power_law_100_8_0.6	100	724	14,48	58	7
power_law_100_8_0.8	100	726	14,52	68	7
power_law_100_9_0.2	100	808	16,16	59	8
power_law_100_9_0.4	100	800	16,00	53	8
power_law_100_9_0.6	100	803	16,06	53	6
power_law_100_9_0.8	100	806	16,12	62	8
power_law_100_10_0.2	100	881	17,62	55	9
power_law_100_10_0.4	100	883	17,66	60	10
power_law_100_10_0.6	100	884	17,68	57	9
power_law_100_10_0.8	100	887	17,74	65	8

Tableau 2.6 – Graphes de données Power-Law de taille 100 pour les tests sur l'énumération de quasi-cliques.

Nom	**|V|**	**|E|**	$deg_{moy}(G)$	$\Delta(G)$	$\delta(G)$
small_20_3_0.2	20	23	2,30	4	2
small_20_3_0.4	20	30	3,00	4	2
small_20_3_0.6	20	34	3,40	6	2
small_20_3_0.8	20	38	3,80	5	2
small_20_4_0.2	20	45	4,50	6	4
small_20_4_0.4	20	58	5,80	9	4
small_20_4_0.6	20	69	6,90	9	4
small_20_4_0.8	20	72	7,20	11	4
small_20_5_0.2	20	50	5,00	7	4
small_20_5_0.4	20	58	5,80	8	4
small_20_5_0.6	20	67	6,70	9	4
small_20_5_0.8	20	77	7,70	9	5
small_20_6_0.2	20	74	7,40	11	6
small_20_6_0.4	20	83	8,30	12	6
small_20_6_0.6	20	91	9,10	11	6
small_20_6_0.8	20	110	11,00	15	8
small_20_7_0.2	20	68	6,80	8	6
small_20_7_0.4	20	84	8,40	12	6
small_20_7_0.6	20	86	8,60	11	6
small_20_7_0.8	20	108	10,80	13	9
small_20_8_0.2	20	98	9,80	12	8
small_20_8_0.4	20	112	11,20	14	9
small_20_8_0.6	20	127	12,70	15	11
small_20_8_0.8	20	150	15,00	18	13
small_20_9_0.2	20	93	9,30	11	8
small_20_9_0.4	20	103	10,30	14	8
small_20_9_0.6	20	127	12,70	14	10
small_20_9_0.8	20	145	14,50	19	9
small_20_10_0.2	20	118	11,80	14	10
small_20_10_0.4	20	144	14,40	17	12
small_20_10_0.6	20	162	16,20	19	13
small_20_10_0.8	20	170	17,00	19	13

Tableau 2.7 – Graphes de données petits mondes de taille 20 pour les tests sur l'énumération de quasi-cliques.

Nom	**|V|**	**|E|**	$deg_{moy}(G)$	$\Delta(G)$	$\delta(G)$
small_50_3_0.2	50	63	2,52	4	2
small_50_3_0.4	50	72	2,88	5	2
small_50_3_0.6	50	79	3,16	6	2
small_50_3_0.8	50	89	3,56	6	2
small_50_4_0.2	50	124	4,96	7	4
small_50_4_0.4	50	147	5,88	8	4
small_50_4_0.6	50	156	6,24	10	4
small_50_4_0.8	50	185	7,40	10	4
small_50_5_0.2	50	117	4,68	7	4
small_50_5_0.4	50	135	5,40	8	4
small_50_5_0.6	50	154	6,16	10	4
small_50_5_0.8	50	179	7,16	10	4
small_50_6_0.2	50	180	7,20	10	6
small_50_6_0.4	50	206	8,24	10	6
small_50_6_0.6	50	240	9,60	15	6
small_50_6_0.8	50	262	10,48	14	6
small_50_7_0.2	50	178	7,12	10	6
small_50_7_0.4	50	212	8,48	11	6
small_50_7_0.6	50	239	9,56	14	6
small_50_7_0.8	50	272	10,88	15	6
small_50_8_0.2	50	232	9,28	12	8
small_50_8_0.4	50	268	10,72	15	8
small_50_8_0.6	50	316	12,64	17	8
small_50_8_0.8	50	356	14,24	18	9
small_50_9_0.2	50	243	9,72	13	8
small_50_9_0.4	50	286	11,44	15	8
small_50_9_0.6	50	329	13,16	17	10
small_50_9_0.8	50	358	14,32	19	10
small_50_10_0.2	50	285	11,40	14	10
small_50_10_0.4	50	349	13,96	17	11
small_50_10_0.6	50	407	16,28	21	10
small_50_10_0.8	50	450	18,00	24	12

Tableau 2.8 – Graphes de données petits mondes de taille 50 pour les tests sur l'énumération de quasi-cliques.

Nom	**|V|**	**|E|**	$deg_{moy}(G)$	$\Delta(G)$	$\delta(G)$
small_100_3_0.2	100	107	2,14	4	2
small_100_3_0.4	100	141	2,82	5	2
small_100_3_0.6	100	168	3,36	7	2
small_100_3_0.8	100	184	3,68	7	2
small_100_4_0.2	100	243	4,86	7	4
small_100_4_0.4	100	282	5,64	9	4
small_100_4_0.6	100	324	6,48	12	4
small_100_4_0.8	100	354	7,08	10	4
small_100_5_0.2	100	225	4,50	7	4
small_100_5_0.4	100	282	5,64	9	4
small_100_5_0.6	100	314	6,28	9	4
small_100_5_0.8	100	370	7,40	11	4
small_100_6_0.2	100	357	7,14	10	6
small_100_6_0.4	100	433	8,66	12	6
small_100_6_0.6	100	481	9,62	14	6
small_100_6_0.8	100	550	11,00	15	8
small_100_7_0.2	100	360	7,20	11	6
small_100_7_0.4	100	422	8,44	12	6
small_100_7_0.6	100	482	9,64	13	6
small_100_7_0.8	100	544	10,88	16	7
small_100_8_0.2	100	481	9,62	14	8
small_100_8_0.4	100	565	11,30	15	8
small_100_8_0.6	100	633	12,66	18	9
small_100_8_0.8	100	716	14,32	21	10
small_100_9_0.2	100	484	9,68	14	8
small_100_9_0.4	100	572	11,44	16	8
small_100_9_0.6	100	652	13,04	18	8
small_100_9_0.8	100	728	14,56	20	10
small_100_10_0.2	100	590	11,80	16	10
small_100_10_0.4	100	707	14,14	19	11
small_100_10_0.6	100	808	16,16	22	11
small_100_10_0.8	100	908	18,16	24	13

Tableau 2.9 – Graphes de données petits mondes de taille 100 pour les tests sur l'énumération de quasi-cliques.

2.4.1 Étude de l'influence du pivot sur l'algorithme

Le fait d'utiliser un pivot permet, pour l'énumération de cliques, d'élaguer des branches de l'arbre de recherche. Cependant, pour la recherche de quasi-clique, l'utilisation d'un pivot ne permet pas systématiquement d'éliminer l'ensemble de son voisinage. Potentiellement, la totalité de son voisinage doit même être conservée. Nous avons donc testé notre algorithme avec et sans utilisation du pivot afin de voir l'impact des calculs supplémentaires engendrés par la conservation des voisins du pivot. Pour chacun des graphes nous avons mesuré les temps effectifs des algorithmes :

— $PEQCO_p$, l'algorithme avec utilisation du pivot
— $PEQCO$, l'algorithme sans utilisation du pivot

Les algorithmes ont été implémentés en Scala avec la librairie Spark [102] et GraphX [117]. Ils ont été exécutés sur deux machines ayant des CPU Intel(R) Xeon(R) E3-1245 v5 avec 3.50GHz et 5 workers Spark avec chacun 16G de RAM.

Nous avons testé les deux algorithmes sur les graphes de taille 20. Pour chacun des graphes nous avons défini les valeurs des paramètres λ_{min} et γ_{min} sur la base de la taille de la plus grande clique du graphe $\omega(G)$:

$$\lambda_{min} = \gamma_{min} = \frac{\omega(G) - 1}{\omega(G) + 1}$$

Résultats

Les résultats des tests d'énumération de quasi-cliques sur des graphes aléatoires sont présentés dans le tableau 2.10 et la figure 2.29. Nous noterons que le temps d'exécution de $PEQCO$ sans le pivot est en moyenne inférieur à la moitié du temps d'exécution de $PEQCO_p$.

Nom	$\omega(G)$	$\|K\|$	λ/γ_{min}	$\|Q\|$	$PEQCO_p$ (s)	$PEQCO$ (s)
erdos_20_0.2	3	24	0,5	44	2,942	0,876
erdos_20_0.4	5	40	0,66	135	47,025	23,251
erdos_20_0.6	5	61	0,66	374	2 099	1 040
erdos_20_0.8	9	90	0,8	765	27 614	13 102

Tableau 2.10 – Temps d'exécution sur des graphes aléatoires pour l'énumération de quasi-cliques, avec et sans pivot.

Les résultats des tests d'énumération de quasi-cliques sur des graphes réguliers sont présentés dans le tableau 2.11 et la figure 2.30. Nous noterons que le temps d'exécution de $PEQCO$ sans le pivot est en moyenne inférieur à la moitié du temps d'exécution de $PEQCO_p$.

Nom	$\omega(G)$	$\|K\|$	λ/γ_{min}	$\|Q\|$	$PEQCO_p$ (s)	$PEQCO$ (s)
regular_20_3	3	28	0,5	49	2,357	0,788
regular_20_4	3	31	0,5	38	3,357	0,996
regular_20_5	4	27	0,6	35	3,545	1,034
regular_20_6	4	32	0,6	44	8,583	3,114
regular_20_7	4	41	0,6	81	60,209	27,472
regular_20_8	4	49	0,6	220	388,776	204,541
regular_20_9	5	59	0,66	187	495,017	252,677
regular_20_10	5	62	0,66	260	2 506	1 234

Tableau 2.11 – Temps d'exécution sur des graphes réguliers pour l'énumération de quasi-cliques, avec et sans pivot.

Les résultats des tests d'énumération de quasi-cliques sur des graphes Power-Law sont présentés dans le tableau 2.12 et la figure 2.31. Nous noterons que le temps d'exécution de $PEQCO$ sans le pivot est en moyenne inférieur à la moitié du temps d'exécution de $PEQCO_p$.

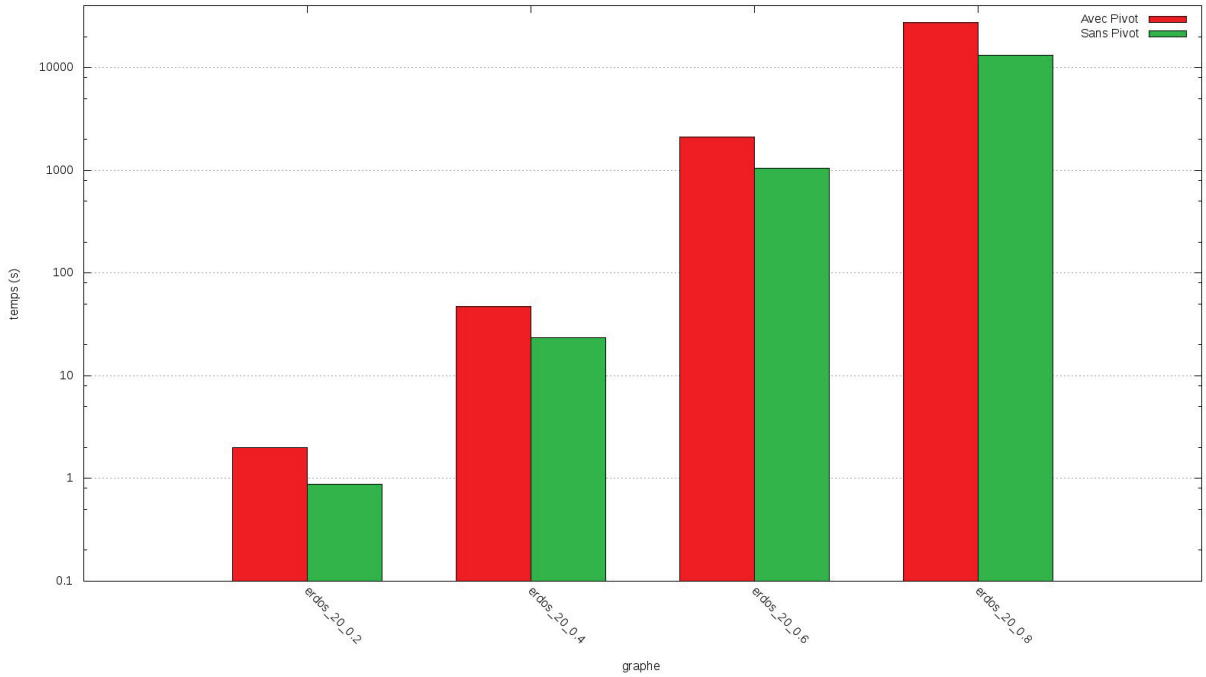


FIGURE 2.29 – Résultat temporels pour l'utilisation ou non d'un pivot sur des graphes aléatoires.

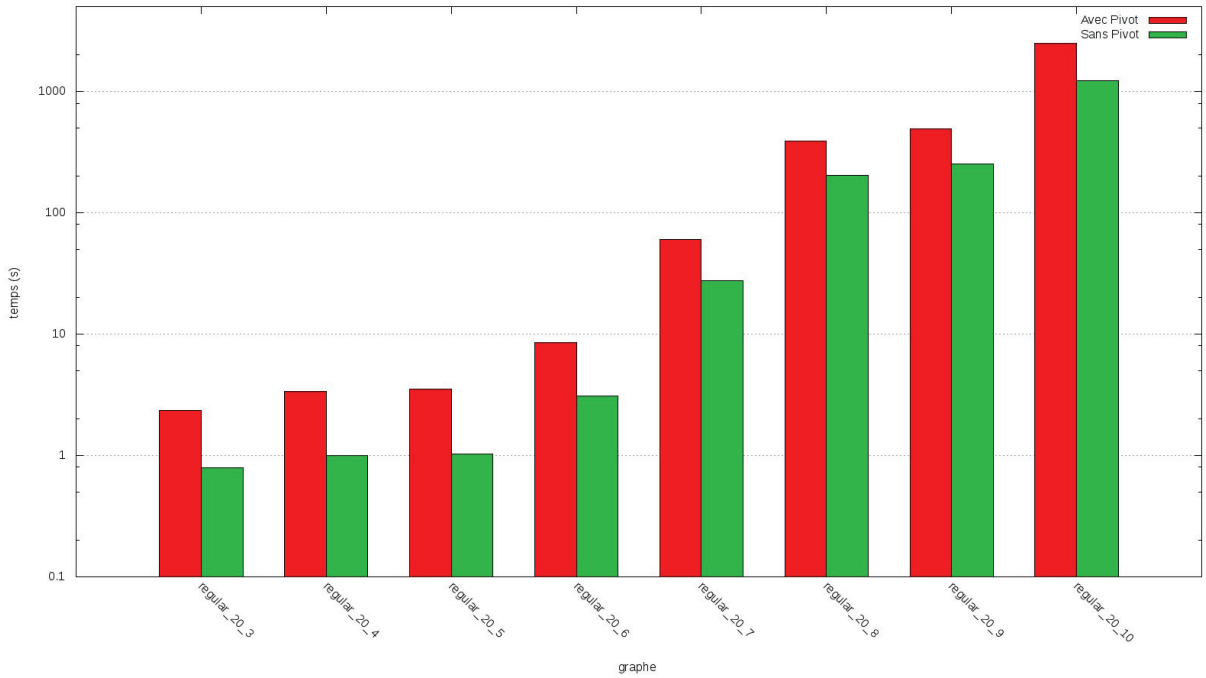


FIGURE 2.30 – Résultats temporels pour l'utilisation ou non d'un pivot sur des graphes réguliers.

Les résultats des tests d'énumération de quasi-cliques sur des graphes petits mondes sont présentés dans

Nom	$\omega(G)$	$\|K\|$	λ/γ_{min}	$\|Q\|$	$PEQCO_p$ (s)	$PEQCO$ (s)
power_law_20_3_0.2	4	23	0,6	39	4,287	1,475
power_law_20_3_0.4	4	25	0,6	46	4,47	1,608
power_law_20_3_0.6	4	24	0,6	32	4,404	1,498
power_law_20_3_0.8	4	20	0,6	26	4,181	1,274
power_law_20_4_0.2	4	32	0,6	77	18,198	8,646
power_law_20_4_0.4	5	27	0,66	81	11,891	4,918
power_law_20_4_0.6	5	24	0,66	61	9,37	4,038
power_law_20_4_0.8	5	22	0,66	61	12,005	6,122
power_law_20_5_0.2	5	37	0,66	107	27,975	15,535
power_law_20_5_0.4	5	30	0,66	79	24,405	11,412
power_law_20_5_0.6	6	25	0,71	55	23,627	10,638
power_law_20_5_0.8	5	23	0,66	81	31,58	12,557
power_law_20_6_0.2	5	38	0,66	136	110,169	56,847
power_law_20_6_0.4	5	33	0,66	156	123,026	60,692
power_law_20_6_0.6	6	36	0,71	81	39,997	25,972
power_law_20_6_0.8	7	23	0,75	65	43,184	22,929
power_law_20_7_0.2	6	42	0,71	114	100,324	48,837
power_law_20_7_0.4	5	49	0,66	229	274,503	126,1
power_law_20_7_0.6	6	31	0,71	76	98,379	56,341
power_law_20_7_0.8	7	19	0,75	72	56,533	35,4
power_law_20_8_0.2	6	31	0,71	102	117,238	59,421
power_law_20_8_0.4	7	31	0,75	89	112,329	58,973
power_law_20_8_0.6	6	38	0,71	115	176,826	87,389
power_law_20_8_0.8	7	29	0,75	94	106,836	55,233
power_law_20_9_0.2	6	43	0,71	148	412,443	202,436
power_law_20_9_0.4	7	27	0,75	91	193,241	88,992
power_law_20_9_0.6	6	37	0,71	170	209,142	87,074
power_law_20_9_0.8	8	22	0,77	58	169,888	78,293
power_law_20_10_0.2	6	50	0,71	193	172,061	73,111
power_law_20_10_0.4	6	35	0,71	151	160,124	73,617
power_law_20_10_0.6	7	35	0,75	107	127,237	73,428
power_law_20_10_0.8	7	30	0,75	110	110,506	44,922

Tableau 2.12 – Temps d'exécution sur des graphes Power Law pour l'énumération de quasi-cliques, avec et sans pivot.

le tableau 2.13 et la figure 2.32. Nous noterons que le temps d'exécution de $PEQCO$ sans le pivot est en moyenne inférieur à la moitié du temps d'exécution de $PEQCO_p$.

En conclusion, nous remarquerons que le gain de temps moyen dû à l'élimination de l'utilisation du pivot est d'environ 50% du temps initial de l'algorithme. Ainsi les gains de temps espérés par les élagages dus à l'utilisation du pivot sont contrebalancés par le temps pris par la sélection des voisins du pivot devant être conservés.

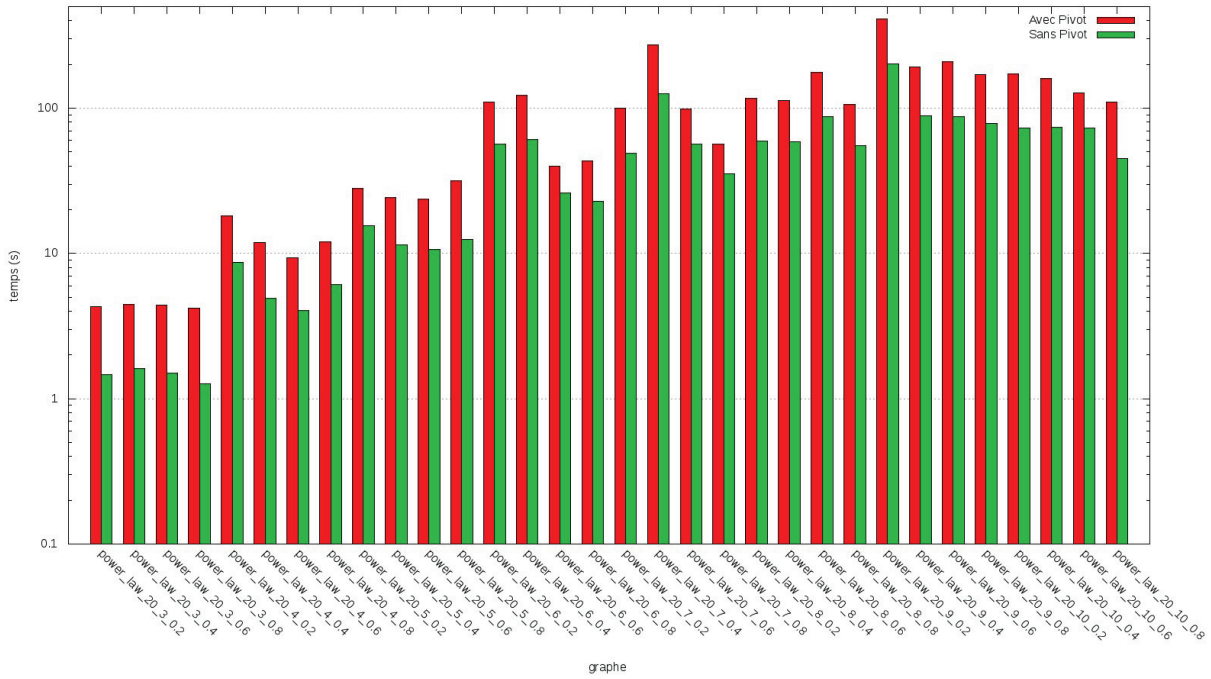


FIGURE 2.31 – Résultats temporels pour l'utilisation ou non d'un pivot sur des graphes Power Law.

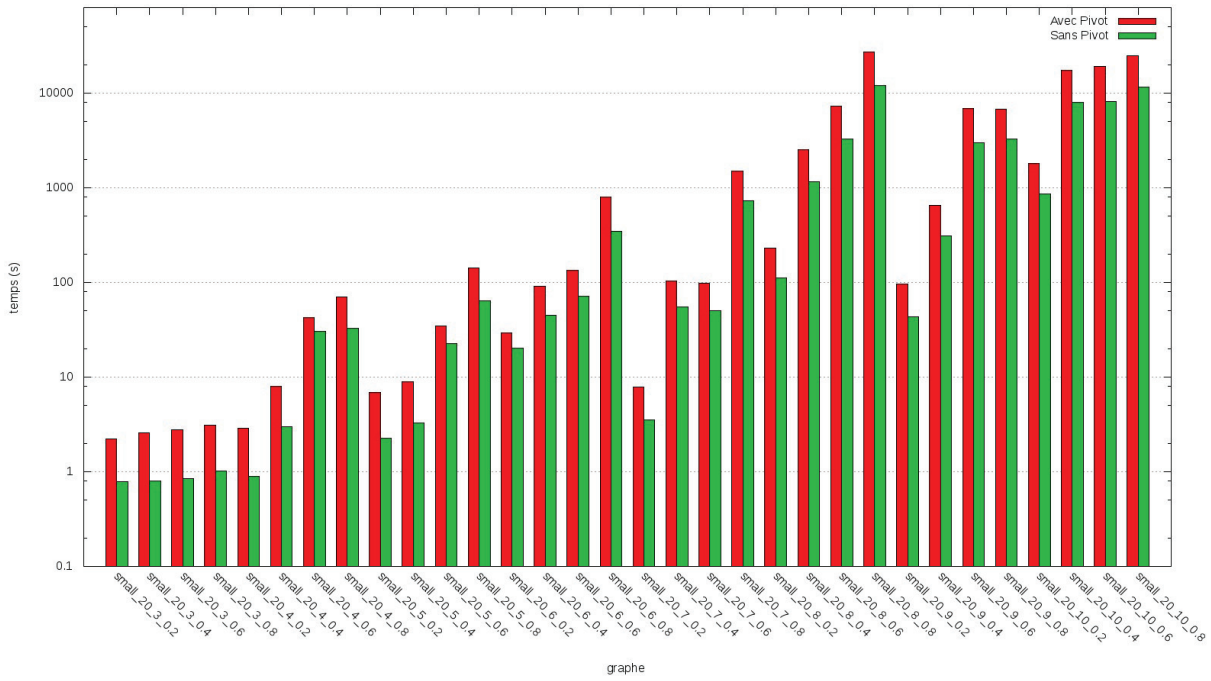


FIGURE 2.32 – Résultats temporels pour l'utilisation ou non d'un pivot sur des graphes petit-mondes.

Nom	$\omega(G)$	$\|K\|$	λ/γ_{min}	$\|Q\|$	$PEQCO_p$ (s)	$PEQCO$ (s)
small_20_3_0.2	2	23	0,33	30	2,221	0,788
small_20_3_0.4	3	28	0,5	38	2,57	0,803
small_20_3_0.6	3	30	0,5	34	2,764	0,842
small_20_3_0.8	3	31	0,5	46	3,104	1,017
small_20_4_0.2	4	23	0,6	18	2,887	0,888
small_20_4_0.4	4	31	0,6	37	8,048	2,989
small_20_4_0.6	4	37	0,6	90	42,272	30,344
small_20_4_0.8	4	31	0,6	124	70,341	32,869
small_20_5_0.2	3	28	0,5	68	6,946	2,249
small_20_5_0.4	4	32	0,6	33	8,912	3,273
small_20_5_0.6	4	39	0,6	87	34,731	22,567
small_20_5_0.8	4	44	0,6	187	141,405	64,211
small_20_6_0.2	5	31	0,66	60	29,235	20,119
small_20_6_0.4	5	28	0,66	107	90,856	45,213
small_20_6_0.6	6	31	0,71	82	133,59	71,475
small_20_6_0.8	7	48	0,75	189	795	349,733
small_20_7_0.2	5	24	0,66	54	7,784	3,565
small_20_7_0.4	5	30	0,66	96	102,945	55,542
small_20_7_0.6	5	33	0,66	120	97,57	50,083
small_20_7_0.8	6	54	0,71	158	1 511	729
small_20_8_0.2	6	33	0,71	76	231,38	111,835
small_20_8_0.4	6	66	0,71	207	2 532	1 152
small_20_8_0.6	7	75	0,75	346	7 291	3 308
small_20_8_0.8	9	81	0,8	476	27 476	11 984
small_20_9_0.2	6	30	0,71	74	95,583	43,027
small_20_9_0.4	6	39	0,71	109	654	308,8
small_20_9_0.6	7	67	0,75	382	6 851	3 004
small_20_9_0.8	10	52	0,81	108	6 762	3 275
small_20_10_0.2	7	39	0,75	184	1 808	857
small_20_10_0.4	8	110	0,77	643	17 314	7 972
small_20_10_0.6	12	53	0,84	129	19 120	8 206
small_20_10_0.8	13	36	0,85	27	24 682	11 501

Tableau 2.13 – Temps d'exécution sur des graphes petit-mondes pour l'énumération de quasi-cliques, avec et sans pivot.

2.4.2 Étude de l'influence du pivot sur l'heuristique

Nous avons également voulu tester l'influence du pivot sur l'heuristique. Dans l'heuristique les élagages amènent à ne pas considérer certaines branches de l'arbre d'exploration. Le fait d'utiliser un pivot permet également d'éviter certaines branches. Le calcul du pivot dans la quasi-clique subissant lui aussi des élagages nous avons souhaité connaître l'influence du pivot sur l'heuristique en termes de nombre de quasi-cliques énumérées et de temps de calcul. Nous avons donc testé deux versions de l'heuristique :

— $PEQCO_{h_p}$, l'heuristique avec utilisation du pivot
— $PEQCO_h$, l'heuristique sans utilisation du pivot

Les algorithmes ont été implémentés en Scala avec la librairie Spark [102] et GraphX [117]. Ils ont été exécutés sur deux machines ayant des CPU Intel(R) Xeon(R) E3-1245 v5 avec 3.50GHz et 5 workers Spark avec chacun 16G de RAM.

Nous avons testé les deux heuristiques sur des graphes de taille 20. Pour chacun des graphes nous avons défini les valeurs du paramètres λ_{min} et γ_{min} sur la base de la taille de la plus grande clique du graphe $\omega(G)$:

$$\lambda_{min} = \gamma_{min} = \frac{\omega(G) - 1}{\omega(G) + 1}$$

Résultats

Les résultats des tests d'énumération de l'heuristique sur des graphes aléatoires et réguliers sont présentés dans le tableau 2.14 et les figures 2.33 et 2.34. Nous noterons pour les graphes aléatoires que le fait de ne pas utiliser de pivot permet de trouver plus de quasi-cliques. Dans le cas où l'on trouve bien plus de quasi-cliques, l'heuristique sans pivot met un temps supérieur à l'heuristique avec pivot pour énumérer les quasi-cliques qu'elle trouve. Pour les graphes réguliers nous noterons que dans chaque cas le même nombre de quasi-cliques est énuméré et que, dans ce cas là, l'heuristique sans pivot est en moyenne trois plus rapide que l'heuristique avec pivot.

			$PEQCO_{h_p}$		$PEQCO_h$	
Nom	$\omega(G)$	λ/γ_{min}	$temps(s)$	$\|Q\|$	$temps(s)$	$\|Q\|$
erdos_100_0.2	5	0,66	34,649	5 484	25,250	5 522
erdos_100_0.4	8	0,77	3 667	12 222	5 256	14 703
regular_100_3	3	0,5	3,174	283	1,037	283
regular_100_4	3	0,5	3,121	507	1,161	507
regular_100_5	3	0,5	3,361	599	1,140	599
regular_100_6	3	0,5	3,703	704	1,061	704
regular_100_7	3	0,5	3,861	1 017	1,038	1 017
regular_100_8	3	0,5	4,176	1 849	1,170	1 849
regular_100_9	4	0,6	3,699	506	1,112	506
regular_100_10	4	0,6	4,145	802	1,247	802

Tableau 2.14 – Temps d'exécution sur des graphes aléatoires et réguliers pour l'heuristique, avec et sans pivot.

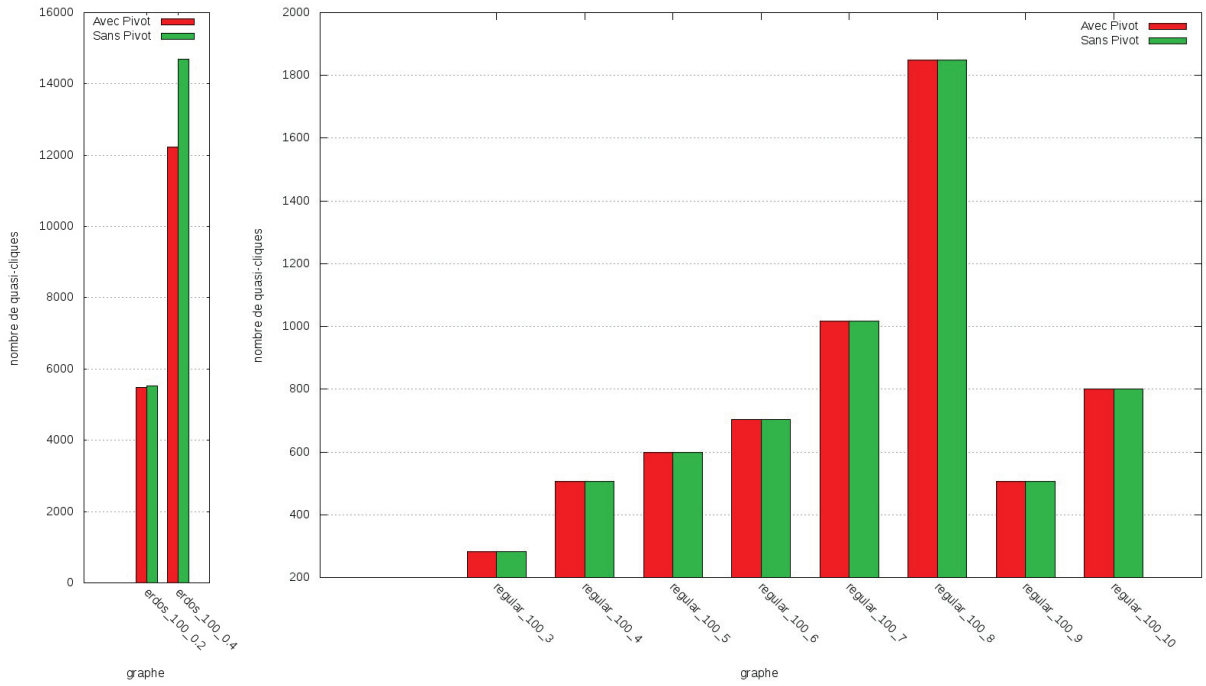


FIGURE 2.33 – Quasi-cliques énumérées par l'heuristique avec ou sans pivot sur des graphes aléatoires et réguliers.

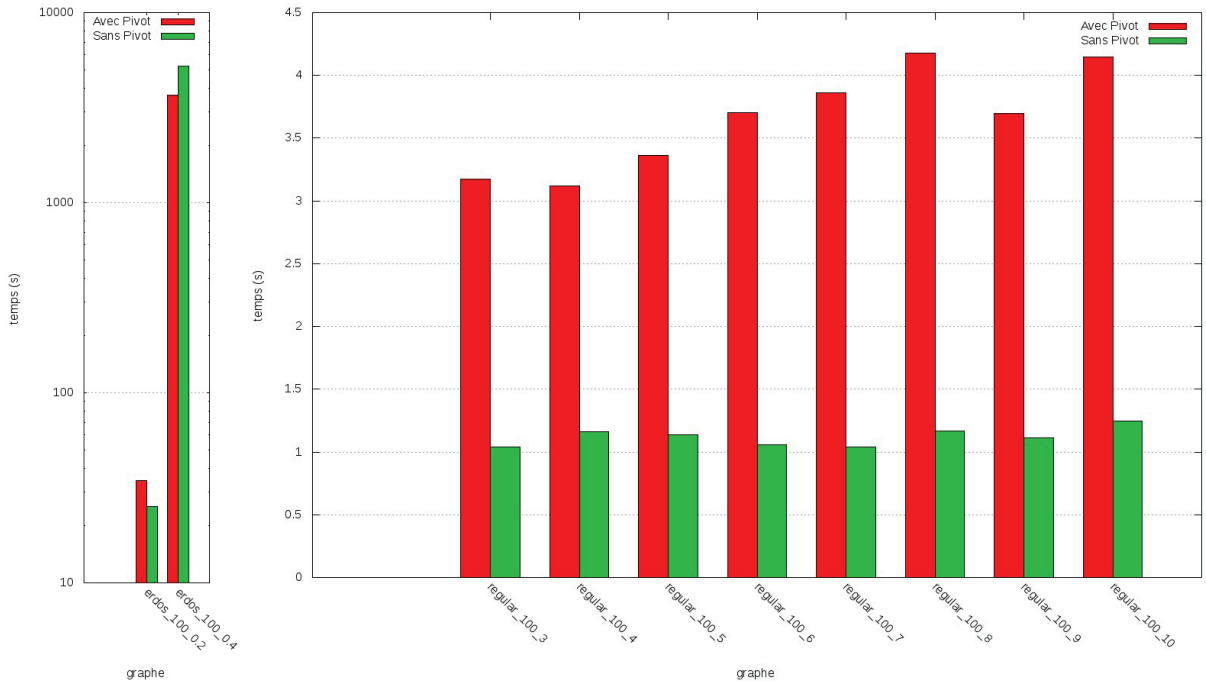


FIGURE 2.34 – Temps d'énumération par l'heuristique avec ou sans pivot sur des graphes aléatoires et réguliers.

Les résultats des tests d'énumération de l'heuristique sur des graphes Power Law sont présentés dans le tableau 2.15 et les figures 2.35 et 2.36. Nous noterons que le fait de ne pas utiliser de pivot permet généralement de trouver plus de quasi-cliques. Dans le cas où l'on trouve plus de quasi-cliques, l'heuristique sans pivot met un temps supérieur à l'heuristique avec pivot pour énumérer les quasi-cliques qu'elle trouve.

			$PEQCO_{h_p}$		$PEQCO_h$	
Nom	$\omega(G)$	λ/γ_{min}	$temps(s)$	$\|Q\|$	$temps(s)$	$\|Q\|$
power_law_100_3_0.2	4	0,6	4,002	250	1,133	250
power_law_100_3_0.4	4	0,6	4,111	210	1,272	211
power_law_100_3_0.6	4	0,6	4,221	210	1,416	210
power_law_100_3_0.8	4	0,6	4,299	235	1,412	235
power_law_100_4_0.2	5	0,66	4,627	643	1,323	652
power_law_100_4_0.4	5	0,66	4,877	725	1,449	737
power_law_100_4_0.6	5	0,66	4,572	611	1,466	615
power_law_100_4_0.8	5	0,66	4,803	501	1,697	508
power_law_100_5_0.2	5	0,66	5,346	1 159	2,122	1 174
power_law_100_5_0.4	5	0,66	6,712	908	3,727	932
power_law_100_5_0.6	5	0,66	8,072	868	3,509	884
power_law_100_5_0.8	6	0,71	4,826	440	1,734	443
power_law_100_6_0.2	5	0,66	10,163	1 422	7,034	1 452
power_law_100_6_0.4	5	0,66	15,992	1 339	18,354	1 382
power_law_100_6_0.6	6	0,71	7,117	553	4,536	555
power_law_100_6_0.8	6	0,71	7,446	661	4,691	664
power_law_100_7_0.2	6	0,71	8,803	777	5,760	779
power_law_100_7_0.4	6	0,71	10,865	702	8,778	710
power_law_100_7_0.6	6	0,71	11,093	762	8,714	768
power_law_100_7_0.8	6	0,71	11,500	715	10,090	724
power_law_100_8_0.2	6	0,71	19,119	1 047	19,931	1 051
power_law_100_8_0.4	6	0,71	23,197	1 127	25,904	1 139
power_law_100_8_0.6	6	0,71	18,531	1 160	25,764	1 167
power_law_100_8_0.8	7	0,75	14,353	1 311	19,161	1 408
power_law_100_9_0.2	6	0,71	56,610	2 612	77,206	2 661
power_law_100_9_0.4	6	0,71	50,773	2 280	91,361	2 359
power_law_100_9_0.6	8	0,77	16,767	900	20,041	1 024
power_law_100_9_0.8	8	0,77	16,955	935	22,774	1 058
power_law_100_10_0.2	6	0,71	83,125	3 035	106,567	3 408
power_law_100_10_0.4	7	0,75	56,807	2 455	86,152	2 664
power_law_100_10_0.6	7	0,75	33,568	2 365	49,661	2 542
power_law_100_10_0.8	10	0,81	14,412	851	13,953	885

Tableau 2.15 – Temps d'exécution sur des graphes Power Law pour l'heuristique, avec et sans pivot.

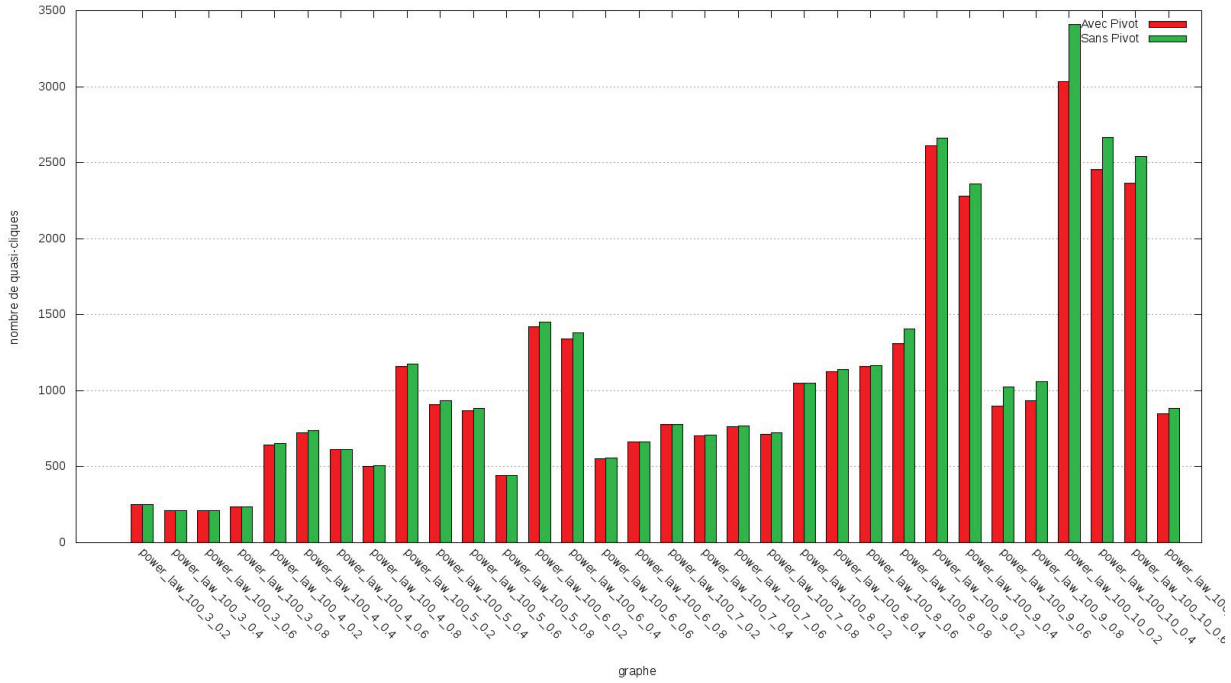


FIGURE 2.35 – Quasi-cliques énumérées par l'heuristique avec ou sans pivot sur des graphes Power Law.

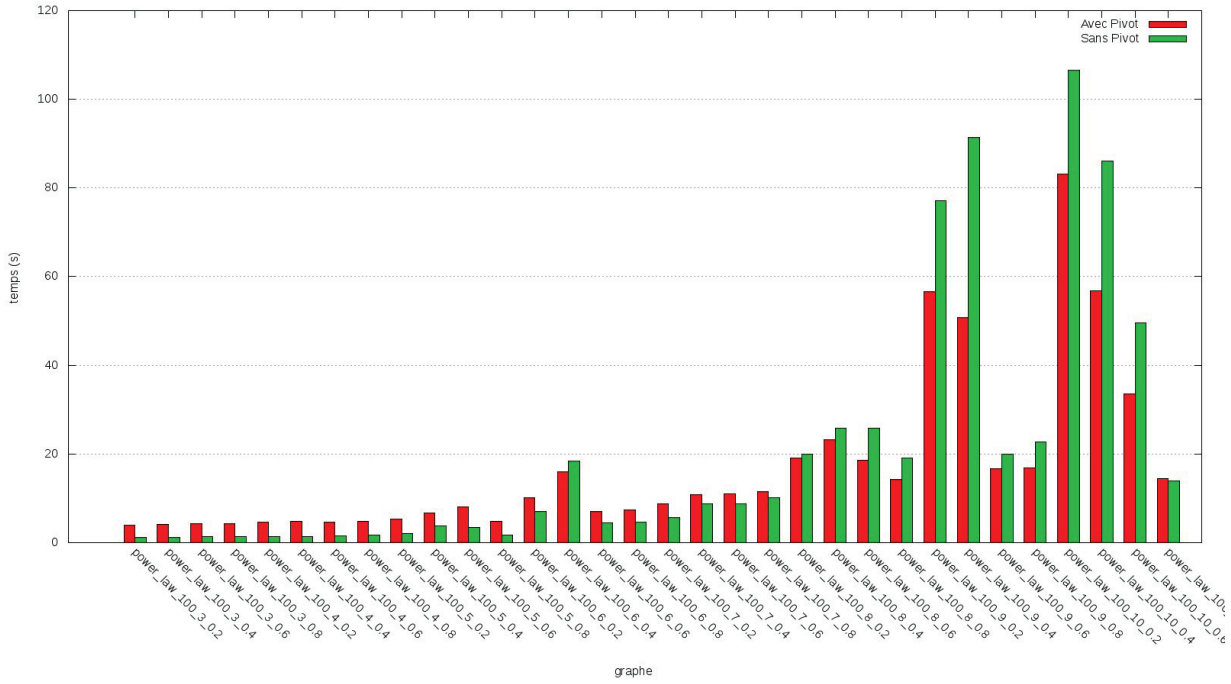


FIGURE 2.36 – Temps d'énumération par l'heuristique avec ou sans pivot sur des graphes Power Law.

Les résultats des tests d'énumération de l'heuristique sur des graphes petit-mondes sont présentés dans le tableau 2.16 et les figures 2.37 et 2.38. Nous noterons que le fait de ne pas utiliser de pivot permet parfois de trouver quelques quasi-cliques supplémentaires. Pour les temps d'exécution on se rapproche de la situation des graphes réguliers (où il y a aussi quasiment le même nombre de quasi-cliques énumérées) : un temps d'exécution trois fois plus rapide pour l'heuristique sans pivot.

			$PEQCO_{h_p}$		$PEQCO_h$	
Nom	$\omega(G)$	λ/γ_{min}	$temps(s)$	$\|Q\|$	$temps(s)$	$\|Q\|$
small_100_3_0.2	3	0,5	3,000	130	1,014	130
small_100_3_0.4	3	0,5	2,982	268	1,086	268
small_100_3_0.6	3	0,5	3,161	378	1,059	378
small_100_3_0.8	3	0,5	3,227	425	1,061	425
small_100_4_0.2	4	0,6	3,188	101	1,141	101
small_100_4_0.4	3	0,5	3,615	592	1,144	592
small_100_4_0.6	4	0,6	3,487	182	1,064	182
small_100_4_0.8	4	0,6	3,605	224	1,149	224
small_100_5_0.2	3	0,5	3,413	278	1,174	278
small_100_5_0.4	4	0,6	3,287	126	1,191	126
small_100_5_0.6	4	0,6	3,529	160	1,023	160
small_100_5_0.8	4	0,6	3,586	260	1,152	260
small_100_6_0.2	5	0,66	3,581	215	1,232	215
small_100_6_0.4	5	0,66	3,749	364	1,278	365
small_100_6_0.6	5	0,66	3,944	647	1,158	648
small_100_6_0.8	5	0,66	4,432	964	1,352	967
small_100_7_0.2	5	0,66	3,565	230	1,066	230
small_100_7_0.4	4	0,6	3,906	250	1,208	250
small_100_7_0.6	5	0,66	4,035	557	1,172	560
small_100_7_0.8	5	0,66	4,544	958	1,228	962
small_100_8_0.2	5	0,66	4,324	205	1,425	205
small_100_8_0.4	6	0,71	4,269	287	1,313	287
small_100_8_0.6	6	0,71	4,671	306	1,360	307
small_100_8_0.8	6	0,71	4,822	428	1,441	429
small_100_9_0.2	6	0,71	3,894	255	1,117	256
small_100_9_0.4	6	0,71	4,168	290	1,182	290
small_100_9_0.6	6	0,71	4,659	350	1,293	350
small_100_9_0.8	6	0,71	4,985	455	1,534	455
small_100_10_0.2	7	0,75	4,373	266	1,406	266
small_100_10_0.4	7	0,75	5,118	340	1,650	342
small_100_10_0.6	7	0,75	5,958	478	2,283	486
small_100_10_0.8	7	0,75	7,017	823	2,811	851

Tableau 2.16 – Temps d'exécution sur des graphes petit-mondes pour l'heuristique, avec et sans pivot.

En conclusion, nous remarquerons que dans les cas où quasiment le même nombre de quasi-cliques est énuméré, le fait de ne pas utiliser le pivot permet de diviser le temps d'exécution par 3. Cependant, dans les cas de graphes plus denses, nous remarquerons que le fait de ne pas utiliser de pivot a comme conséquence d'augmenter la quantité de quasi-cliques trouvées. La conséquence de l'augmentation du nombre de quasi-cliques est que le temps nécessaire pour les énumérer est plus long.

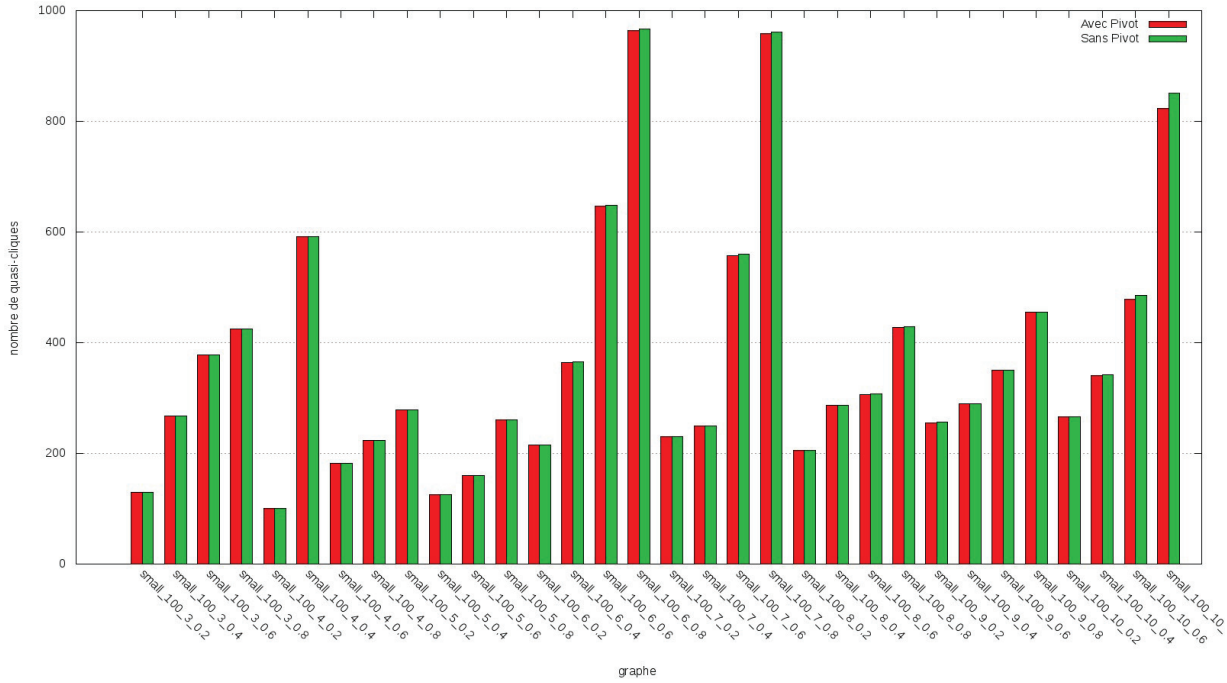


FIGURE 2.37 – Quasi-cliques énumérées par l'heuristique avec ou sans pivot sur des graphes petit-mondes.

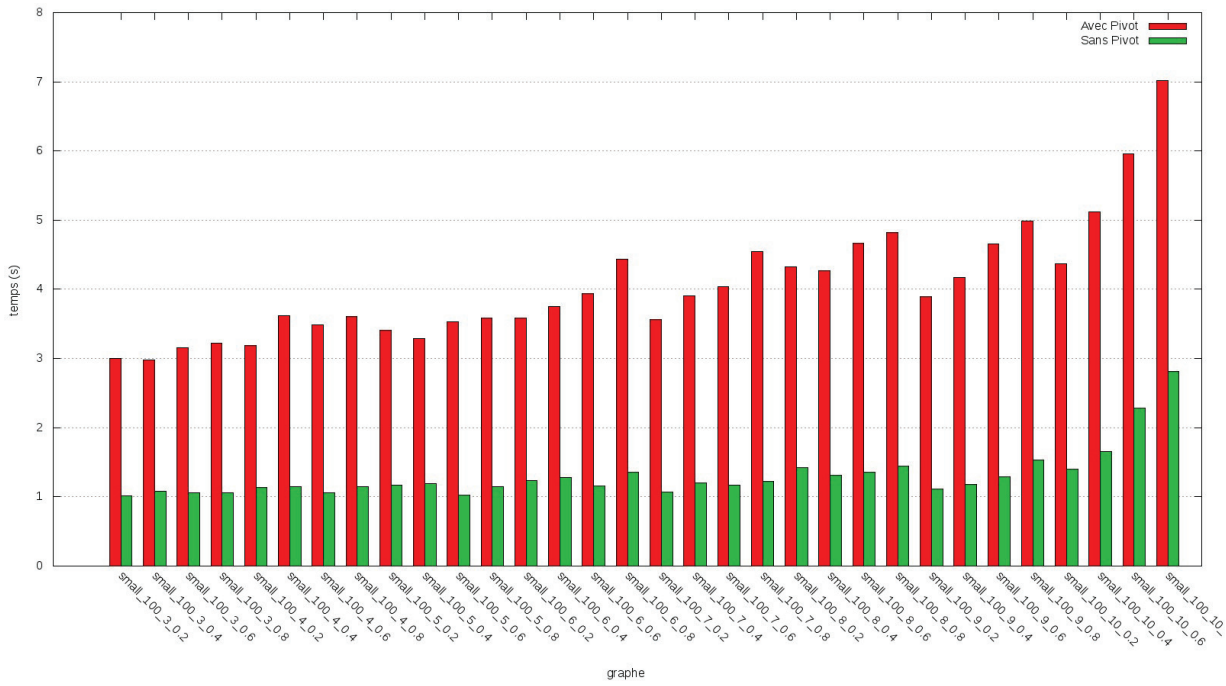


FIGURE 2.38 – Temps d'énumération par l'heuristique avec ou sans pivot sur des graphes petit-mondes.

2.4.3 Étude comparative d'énumérations de quasi-cliques

Notre heuristique d'énumération de quasi-cliques permet une énumération plus rapide même si nous savons qu'elle n'est pas parfaite et qu'elle peut rater l'énumération de quasi-cliques maximales. Nous avons donc choisi de tester notre heuristique et de comparer les résultats observés en terme de qualité par rapport aux algorithmes *RLS* et *DLS-MC* [19].

Nous avons donc adapté les algorithmes *RLS* et *DLS-MC* en proposant une version distribuée et limitée dans le temps de ceux-ci. Le fonctionnement repose sur une distribution des calculs de chacun des noeuds initiaux sur les différents coeurs de la machine. En conséquence plusieurs explorations ont lieu en parallèle. Lorsqu'une exploration est terminée, s'il reste du temps, on en lance une nouvelle.

Pour mesurer la qualité de nos algorithmes nous avons choisi de regarder le silence et le bruit parmi les quasi-cliques énumérées.

Définition 2.23. Le *silence* représente la situation ou des éléments pertinents sont absents des résultats exposés.

Dans notre situation, le silence est la situation où une quasi-clique maximale n'est pas énumérée.

Définition 2.24. Le *bruit* représente la situation ou des éléments non-pertinents sont présents dans les résultats exposés.

Dans notre situation, le bruit est la situation où une quasi-clique énumérée est non-maximale.

Comparaison sur la qualité des quasi-cliques énumérées

Nous avons choisi de tester initialement la qualité des quasi-cliques énumérées, indépendamment du temps de calcul des algorithmes d'énumération de quasi-cliques. Pour cela nous avons énuméré l'ensemble des quasi-cliques à l'aide de l'algorithme puis avons comparé celles trouvées par les algorithmes *RLS*, *DLS-MC* et l'heuristique $PEQCO_h$.

Les résultats sont présentés dans les tableaux 2.17 et 2.18 ainsi que dans les figures 2.39 et 2.40.

Nous noterons que l'algorithme *RLS* donne globalement de bons résultats en termes de quasi-cliques trouvées qui s'avèrent être de véritables quasi-cliques. En conséquence le silence produit est généralement très faible. Cependant nous noterons que le bruit produit est souvent élevé : il y a plus de quasi-cliques énumérés qui sont du bruit que de vrais quasi-cliques maximales.

Pour l'algorithme *DLS-MC* nous remarquerons que le bruit généré est moins important que pour l'algorithme *RLS*. Cependant le silence reste important et nous notons que, en fonction des graphes, il y a généralement autant de bruit que de silence.

Pour notre heuristique $PEQCO_h$ nous noterons que bien que ne trouvant pas autant de quasi-cliques maximales exactes que l'algorithme *RLS* elle a de meilleurs résultats en termes de silence que l'algorithme *DLS-MC*. De plus, elle ne produit jamais de bruit, ce qui permet de s'assurer que les quasi-cliques énumérées sont bien des quasi-cliques maximales.

Comparaison sur le temps des algorithmes d'énumération de quasi-cliques

Nous avons ensuite testé la qualité des quasi-cliques énumérées à temps de calcul quasiment identique pour l'ensemble des algorithmes. Nous avons donc exécuté les algorithmes en imposant un temps d'exécution maximal indexé sur le temps mis par notre heuristique pour terminer les calculs.

Les résultats sont présentés dans les tableaux 2.19 et 2.20 ainsi que dans les figures 2.41 et 2.42.

Nous noterons qu'avec des temps quasiment similaires, les résultats des algorithmes *RLS* et *DLS-MC* sont moins bons car il y a une augmentation du silence. Nous noterons cependant que pour *RLS* la diminution du temps permet également de diminuer le bruit généré.

Finalement, nous remarquerons que notre heuristique permet de fournir des résultats plus fiable dans un même temps de calcul que les algorithmes *RLS* et *DLS-MC*. Ceci s'explique par le fait que l'heuristique a un calcul qui se termine, et l'énumération des quasi-cliques fournies ne comporte pas de bruit. Il faut toutefois noter que l'algorithme *RLS* donne de meilleurs résultats pour le silence. Cependant, il n'est pas possible de déterminer, uniquement à l'aide de l'algorithme *RLS*, quelles quasi-cliques sont maximales et lesquelles sont du bruit.

Nom	λ/γ_{min}	**algo**	*temps (s)*	$\|Q\|$	*exactes*	*silence*	*bruit*
erdos_20_0.2	0,5	RLS	121,382	81	31	13	50
		$DLS-MC$	120,349	73	30	14	43
		$PEQCO_h$	0,529	44	44	0	0
erdos_20_0.8	0,8	RLS	121,727	2 869	760	5	2 109
		$DLS-MC$	42,478	194	91	674	103
		$PEQCO_h$	1,889	99	99	666	0
regular_20_3	0,5	RLS	121,179	49	49	0	0
		$DLS-MC$	120,315	49	49	0	0
		$PEQCO_h$	0,539	49	49	0	0
regular_20_10	0,67	RLS	121,567	1 421	241	19	1 180
		$DLS-MC$	120,574	220	24	236	196
		$PEQCO_h$	2,034	229	229	31	0
power_law_20_4_0.2	0,6	RLS	121,383	217	67	10	150
		$DLS-MC$	120,973	80	35	42	45
		$PEQCO_h$	0,849	76	76	1	0
power_law_20_4_0.8	0,67	RLS	121,278	76	57	4	19
		$DLS-MC$	120,712	64	51	10	13
		$PEQCO_h$	0,811	18	18	43	0
power_law_20_9_0.4	0,75	RLS	121,479	388	90	1	298
		$DLS-MC$	120,523	93	56	35	37
		$PEQCO_h$	1,419	84	84	7	0
power_law_20_10_0.6	0,75	RLS	121,501	434	107	0	327
		$DLS-MC$	120,777	92	77	30	15
		$PEQCO_h$	1,598	73	73	34	0
small_20_3_0.2	0,5	RLS	121,162	30	30	0	0
		$DLS-MC$	120,591	30	30	0	0
		$PEQCO_h$	0,526	30	30	0	0
small_20_3_0.8	0,5	RLS	121,329	81	34	12	47
		$DLS-MC$	120,329	71	33	13	38
		$PEQCO_h$	0,594	46	46	0	0
small_20_10_0.2	0,75	RLS	121,487	501	177	7	324
		$DLS-MC$	120,841	154	94	90	60
		$PEQCO_h$	2,054	35	35	149	0
small_20_10_0.8	0,86	RLS	122,115	638	27	0	611
		$DLS-MC$	120,563	8	2	25	6
		$PEQCO_h$	21,968	15	15	12	0

Tableau 2.17 – Résultats comparatifs de différents algorithmes d'énumération de quasi-cliques pour des graphes de taille 20.

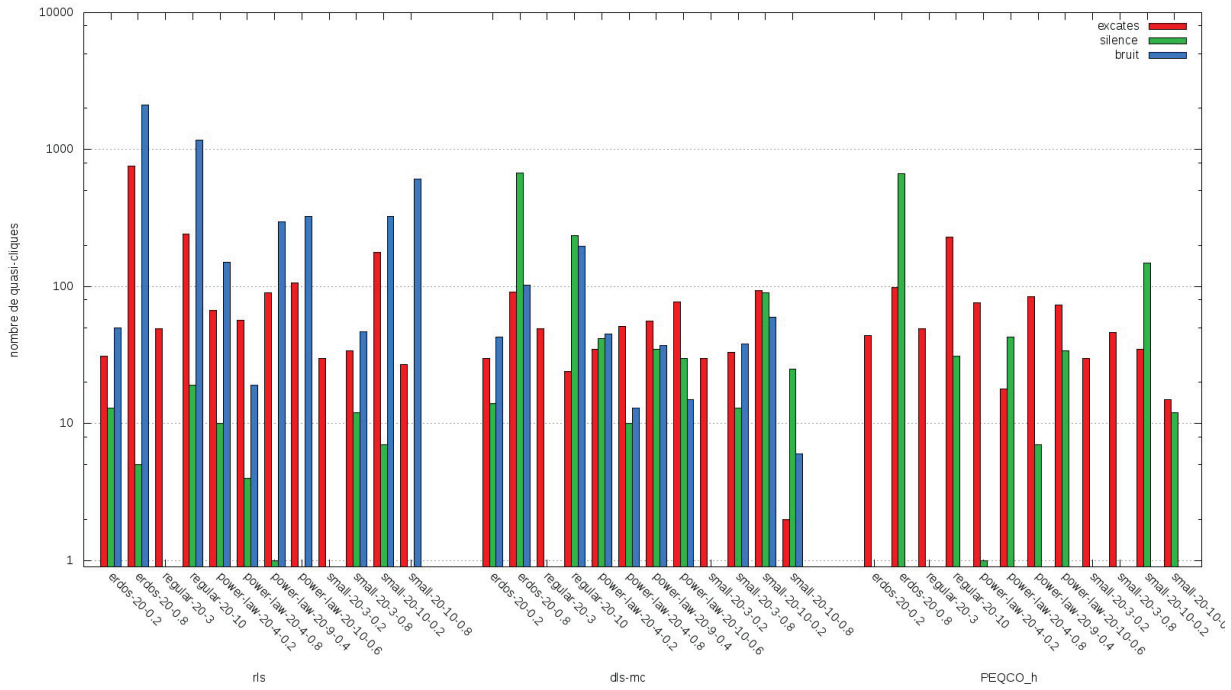


FIGURE 2.39 – Résultats comparatifs de différents algorithmes d'énumération de quasi-cliques pour des graphes de taille 20.

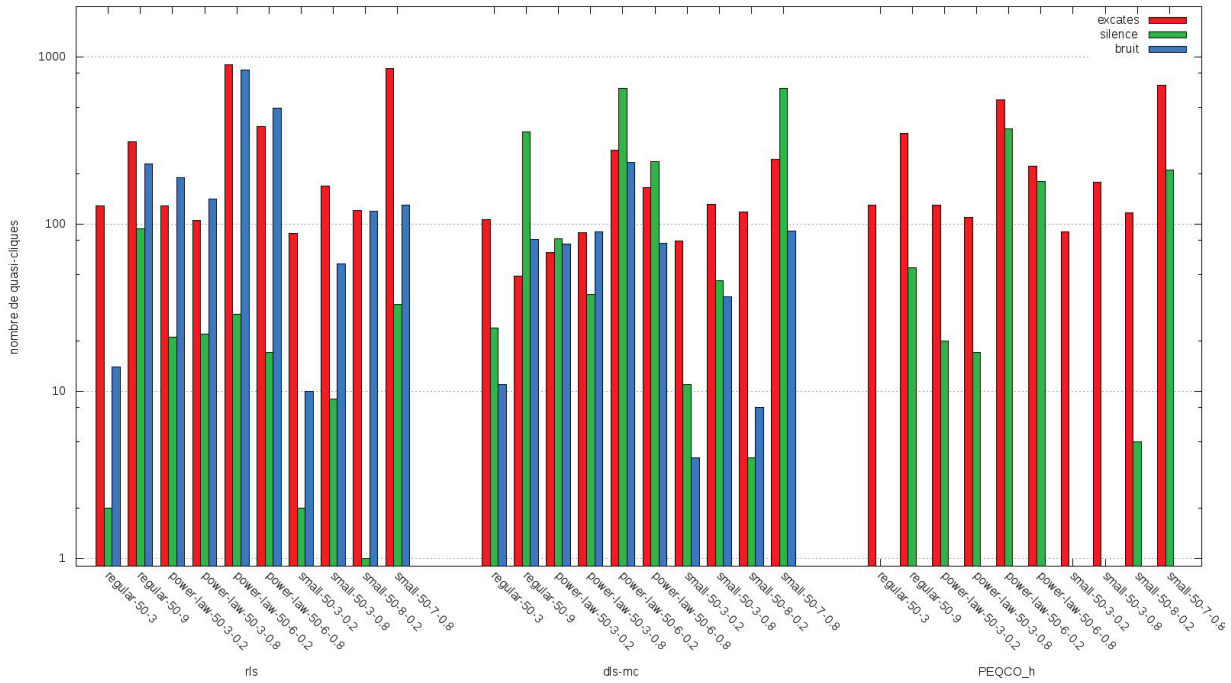


FIGURE 2.40 – Résultats comparatifs de différents algorithmes d'énumération de quasi-cliques pour des graphes de taille 50.

Nom	λ/γ_{min}	**algo**	*temps* (s)	$\|Q\|$	*exactes*	*silence*	*bruit*
regular_50_3	0,5	RLS	121,446	142	128	2	14
		$DLS-MC$	121,254	117	106	24	11
		$PEQCO_h$	0,586	130	130	0	0
regular_50_9	0,6	RLS	121,592	539	310	94	229
		$DLS-MC$	121,402	130	49	355	81
		$PEQCO_h$	1,149	349	349	55	0
power_law_50_3_0.2	0,6	RLS	121,513	318	129	21	189
		$DLS-MC$	120,968	144	68	82	76
		$PEQCO_h$	1,015	130	130	20	0
power_law_50_3_0.8	0,6	RLS	121,631	247	105	22	142
		$DLS-MC$	120,433	179	89	38	90
		$PEQCO_h$	1,373	110	110	17	0
power_law_50_6_0.2	0,67	RLS	122,114	1 733	895	29	838
		$DLS-MC$	123,183	510	276	648	234
		$PEQCO_h$	3,865	553	553	371	0
power_law_50_6_0.8	0,71	RLS	121,868	881	385	17	496
		$DLS-MC$	122,178	242	165	237	77
		$PEQCO_h$	3,207	222	222	180	0
small_50_3_0.2	0,5	RLS	121,456	98	88	2	10
		$DLS-MC$	121,224	83	79	11	4
		$PEQCO_h$	0,474	90	90	0	0
small_50_3_0.8	0,5	RLS	121,622	227	169	9	58
		$DLS-MC$	120,482	169	132	46	37
		$PEQCO_h$	0,572	178	178	0	0
small_50_8_0.2	0,71	RLS	121,781	241	121	1	120
		$DLS-MC$	121,050	126	118	4	8
		$PEQCO_h$	0,829	117	117	5	0
small_50_7_0.8	0,67	RLS	121,782	987	857	33	130
		$DLS-MC$	122,199	335	244	646	91
		$PEQCO_h$	1,882	679	679	211	0

Tableau 2.18 – Résultats comparatifs de différents algorithmes d'énumération de quasi-cliques pour des graphes de taille 50.

Nom	λ/γ_{min}	**algo**	*temps (s)*	$\|Q\|$	*exactes*	*silence*	*bruit*
erdos_20_0.2	0,5	rls	1,727	69	28	16	41
		$dls-mc$	0,999	72	30	14	42
		$PEQCO\_h$	0,529	44	44	0	0
erdos_20_0.8	0,8	rls	3,391	355	279	486	76
		$dls-mc$	2,269	119	52	713	67
		$PEQCO\_h$	1,889	99	99	666	0
regular_20_3	0,5	rls	1,634	49	49	0	0
		$dls-mc$	0,973	49	49	0	0
		$PEQCO\_h$	0,539	49	49	0	0
regular_20_10	0,67	rls	3,291	615	160	100	455
		$dls-mc$	2,461	220	24	236	196
		$PEQCO\_h$	2,034	229	229	31	0
power_law_20_4_0.2	0,6	rls	2,360	150	57	20	93
		$dls-mc$	1,359	77	33	44	44
		$PEQCO\_h$	0,849	76	76	1	0
power_law_20_4_0.8	0,67	rls	2,224	69	50	11	19
		$dls-mc$	1,331	55	44	17	11
		$PEQCO\_h$	0,811	18	18	43	0
power_law_20_9_0.4	0,75	rls	2,829	144	55	36	89
		$dls-mc$	1,904	78	49	42	29
		$PEQCO\_h$	1,419	84	84	7	0
power_law_20_10_0.6	0,75	rls	3,051	136	75	32	61
		$dls-mc$	2,094	83	69	38	14
		$PEQCO\_h$	1,598	73	73	34	0
small_20_3_0.2	0,5	rls	1,652	30	30	0	0
		$dls-mc$	0,967	29	29	1	0
		$PEQCO\_h$	0,526	30	30	0	0
small_20_3_0.8	0,5	rls	1,817	76	32	14	44
		$dls-mc$	1,058	71	33	13	38
		$PEQCO\_h$	0,594	46	46	0	0
small_20_10_0.2	0,75	rls	3,440	294	161	23	133
		$dls-mc$	2,582	139	86	98	53
		$PEQCO\_h$	2,054	35	35	149	0
small_20_10_0.8	0,86	rls	21,370	290	23	4	267
		$dls-mc$	22,420	8	2	25	6
		$PEQCO\_h$	21,968	15	15	12	0

Tableau 2.19 – Résultats temporels de différents algorithmes d'énumération de quasi-cliques pour des graphes de taille 20.

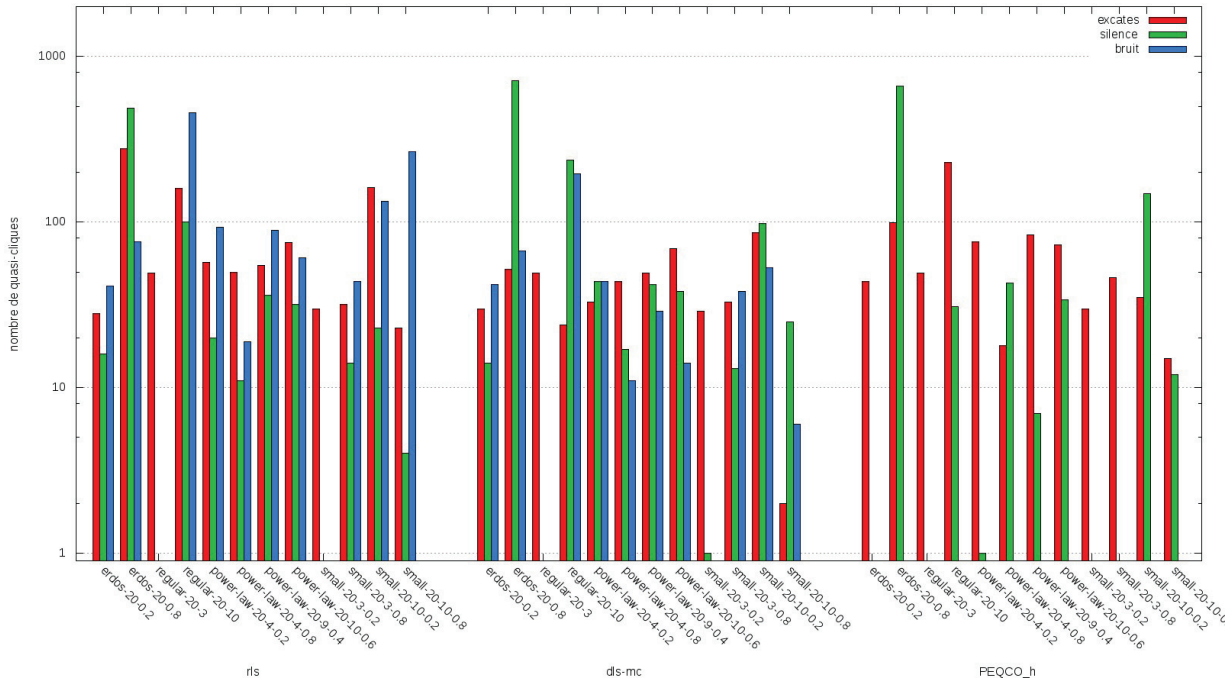


FIGURE 2.41 – Résultats temporels de différents algorithmes d'énumération de quasi-cliques pour des graphes de taille 20.

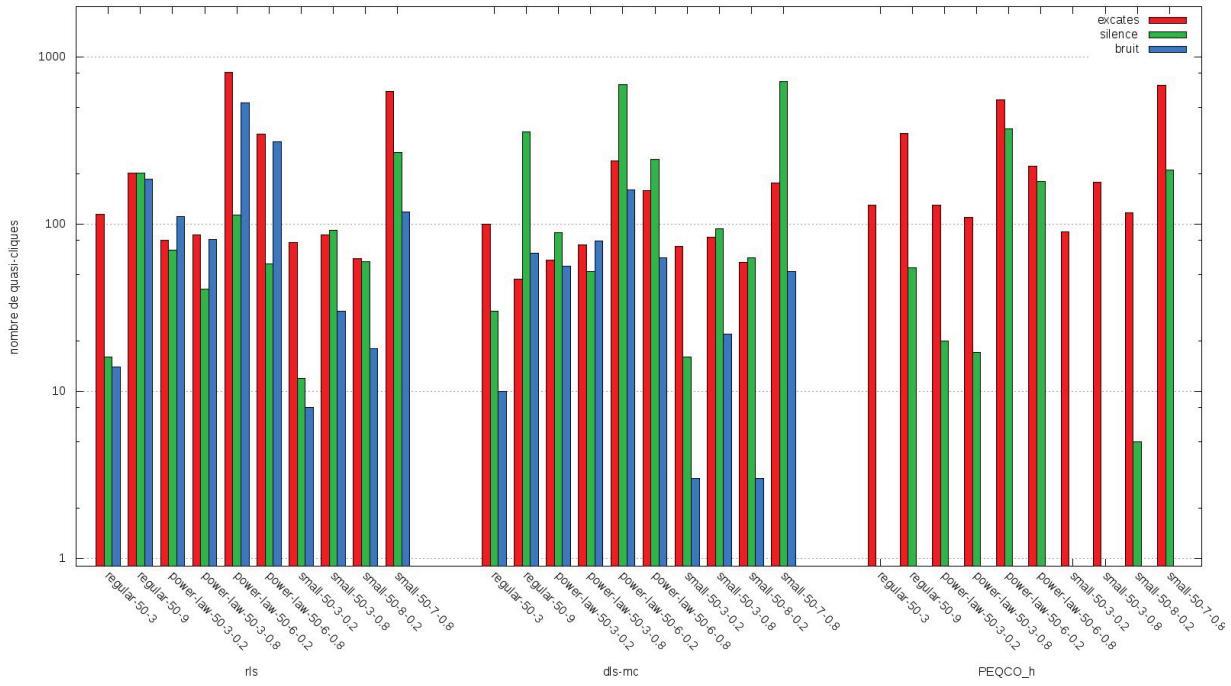


FIGURE 2.42 – Résultats temporels de différents algorithmes d'énumération de quasi-cliques pour des graphes de taille 50.

Nom	λ/γ_{min}	**algo**	*temps (s)*	$\|Q\|$	*exactes*	*silence*	*bruit*
regular_50_3	0,5	rls	2,134	128	114	16	14
		$dls-mc$	1,109	110	100	30	10
		$PEQCO\_h$	0,586	130	130	0	0
regular_50_9	0,6	rls	2,799	388	202	202	186
		$dls-mc$	1,690	114	47	357	67
		$PEQCO\_h$	1,149	349	349	55	0
power_law_50_3_0.2	0,6	rls	2,476	191	80	70	111
		$dls-mc$	1,453	117	61	89	56
		$PEQCO\_h$	1,015	130	130	20	0
power_law_50_3_0.8	0,6	rls	2,899	167	86	41	81
		$dls-mc$	1,927	154	75	52	79
		$PEQCO\_h$	1,373	110	110	17	0
power_law_50_6_0.2	0,67	rls	5,312	1340	811	113	529
		$dls-mc$	4,372	400	239	685	161
		$PEQCO\_h$	3,865	553	553	371	0
power_law_50_6_0.8	0,71	rls	4,858	656	344	58	312
		$dls-mc$	3,715	221	158	244	63
		$PEQCO\_h$	3,207	222	222	180	0
small_50_3_0.2	0,5	rls	1,890	86	78	12	8
		$dls-mc$	0,993	77	74	16	3
		$PEQCO\_h$	0,474	90	90	0	0
small_50_3_0.8	0,5	rls	2,132	116	86	92	30
		$dls-mc$	1,253	106	84	94	22
		$PEQCO\_h$	0,572	178	178	0	0
small_50_8_0.2	0,71	rls	2,598	80	62	60	18
		$dls-mc$	1,391	62	59	63	3
		$PEQCO\_h$	0,829	117	117	5	0
small_50_7_0.8	0,67	rls	3,673	740	622	268	118
		$dls-mc$	2,446	228	176	714	52
		$PEQCO\_h$	1,882	679	679	211	0

Tableau 2.20 – Résultats temporels de différents algorithmes d'énumération de quasi-cliques pour des graphes de taille 50.

2.4.4 Étude de l'influence des paramètres

Les paramètres de voisinage minimum λ_{min} et de densité minimum λ_{min} ont une influence sur le nombre de quasi-cliques énumérées, affectant ainsi le temps d'exécution des algorithmes. En effet, tel que le présente la figure 2.43, les paramètres λ_{min} et γ_{min} ont une influence directe sur les quasi-cliques énumérées par l'algorithme.

Si l'on cherche des quasi-cliques avec $\lambda_{min} = \gamma_{min} = \frac{1}{2}$, on va trouver 17 quasi-cliques :

— 1 $\frac{3}{4} - \frac{8}{10}$ – *clique* composée des noeud {0,1,2,3,4}

— 4 $\frac{2}{4} - \frac{6}{10}$ – *cliques* composées d'un noeud parmi {5,6,7,8} et des 4 noeuds {1,2,3,4}

— 4 $\frac{2}{4} - \frac{6}{10}$ – *cliques* composées de deux noeuds parmi {5,6,7,8} (et dont le plus court chemin est égal à deux) et de leurs 3 noeuds voisins parmi {1,2,3,4}

— 8 $\frac{2}{4}$-$\frac{7}{10}$ – *cliques* composées du noeud 0, de 3 noeuds parmi {1,2,3,4} et d'un noeud parmi {5,6,7,8} ayant deux voisins avec la quasi-clique

Si l'on cherche des quasi-cliques avec $\lambda_{min} = \frac{1}{2}$ et $\gamma_{min} = \frac{2}{3}$, on va trouver 9 quasi-cliques :

— 1 $\frac{3}{4} - \frac{8}{10}$ – *clique* composée des noeuds {0,1,2,3,4}

— 8 $\frac{2}{4}$-$\frac{7}{10}$ – *cliques* composées du noeud 0, de 3 noeuds parmi {1,2,3,4} et d'un noeud parmi {5,6,7,8} ayant deux voisins avec la quasi-clique

Si l'on cherche des quasi-cliques avec $\lambda_{min} = \gamma_{min} = \frac{2}{3}$, on va trouver 5 quasi-cliques :

— 1 $\frac{3}{4} - \frac{8}{10}$ – *clique* composée des noeud {0,1,2,3,4}

— 4 $\frac{2}{3}$-$\frac{5}{6}$-cliques composées du noeud 0, de 2 noeuds voisins parmi {1,2,3,4} et d'un noeud parmi {5,6,7,8} ayant deux voisins avec la quasi-clique

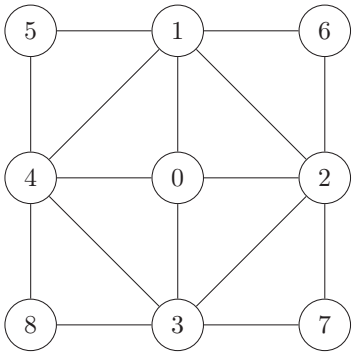


FIGURE 2.43 – Énumération de quasi-cliques pour un graphe donné avec différentes valeurs de paramètres λ_{min} et γ_{min}.

Nous avons testé l'influence des paramètres λ et γ sur le temps et le nombre de quasi-cliques.

Résultats

Nous avons effectué des tests sur 4 graphes, présentés dans le tableau 2.21.

Nom	$\omega(G)$
erdos_100_0.2	5
regular_100_10	4
power_law_100_7_0.2	6
small_100_10_0.8	7

Tableau 2.21 – Présentation de graphes de tailles 100 sur lesquels sont testées différentes valeurs λ et γ.

Les tests ont été effectués sur une machine Debian avec un processeur *Intel(R) Core(TM) i5-4460 CPU @ 3.20GHz* et 16 Giga de RAM. Nous avons testé le temps mis par l'heuristique sans pivot avec différentes valeurs λ et γ. Nous avons testé les valeurs *0,9*, *0,8*, *0,75*, *0,66*, *0,6* et *0,5*.

Pour le graphe *erdos*, les résultats sont présentés dans le tableau 2.22 et les figures 2.44, 2.45 et 2.46.

λ	γ	temps(s)	|Q|	% quasi-cliques
1,0	1,0	1,087	789	0
0,9	0,9	1,139	789	0
0,8	0,9	1,098	784	0,12
	0,8	2,990	642	1,40
0,75	0,9	1,278	829	22,7
	0,8	4,219	1 498	94,6
	0,75	4,751	1 498	94,6
0,66	0,9	1,089	829	22,7
	0.8	5,070	2 808	99,1
	0,75	4,571	2 775	99,1
	0,66	52,247	5 484	100
0,6	0,9	1,171	829	22,7
	0,8	5,225	2 618	99,0
	0,75	5,317	2 472	99,0
	0,66	59,894	18 837	100
	0,6	397,444	17 804	100
0,5	0,9	1,035	829	22,7
	0,8	5,143	3 745	99,3
	0,75	5,596	3 557	99,3
	0,66	78,916	27 418	100
	0,6	558,507	63 182	100
	0,5	58 132	27 698	100

Tableau 2.22 – Temps et quasi-cliques énumérées pour différentes valeurs λ et γ pour le graphe *erdos_ 100_ 0.2*.

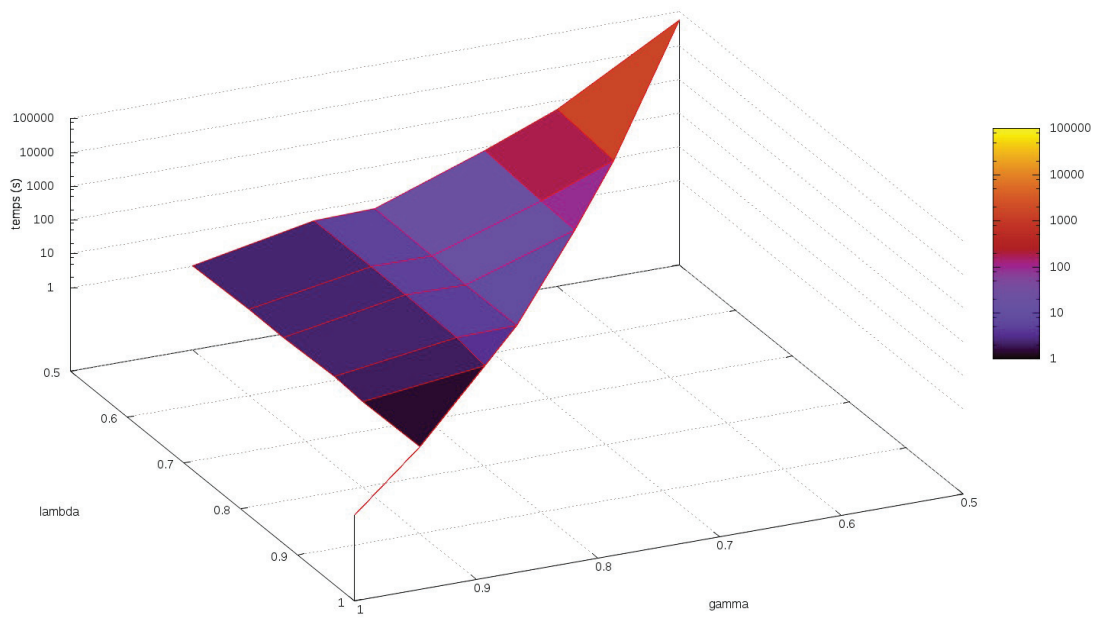


FIGURE 2.44 – Résultats temporels pour différentes valeurs λ et γ pour le graphe *erdos_ 100_ 0.2*.

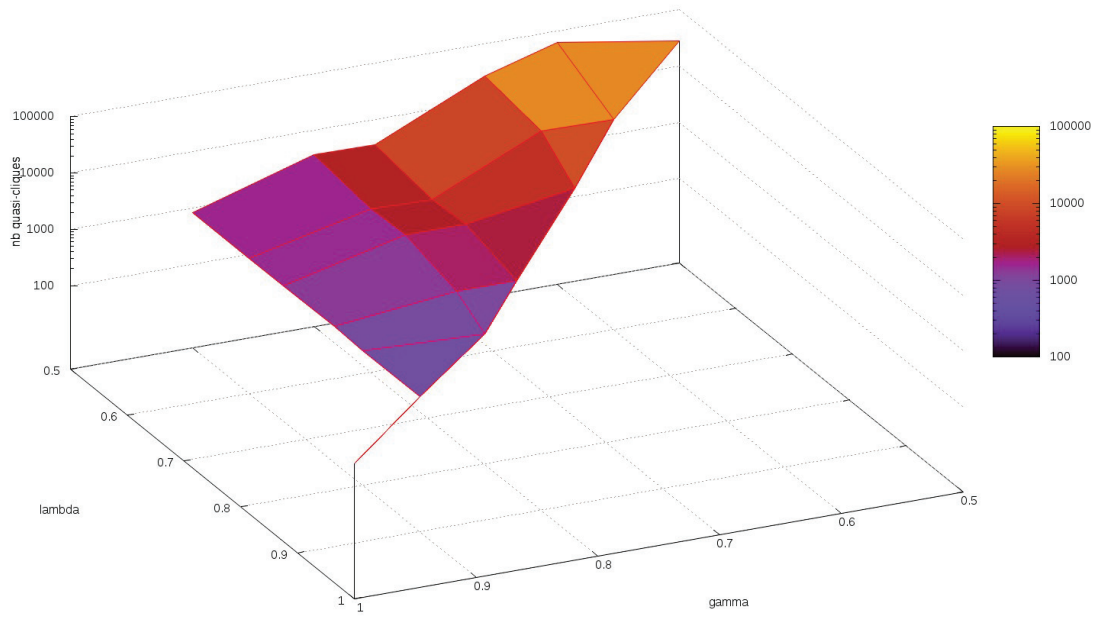


FIGURE 2.45 – Nombre de quasi-cliques énumérées pour différentes valeurs λ et γ pour le graphe *erdos_100_0.2*.

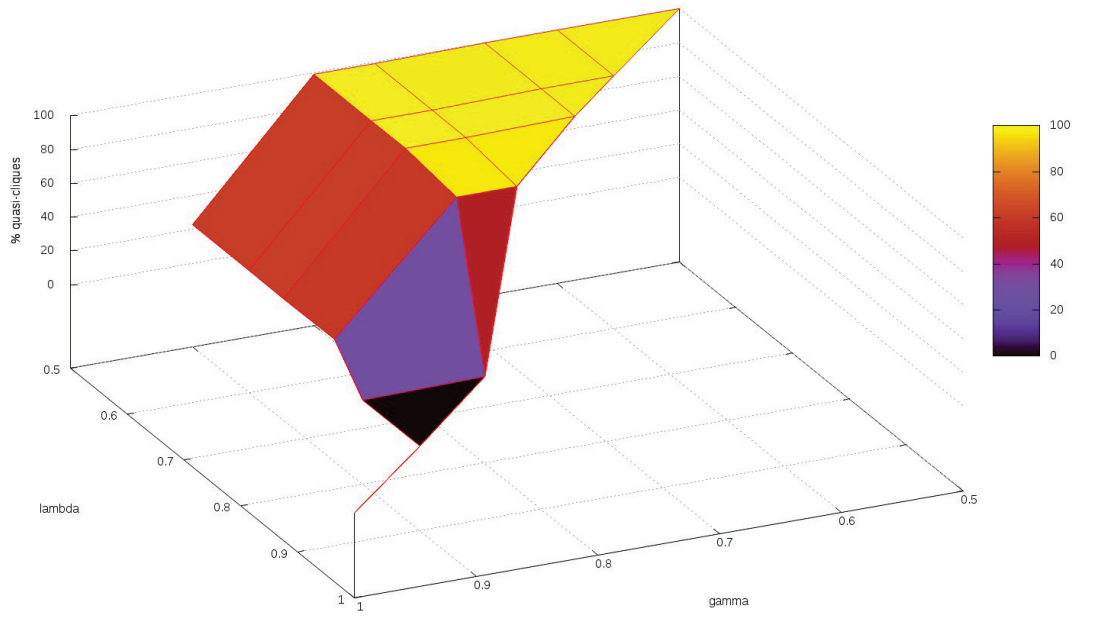


FIGURE 2.46 – Pourcentage de quasi-cliques énumérées pour différentes valeurs λ et γ pour le graphe *erdos_100_0.2*.

Pour le graphe *regular*, les résultats sont présentés dans le tableau 2.23 et les figures 2.47, 2.48 et 2.49.

λ	γ	temps(s)	|Q|	% quasi-cliques
1,0	1,0	0,656	358	0,0
0,9	0,9	0,675	358	0,0
0,8	0,9	0,665	358	0,0
	0,8	0,660	358	0,0
0,75	0,9	0,698	358	0,0
	0,8	0,816	358	0,0
	0,75	0,675	358	0,0
0,66	0,9	0,686	358	0,0
	0.8	0,753	352	21,0
	0,75	0,787	352	21,0
	0,66	1,057	877	97,5
0,6	0,9	0,648	358	0,0
	0,8	0,684	352	21,0
	0,75	0,680	352	21,0
	0,66	1,125	850	99,4
	0,6	1,229	802	99,4
0,5	0,9	0,678	358	0,0
	0,8	0,704	349	20,6
	0,75	0,683	349	20,6
	0,66	1,103	2 265	100
	0,6	1,783	2 814	100
	0,5	3,195	5 997	100

Tableau 2.23 – Temps et quasi-cliques énumérées pour différentes valeurs λ et γ pour le graphe *regular_100_10*.

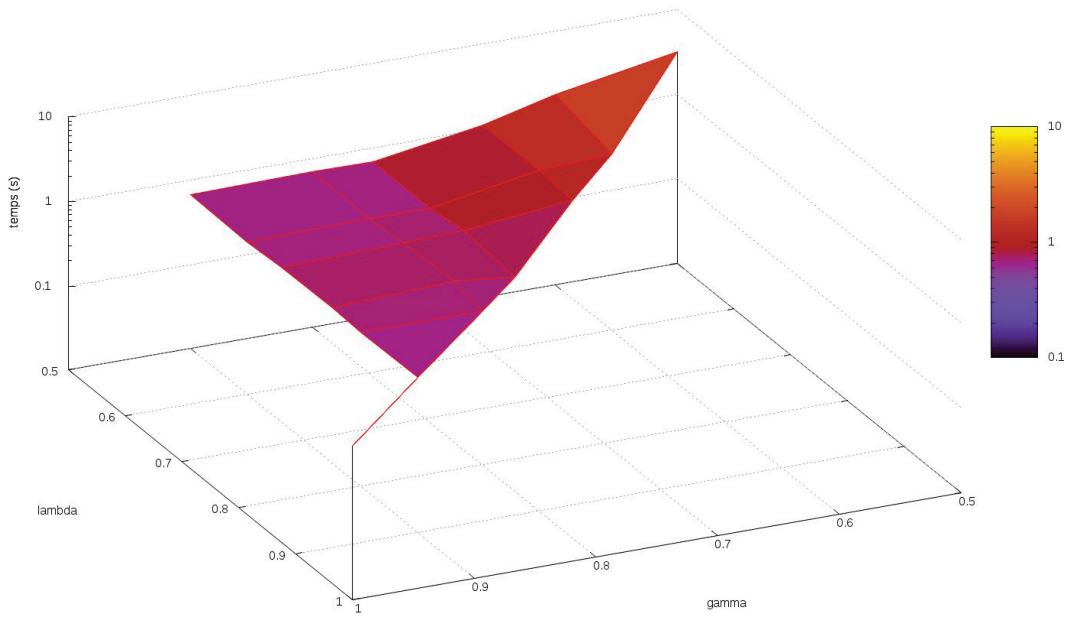


FIGURE 2.47 – Résultats temporels pour différentes valeurs λ et γ pour le graphe *regular_100_10*.

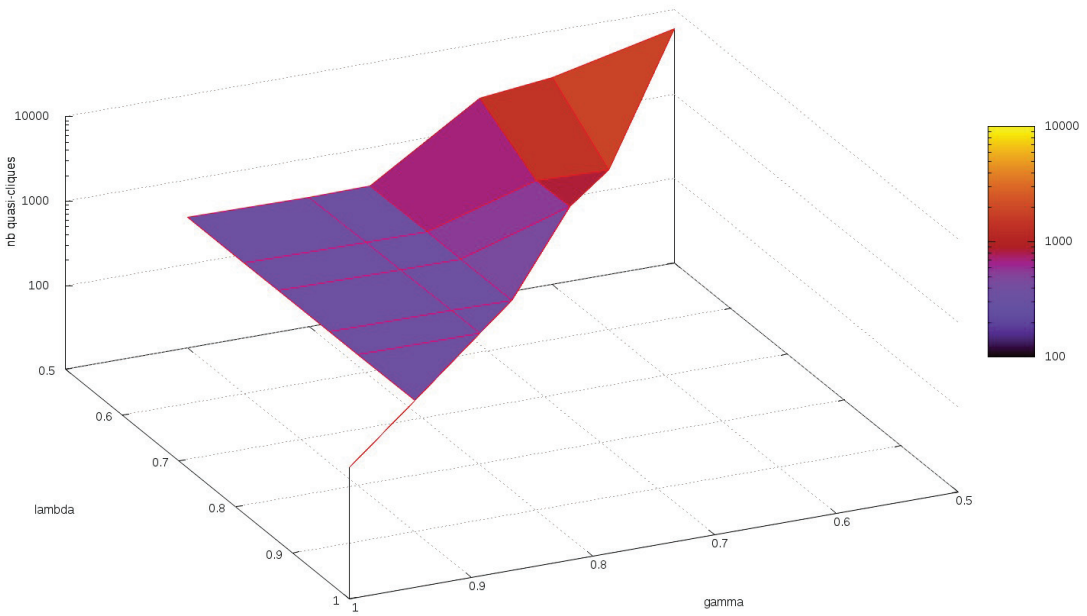


FIGURE 2.48 – Nombre de quasi-cliques énumérées pour différentes valeurs λ et γ pour le graphe *regular_100_10*.

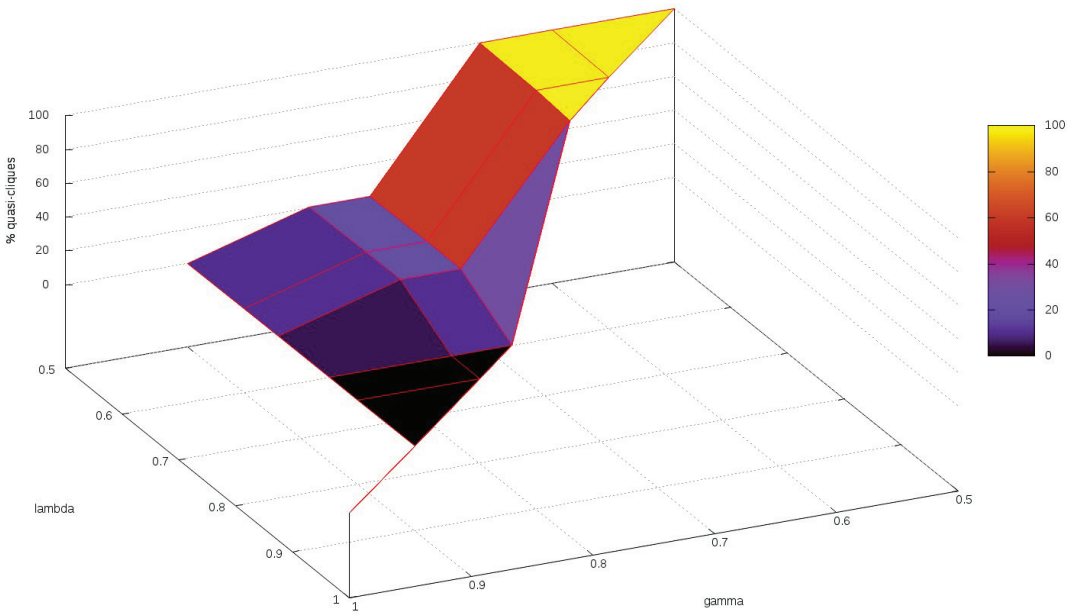


FIGURE 2.49 – Pourcentage de quasi-cliques énumérées pour différentes valeurs λ et γ pour le graphe *regular_100_10*.

Pour le graphe *power_ law*, les résultats sont présentés dans le tableau 2.24 et les figures 2.50, 2.51 et 2.52.

λ	γ	temps(s)	|Q|	% quasi-cliques
1,0	1,0	0,961	436	0
0,9	0,9	0,969	436	0
0,8	0,9	1,334	362	5,0
	0,8	3,354	435	40,2
0,75	0,9	1,330	810	68,0
	0,8	4,036	1 262	93,3
	0,75	4,592	1 246	93,3
0,66	0,9	1,239	812	68,2
	0.8	5,665	1 785	97,2
	0,75	7,164	2 266	97,8
	0,66	46,940	2 339	99,9
0,6	0,9	1,231	812	68,2
	0,8	5,788	3 266	98,5
	0,75	8,426	2 294	97,9
	0,66	80,383	5 851	100
	0,6	309,162	3 143	100
0,5	0,9	1,422	801	67,8
	0,8	6,502	4 172	98,9
	0,75	10,635	4 956	99,1
	0,66	161,203	42 744	100
	0,6	1 183	32 464	100
	0,5	25 253	12 731	100

Tableau 2.24 – Temps et quasi-cliques énumérées pour différentes valeurs λ et γ pour le graphe *power_ law_ 100_ 7_ 0.2*.

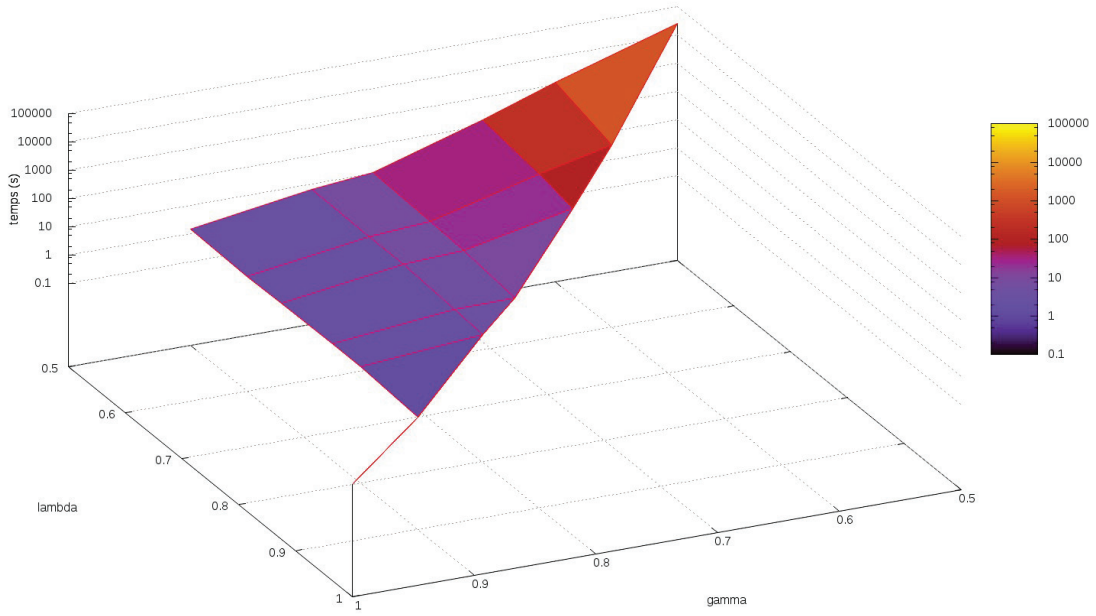


FIGURE 2.50 – Résultats temporels pour différentes valeurs λ et γ pour le graphe *power_ law_ 100_ 7_ 0.2*.

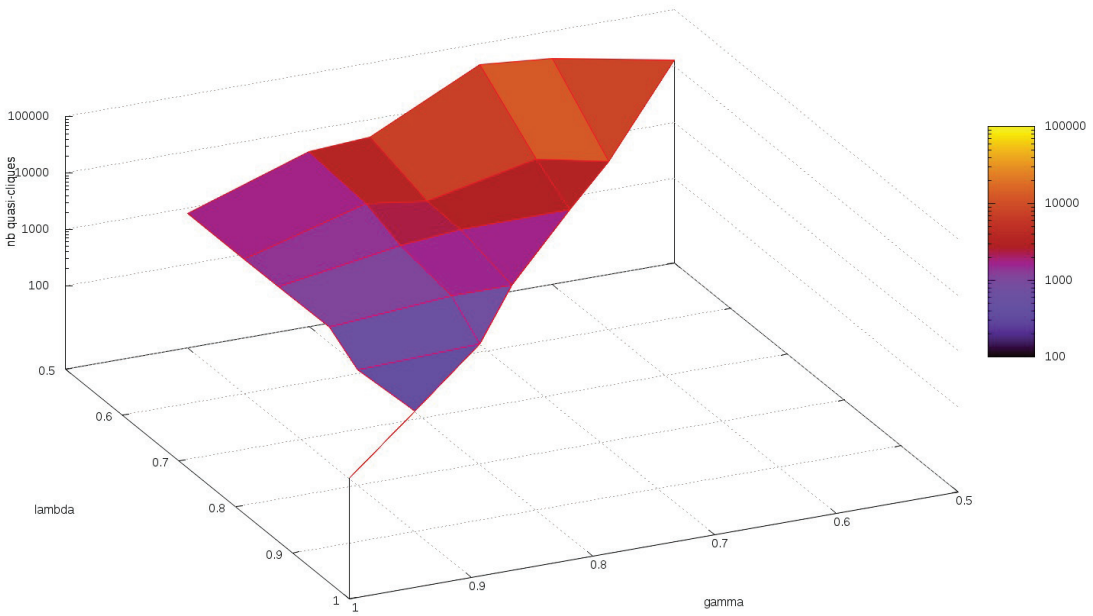


FIGURE 2.51 – Nombre de quasi-cliques énumérées pour différentes valeurs λ et γ pour le graphe *power_law_100_7_0.2*.

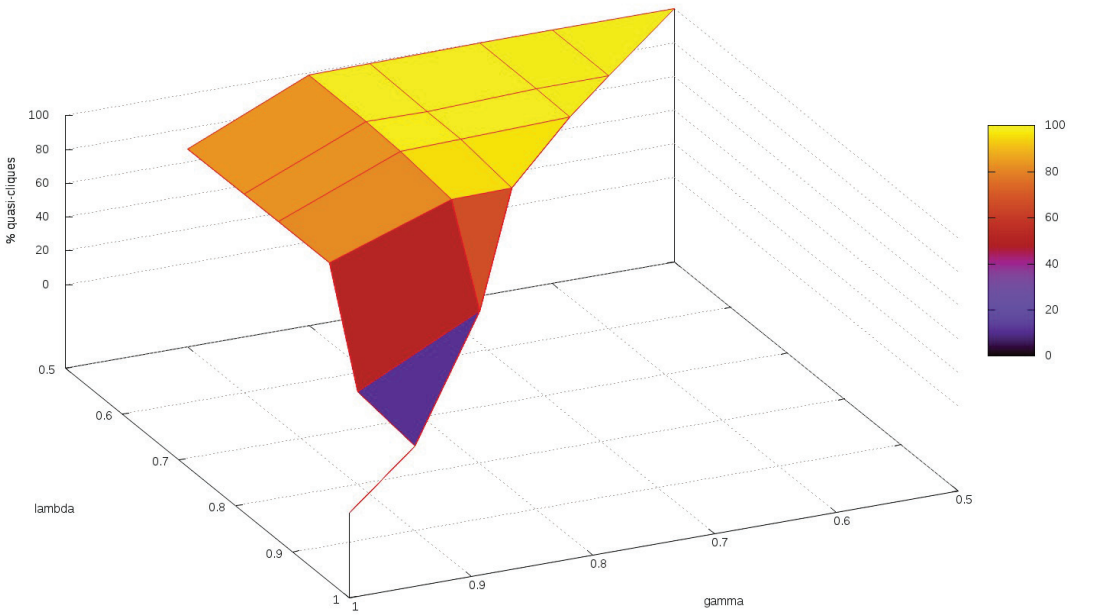


FIGURE 2.52 – Pourcentage de quasi-cliques énumérées pour différentes valeurs λ et γ pour le graphe *power_law_100_7_0.2*.

Pour le graphe *small*, les résultats sont présentés dans le tableau 2.25 et les figures 2.53, 2.54 et 2.55.

λ	γ	temps(s)	|Q|	% quasi-cliques
1,0	1,0	1,103	427	0
0,9	0,9	1,104	427	0
0,8	0,9	1,393	534	41,8
	0,8	3,114	549	52,6
0,75	0,9	1,479	770	66,2
	0,8	4,232	926	93,1
	0,75	4,716	866	92,6
0,66	0,9	1,800	931	72,5
	0,8	6,696	2 181	98,5
	0,75	7,319	2 062	98,4
	0,66	35,379	2 715	100
0,6	0,9	1,788	931	72,5
	0,8	7,460	2 840	98,8
	0,75	10,886	2 151	98,4
	0,66	73,017	7 475	100
	0,6	262,270	4 624	100
0,5	0,9	1,357	876	70,8
	0,8	8,157	8 157	99,2
	0,75	17,113	4 764	99,3
	0,66	207,814	18 381	100
	0,6	1 378	29 021	100
	0,5	45 441	10 466	100

Tableau 2.25 – Temps et quasi-cliques énumérées pour différentes valeurs λ et γ pour le graphe *small_100_10_0.8*.

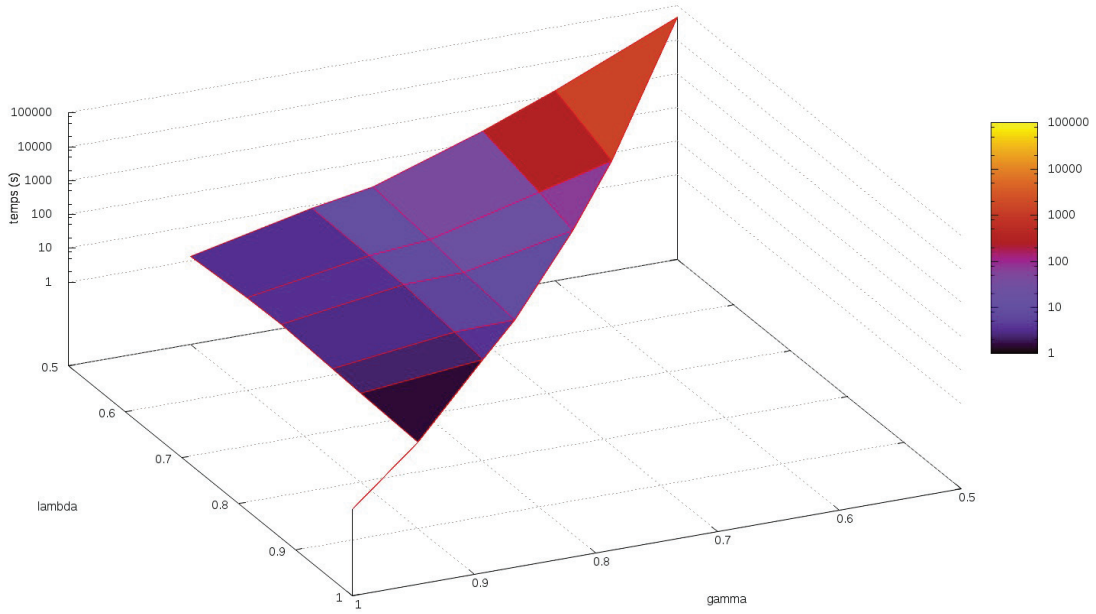


FIGURE 2.53 – Résultats temporels pour différentes valeurs λ et γ pour le graphe *small_100_10_0.8*.

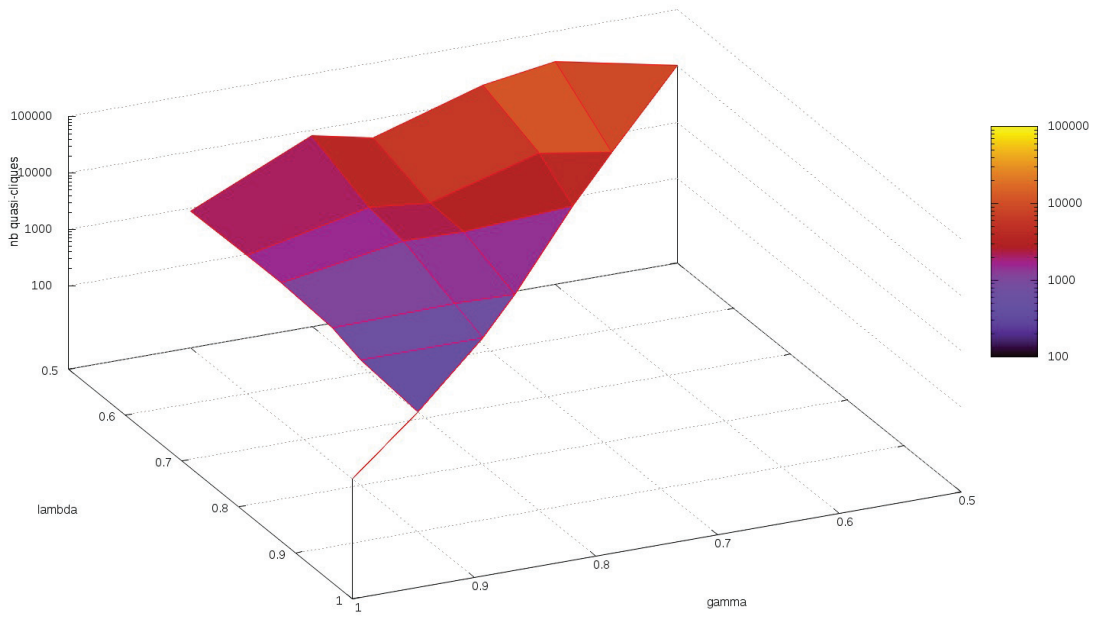


FIGURE 2.54 – Nombre de quasi-cliques énumérées pour différentes valeurs λ et γ pour le graphe *small_100_10_0.8*.

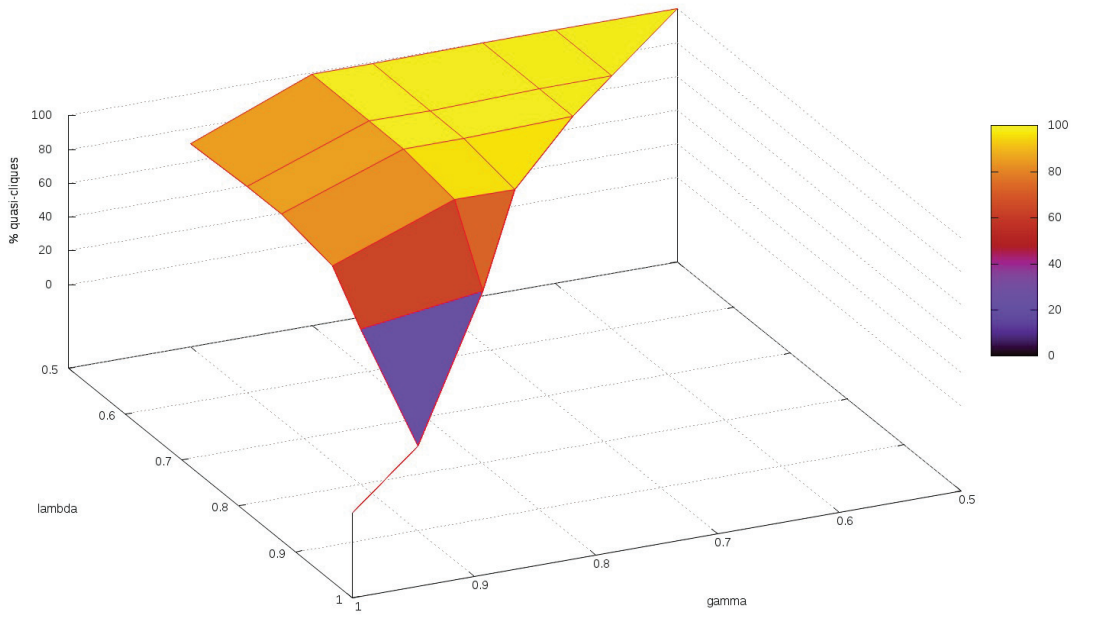


FIGURE 2.55 – Pourcentage de quasi-cliques énumérées pour différentes valeurs λ et γ pour le graphe *small_100_10_0.8*.

Les résultats montrent que le temps est fortement impacté lorsque λ et γ ont des valeurs faibles ($\leq 0{,}66$). Le temps est également dépendant de la classe du graphe : ainsi pour $\lambda_{min} = \gamma_{min} = \frac{1}{2}$, le temps de calcul varie de $3s$ pour le graphe régulier à plus de $16h$ pour le graphe aléatoire.

Le nombre et le pourcentage de quasi-cliques énumérées dépendent également du type de graphe et de la taille de la plus grande clique ω dans le graphe. En effet, quelles que soit les valeurs des paramètres λ et γ, les quasi-cliques ne peuvent être découvertes qu'à partir du moment où les paramètres permettent de découvrir au moins une quasi-clique. Cette observation est valable lorsque le paramètre λ devient inférieur ou égal à $\frac{\omega-1}{\omega}$ et le paramètre γ devient inférieur ou égal à $1 - \frac{2}{\omega\cdot(\omega+1)}$.

Proposition 2.25. Il est possible de découvrir une quasi-clique à partir du moment où $\lambda \leq \frac{\omega-1}{\omega}$ et $\gamma \leq 1 - \frac{2}{\omega\cdot(\omega+1)}$.

Preuve. Soit la taille de la plus grande clique ω du graphe et deux noeuds u et v, u étant inclus dans la clique tandis que v ne l'est pas. La quasi-clique la plus proche d'une clique sera celle à laquelle il ne manque qu'une arête. Parmi les quasi-cliques auxquelles il ne manque qu'une arête, celle qui aura la plus haute valeur de λ et γ sera celle dont l'ensemble la constituant est le plus grand.

Ainsi, hypothétiquement, cette quasi-clique sera composée de la plus grande clique du graphe et du noeud v qui n'est pas connecté à la clique car l'arête (u,v) n'est pas présente dans le graphe.

Ce qui implique, pour le paramètre λ :

$$\lambda = \frac{\omega - 1}{\omega}$$

Et le paramètre γ :

$$\begin{aligned}
\gamma &= \frac{\frac{\omega\cdot(\omega-1)}{2} + \omega - 1}{\frac{(\omega+1)\cdot\omega}{2}} \\
&= \frac{\omega\cdot(\omega-1) + 2\omega - 2}{(\omega+1)\cdot\omega} \\
&= \frac{\omega^2 - \omega + 2\omega - 2}{\omega^2 + \omega} \\
&= \frac{\omega^2 + \omega - 2}{\omega^2 + \omega} \\
&= 1 - \frac{2}{\omega\cdot(\omega+1)}
\end{aligned}$$

□

Nous remarquons que la valeur $\frac{\omega-1}{\omega+1}$ permet de trouver de nombreuses quasi-cliques, les pourcentages étant proches de 100%, les cliques restantes étant celles dont les noeuds la composant sont de faibles degrés.

Nous remarquons également que lorsque la valeur des paramètres devient petite le nombre de quasi-cliques diminue. Ceci s'explique par le fait que les contraintes imposées par les paramètres deviennent faibles et des agglomérats de noeuds sont regroupés dans une même quasi-clique qui est maximale. Dans le cas d'énumération de quasi-cliques avec des paramètres λ et γ nuls, une seule quasi-clique composée de l'ensemble des noeuds du graphe serait trouvée.

2.4.5 Étude de l'intérêt du parallélisme

Notre algorithme et notre heuristique d'énumération de quasi-cliques ont été adaptés à partir de l'algorithme PECO [105]. Cet algorithme distribue les calculs d'énumération de cliques sur plusieurs machines, chaque noeud effectuant des calculs pour déterminer à quelles quasi-cliques il appartient ; ces calculs pouvant être effectués en parallèle s'ils sont distribués sur plusieurs machines.

Nous avons donc souhaité tester notre heuristique sur de plus grands graphes et sur deux clusters de tailles différentes :

— Le premier cluster comporte deux machines ayant des CPU Intel(R) Xeon(R) E3-1245 v5 avec 3.50GHz et 5 workers Spark avec chacun 16G de RAM.
— Le second cluster comporte seize machines ayant des CPU Intel(R) Xeon(R) E3-1245 v5 avec 3.50GHz et 46 workers Spark avec chacun 16G de RAM.

Graphes de test

Nous avons testé l'heuristique sur 4 graphes composés de 100 000 noeuds. Ces graphes sont présentés dans le tableau 2.26.

Nom	$\|E\|$	$\omega(G)$	$\|K\|$
erdos_100000_0.001	5 002 232	4	4 692 753
regular_100000_50	2 500 000	4	2 461 523
power_law_100000_10_0.05	999 866	7	898 594
small_100000_20_0.05	1 049 592	12	149 575

Tableau 2.26 – Présentation de graphes pour tester la distributivité.

Résultats

Les résultats sont présentés dans le tableau 2.27 et la figure 2.56. Nous remarquonse que le fait d'augmenter le nombre de machine diminue le temps. Cependant l'ajout de 8 fois plus de machines ne garanti pas un temps de calcul divisé par 8. Ceci semble s'expliquer par la manière dont est effectué le calcul. En effet, chaque noeud effectue un calcul propre en fonction de son voisinage. Ainsi, avec potentiellement 100 000 machines, chacune aurait à effectuer le calcul d'un noeud, et en conséquence, le temps de calcul serait réduit à celui du temps de calcul pour le plus long des noeuds. Nous pensons donc que c'est le cas dans cette situation, et c'est ce qui expliquerait notamment le peu de temps gagné pour le graphe régulier.

Nom	λ/γ_{min}	$\|Q\|$	*cluster* 1 (s)	*cluster* 2 (s)
erdos_100000_0.001	0,6	12 641 285	8 306	2 288
regular_100000_50	0,6	1 366 923	344	277
power_law_100000_10_0.05	0,75	908 496	1 642	1 267
small_100000_20_0.05	0,84	249 563	7 178	1 738

Tableau 2.27 – Temps d'exécution de l'heuristique sur des clusters de différentes tailles.

Il faudrait confirmer cette idée en mesurant le temps le plus long mis pour un noeud. Si cette idée se confirme, il faudrait donc réfléchir à une répartition initiale différente des candidats par noeuds afin de mieux répartir le calcul.

2.5 Exemples applicatifs

Nous montrons ici l'intérêt de l'énumération des quasi-cliques via l'utilisation de matrices de co-occurrences. Nous utiliserons ces matrices sur un premier exemple théorique suivi d'un second exemple appliqué qui a donné lieu à une collaboration avec l'université de Tucson en Arizona.

2.5.1 Matrices de co-occurrences

Une fois les quasi-cliques énumérées, on peut calculer des matrices de co-occurrences. Le principe est de regarder, pour chaque paire de noeud du graphe, le nombre de quasi-cliques communes dans lesquelles ils

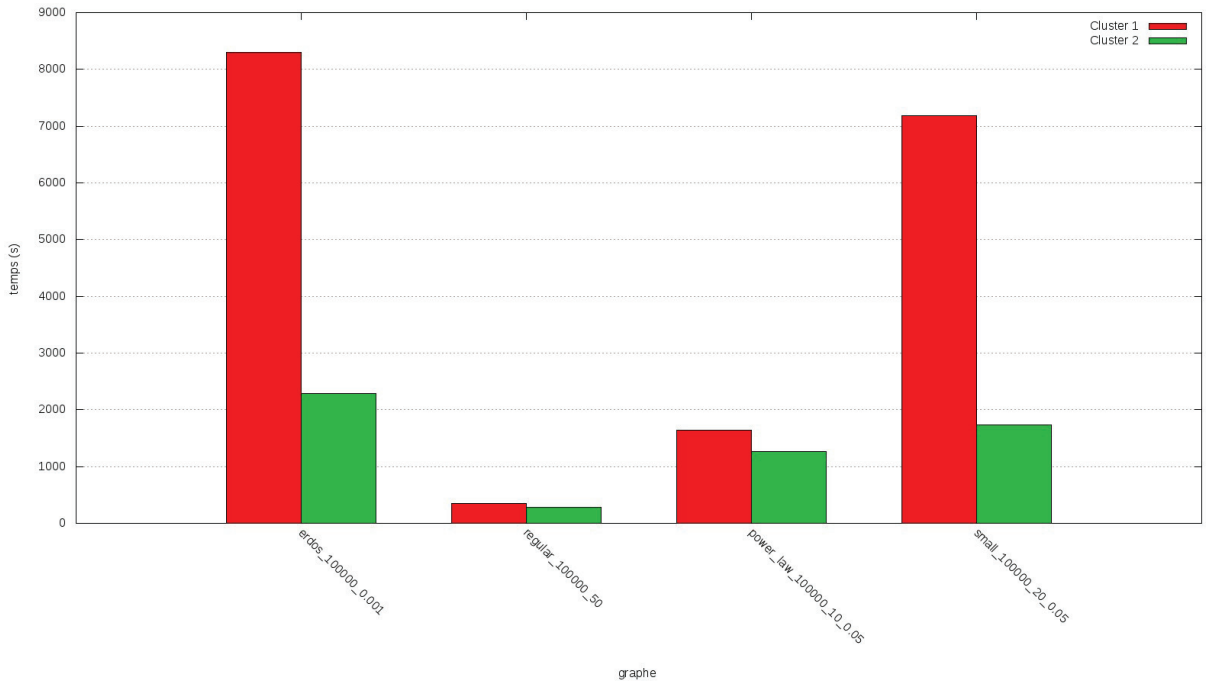


FIGURE 2.56 – Temps d'exécution de l'heuristique sur des clusters de différentes tailles.

sont tous les deux présents. Ensuite ce nombre est relativisé par le nombre de quasi-cliques où un des noeuds apparaît individuellement [4].

L'équation 2.1 présente la méthode du calcul de matrice où :

— Q_G représente l'ensemble des quasi-cliques énumérées pour le graphe G
— Q représente une quasi-clique, $Q \subset Q_G$
— u et v sont deux noeuds du graphe, $u,v \in V$

$$matrice_1[u,v] = \frac{\sum\limits_{\{u,v\} \in Q} 1}{\sum\limits_{u \in Q} 1} \tag{2.1}$$

2.5.2 Matrices pondérées

On peut également souhaiter utiliser la valeur des paramètres des quasi-cliques afin d'utiliser la densité des quasi-cliques dans le calcul du score de la co-occurrence. On utilise alors la valeur du paramètre γ de la quasi-clique.

L'équation 2.2 présente la méthode du calcul de ces matrices pondérées.

$$matrice_2[u,v] = \frac{\sum\limits_{\{u,v\} \in Q} \gamma_Q}{\sum\limits_{u \in Q} \gamma_Q} \tag{2.2}$$

La figure 2.57 présente un graphe sur lequel on va énumérer des quasi-cliques avec $\lambda_{min} = \gamma_{min} = \frac{2}{3}$. Les quasi-cliques énumérées sont présentées dans le tableau 2.28. Nous notons que l'arête présente entre les noeuds 1 et 2 augmente le score du paramètre γ pour la quasi-clique $\{A,B,1,2\}$.

4. par convention on fixera à 0 la co-occurrence d'un noeud avec lui même

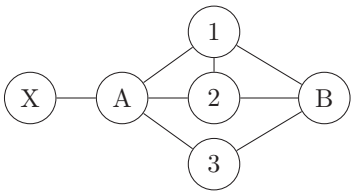


FIGURE 2.57 – Graphe exemple pour l'énumération de quasi-cliques pour le calcul d'une matrice pondérée avec $\lambda_{min} = \gamma_{min} = \frac{2}{3}$.

λ	γ	$quasi - clique$
1,0	1,0	A, X
0,67	0,83	A, B, 1, 2
0,67	0,67	A, B, 1, 3
0,67	0,67	A, B, 2, 3

Tableau 2.28 – Quasi-cliques énumérées pour le calcul d'une matrice pondérée avec $\lambda_{min} = \gamma_{min} = \frac{2}{3}$.

Les matrices de co-occurrences sont présentées dans le tableau 2.29 et le tableau 2.30 pour la matrice pondérée. On remarque ainsi que la matrice pondérée différencie les noeuds 1 et 2 du noeud 3 pour les noeuds A et B, conséquence du lien entre les noeuds 1 et 2 qui augmente la densité de la quasi-clique $\{A,B,1,2\}$. Nous noterons également que les noeuds A et B, bien que n'ayant pas d'arêtes, ont un score très élevé car ils ont de nombreux voisins communs.

	A	B	1	2	3	X
A	0,00	0,75	0,50	0,50	0,50	0,25
B	1,00	0,00	0,67	0,67	0,67	0,00
1	1,00	1,00	0,00	0,50	0,50	0,00
2	1,00	1,00	0,50	0,00	0,50	0,00
3	1,00	1,00	0,50	0,50	0,00	0,00
X	1,00	0,00	0,00	0,00	0,00	0,00

Tableau 2.29 – Matrice de co-occurrences pour les quasi-cliques énumérées.

	A	B	1	2	3	X
A	0,00	0,68	0,47	0,47	0,42	0,32
B	1,00	0,00	0,69	0,69	0,62	0,00
1	1,00	1,00	0,00	0,56	0,44	0,00
2	1,00	1,00	0,56	0,00	0,44	0,00
3	1,00	1,00	0,50	0,50	0,00	0,00
X	1,00	0,00	0,00	0,00	0,00	0,00

Tableau 2.30 – Matrice pondérée de co-occurrences pour les quasi-cliques énumérées.

2.5.3 Cas d'utilisation

Pour démontrer l'intérêt de l'utilisation des quasi-cliques nous avions initialement travaillé sur un graphe représentant un réseau social d'interactions entre kangourous [47]. Ce graphe a été construit par l'observation, sur le terrain, d'interactions entre kangourous [46]. Le graphe est pondéré au niveau des liens, le poids du lien correspondant au nombre d'interractions entre les kangourous. Le graphe initial, que nous nommerons G_1, comprend 17 noeuds représentant les kangourous et 91 arêtes représentant les interactions. Le graphe résultant des observations est présenté en figure 2.58.

L'intérêt d'utiliser des quasi-cliques est ici de proposer une probabilité d'interactions entre deux kangourous. On peut en effet imaginer que les études sur le terrain comportent des biais résultants du fait que des interactions ont pu ne pas être observés.

Pour simuler cette situation nous allons créer un second graphe G_2 à partir du graphe initial dans lequel nous allons retirer l'arête ayant le plus grand poids. L'arête possédant le plus grand poids dans G_1 est celle liant les kangourous 1 et 5.

Nous avons ensuite effectué la recherche de cliques et de quasi-cliques avec $\lambda_{min} = \gamma_{min} = \frac{8}{10}$. Les cliques trouvées sont présentées dans le tableau 2.31, les quasi-cliques trouvées par l'algorithme sont présentées dans le tableau 2.32 et celles par l'heuristique sont présentées dans le tableau 2.33.

Les matrices de co-occurrences produites sont présentées dans le tableau 2.34 pour les cliques, dans les tableaux 2.35 et 2.36 *(matrice pondérée)* pour l'algorithme et les tableaux 2.37 et 2.38 *(matrice pondérée)* pour l'heuristique.

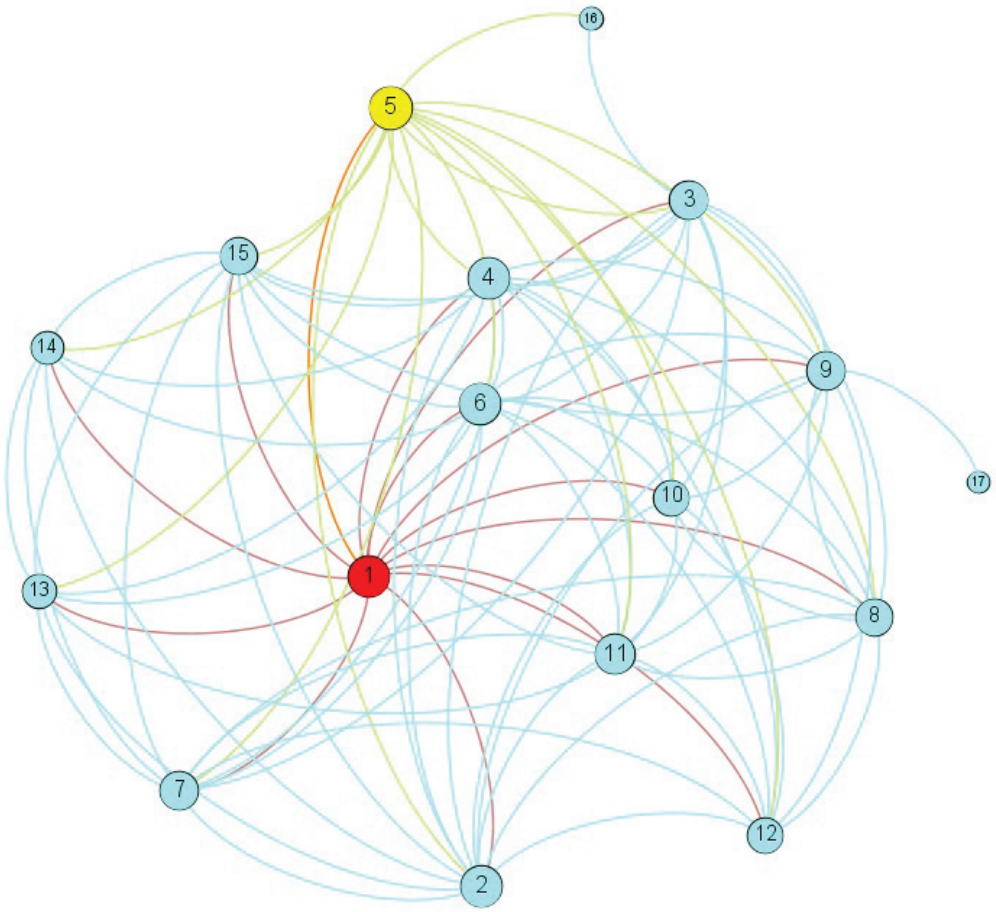


FIGURE 2.58 – Graphe résultant des observations d'interactions entre kangourous.

λ	γ	*clique*
1,0	1,0	{5, 6, 13, 2, 7, 11, 4, 15}
1,0	1,0	{1, 6, 13, 2, 7, 11, 4, 15}
1,0	1,0	{5, 6, 9, 2, 3, 11, 4, 15}
1,0	1,0	{5, 6, 2, 7, 3, 11, 4, 15}
1,0	1,0	{1, 6, 9, 2, 3, 11, 4, 15}
1,0	1,0	{1, 6, 2, 7, 3, 11, 4, 15}
1,0	1,0	{17, 9}
1,0	1,0	{16, 5, 3}
1,0	1,0	{5, 14, 6, 13, 2, 7, 4, 15}

λ	γ	*clique*
1,0	1,0	{14, 1, 6, 13, 2, 7, 4, 15}
1,0	1,0	{5, 6, 2, 7, 3, 11, 8, 4}
1,0	1,0	{1, 6, 2, 7, 3, 11, 8, 4}
1,0	1,0	{5, 6, 2, 12, 7, 11, 8, 4}
1,0	1,0	{1, 6, 2, 12, 7, 11, 8, 4}
1,0	1,0	{5, 10, 6, 9, 2, 12, 11, 8, 4}
1,0	1,0	{5, 10, 6, 9, 2, 3, 11, 8, 4}
1,0	1,0	{10, 1, 6, 9, 2, 12, 11, 8, 4}
1,0	1,0	{10, 1, 6, 9, 2, 3, 11, 8, 4}

Tableau 2.31 – Cliques énumérées sur le graphe G_2.

λ	γ	**quasi-clique**
0,8571428571428571	0,9285714285714286	{5, 10, 1, 6, 13, 2, 11, 4}
0,8	0,9090909090909091	{5, 1, 6, 9, 13, 2, 12, 7, 11, 4, 15}
0,8	0,9272727272727272	{5, 1, 6, 9, 13, 2, 7, 3, 11, 4, 15}
0,8	0,9090909090909091	{5, 1, 6, 9, 13, 2, 7, 11, 8, 4, 15}
0,8	0,9090909090909091	{5, 1, 6, 13, 2, 12, 7, 11, 8, 4, 15}
0,8	0,9272727272727272	{5, 1, 6, 13, 2, 7, 3, 11, 8, 4, 15}
1,0	1,0	{17, 9}
1,0	1,0	{16, 3, 5}
0,8333333333333334	0,9047619047619048	{5, 10, 14, 1, 6, 2, 4}
0,8571428571428571	0,9285714285714286	{5, 14, 1, 6, 9, 2, 4, 15}
0,8	0,9272727272727272	{5, 14, 1, 6, 13, 2, 7, 3, 11, 4, 15}
0,8571428571428571	0,9285714285714286	{5, 14, 1, 6, 2, 12, 7, 4}
0,8571428571428571	0,9285714285714286	{5, 14, 1, 6, 2, 7, 8, 4}
0,8181818181818182	0,9242424242424242	{5, 1, 6, 9, 2, 12, 7, 3, 11, 8, 4, 15}
0,8181818181818182	0,9090909090909091	{5, 10, 1, 6, 9, 2, 12, 7, 3, 11, 4, 15}
0,8181818181818182	0,9393939393939394	{5, 10, 1, 6, 9, 2, 12, 7, 3, 11, 8, 4}
0,8181818181818182	0,9242424242424242	{5, 10, 1, 6, 9, 2, 7, 3, 11, 8, 4, 15}

Tableau 2.32 – Quasi-cliques énumérées par l'algorithme sur le graphe G_2 avec $\lambda_{min} = \gamma_{min} = \frac{8}{10}$.

λ	γ	**quasi-clique**
0,8	0,9272727272727272	{5, 1, 6, 9, 13, 2, 7, 3, 11, 4, 15}
0,8	0,9090909090909091	{5, 1, 6, 9, 13, 2, 12, 7, 11, 4, 15}
0,8	0,9090909090909091	{5, 1, 6, 9, 13, 2, 7, 11, 8, 4, 15}
0,8	0,9272727272727272	{5, 1, 6, 13, 2, 7, 3, 11, 8, 4, 15}
0,8	0,9090909090909091	{5, 1, 6, 13, 2, 12, 7, 11, 8, 4, 15}
1,0	1,0	{17, 9}
1,0	1,0	{16, 5, 3}
0,8	0,9272727272727272	{5, 14, 1, 6, 13, 2, 7, 3, 11, 4, 15}
0,8181818181818182	0,9242424242424242	{5, 1, 6, 9, 2, 12, 7, 3, 11, 8, 4, 15}
0,8181818181818182	0,9393939393939394	{5, 10, 1, 6, 9, 2, 12, 7, 3, 11, 8, 4}
0,8181818181818182	0,9090909090909091	{5, 10, 1, 6, 9, 2, 12, 7, 3, 11, 4, 15}
0,8181818181818182	0,9242424242424242	{5, 10, 1, 6, 9, 2, 7, 3, 11, 8, 4, 15}

Tableau 2.33 – Quasi-cliques énumérées par l'heuristique sur le graphe G_2 avec $\lambda_{min} = \gamma_{min} = \frac{8}{10}$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0,000	1,000	0,500	1,000	0,000	1,000	0,625	0,500	0,375	0,250	0,875	0,250	0,250	0,125	0,500	0,000	0,000
2	0,500	0,000	0,500	1,000	0,500	1,000	0,625	0,500	0,375	0,250	0,875	0,250	0,250	0,125	0,500	0,000	0,000
3	0,444	0,889	0,000	0,889	0,556	0,889	0,444	0,444	0,444	0,222	0,889	0,000	0,000	0,000	0,444	0,111	0,000
4	0,500	1,000	0,500	0,000	0,500	1,000	0,625	0,500	0,375	0,250	0,875	0,250	0,250	0,125	0,500	0,000	0,000
5	0,000	0,889	0,556	0,889	0,000	0,889	0,556	0,444	0,333	0,222	0,778	0,222	0,222	0,111	0,444	0,111	0,000
6	0,500	1,000	0,500	1,000	0,500	0,000	0,625	0,500	0,375	0,250	0,875	0,250	0,250	0,125	0,500	0,000	0,000
7	0,500	1,000	0,400	1,000	0,500	1,000	0,000	0,400	0,000	0,000	0,800	0,200	0,400	0,200	0,600	0,000	0,000
8	0,500	1,000	0,500	1,000	0,500	1,000	0,500	0,000	0,500	0,500	1,000	0,500	0,000	0,000	0,000	0,000	0,000
9	0,429	0,857	0,571	0,857	0,429	0,857	0,000	0,571	0,000	0,571	0,857	0,286	0,000	0,000	0,286	0,000	0,143
10	0,500	1,000	0,500	1,000	0,500	1,000	0,000	1,000	1,000	0,000	1,000	0,500	0,000	0,000	0,000	0,000	0,000
11	0,500	1,000	0,571	1,000	0,500	1,000	0,571	0,571	0,429	0,286	0,000	0,286	0,143	0,000	0,429	0,000	0,000
12	0,500	1,000	0,000	1,000	0,500	1,000	0,500	1,000	0,500	0,500	1,000	0,000	0,000	0,000	0,000	0,000	0,000
13	0,500	1,000	0,000	1,000	0,500	1,000	1,000	0,000	0,000	0,000	0,500	0,000	0,000	0,500	1,000	0,000	0,000
14	0,500	1,000	0,000	1,000	0,500	1,000	1,000	0,000	0,000	0,000	0,000	0,000	1,000	0,000	1,000	0,000	0,000
15	0,500	1,000	0,500	1,000	0,500	1,000	0,750	0,000	0,250	0,000	0,750	0,000	0,500	0,250	0,000	0,000	0,000
16	0,000	0,000	1,000	0,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
17	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

Tableau 2.34 – Matrice des co-occurrences au sein des cliques pour le graphe G_2.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0,000	1,000	0,467	1,000	1,000	1,000	0,800	0,467	0,533	0,333	0,733	0,400	0,467	0,333	0,667	0,000	0,000
2	1,000	0,000	0,467	1,000	1,000	1,000	0,800	0,467	0,533	0,333	0,733	0,400	0,467	0,333	0,667	0,000	0,000
3	0,875	0,875	0,000	0,875	1,000	0,875	0,875	0,500	0,625	0,375	0,875	0,375	0,375	0,125	0,750	0,125	0,000
4	1,000	1,000	0,467	0,000	1,000	1,000	0,800	0,467	0,533	0,333	0,733	0,400	0,467	0,333	0,667	0,000	0,000
5	0,938	0,938	0,500	0,938	0,000	0,938	0,750	0,438	0,500	0,312	0,688	0,375	0,438	0,312	0,625	0,062	0,000
6	1,000	1,000	0,467	1,000	1,000	0,000	0,800	0,467	0,533	0,333	0,733	0,400	0,467	0,333	0,667	0,000	0,000
7	1,000	1,000	0,583	1,000	1,000	1,000	0,000	0,583	0,583	0,250	0,833	0,500	0,500	0,250	0,750	0,000	0,000
8	1,000	1,000	0,571	1,000	1,000	1,000	1,000	0,000	0,571	0,286	0,857	0,429	0,429	0,143	0,714	0,000	0,000
9	0,889	0,889	0,556	0,889	0,889	0,889	0,778	0,444	0,000	0,333	0,778	0,444	0,333	0,111	0,778	0,000	0,111
10	1,000	1,000	0,600	1,000	1,000	1,000	0,600	0,400	0,600	0,000	0,800	0,400	0,200	0,200	0,400	0,000	0,000
11	1,000	1,000	0,636	1,000	1,000	1,000	0,909	0,545	0,636	0,364	0,000	0,455	0,636	0,091	0,818	0,000	0,000
12	1,000	1,000	0,500	1,000	1,000	1,000	1,000	0,500	0,667	0,333	0,833	0,000	0,333	0,167	0,667	0,000	0,000
13	1,000	1,000	0,429	1,000	1,000	1,000	0,857	0,429	0,429	0,143	1,000	0,286	0,000	0,143	0,857	0,000	0,000
14	1,000	1,000	0,200	1,000	1,000	1,000	0,600	0,200	0,200	0,200	0,200	0,200	0,200	0,000	0,400	0,000	0,000
15	1,000	1,000	0,600	1,000	1,000	1,000	0,900	0,500	0,700	0,200	0,900	0,400	0,600	0,200	0,000	0,000	0,000
16	0,000	0,000	1,000	0,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
17	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

Tableau 2.35 – Matrice des co-occurrences au sein des quasi-cliques énumérées par l'algorithme pour le graphe G_2 avec $\lambda_{min} = \gamma_{min} = \frac{8}{10}$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0,000	1,000	0,469	1,000	1,000	1,000	0,800	0,467	0,533	0,333	0,733	0,399	0,466	0,334	0,665	0,000	0,000
2	1,000	0,000	0,469	1,000	1,000	1,000	0,800	0,467	0,533	0,333	0,733	0,399	0,466	0,334	0,665	0,000	0,000
3	0,866	0,866	0,000	0,866	1,000	0,866	0,866	0,497	0,618	0,371	0,866	0,371	0,372	0,124	0,741	0,134	0,000
4	0,974	0,974	0,456	0,000	0,974	0,974	0,779	0,455	0,519	0,324	0,714	0,389	0,454	0,325	0,648	0,000	0,000
5	0,933	0,933	0,504	0,933	0,000	0,933	0,746	0,436	0,497	0,311	0,684	0,372	0,434	0,311	0,620	0,067	0,000
6	1,000	1,000	0,469	1,000	1,000	0,000	0,800	0,467	0,533	0,333	0,733	0,399	0,466	0,334	0,665	0,000	0,000
7	1,000	1,000	0,586	1,000	1,000	1,000	0,000	0,584	0,582	0,251	0,832	0,499	0,498	0,252	0,747	0,000	0,000
8	1,000	1,000	0,575	1,000	1,000	1,000	1,000	0,000	0,572	0,288	0,856	0,429	0,425	0,144	0,711	0,000	0,000
9	0,881	0,881	0,552	0,881	0,881	0,881	0,770	0,442	0,000	0,331	0,770	0,440	0,328	0,111	0,768	0,000	0,119
10	1,000	1,000	0,602	1,000	1,000	1,000	0,602	0,405	0,602	0,000	0,804	0,401	0,202	0,196	0,398	0,000	0,000
11	1,000	1,000	0,639	1,000	1,000	1,000	0,908	0,546	0,636	0,365	0,000	0,453	0,635	0,091	0,816	0,000	0,000
12	1,000	1,000	0,502	1,000	1,000	1,000	1,000	0,502	0,667	0,335	0,832	0,000	0,329	0,168	0,662	0,000	0,000
13	1,000	1,000	0,432	1,000	1,000	1,000	0,856	0,426	0,426	0,144	1,000	0,282	0,000	0,144	0,856	0,000	0,000
14	1,000	1,000	0,201	1,000	1,000	1,000	0,603	0,201	0,201	0,196	0,201	0,201	0,201	0,000	0,402	0,000	0,000
15	0,920	0,920	0,554	0,920	0,920	0,920	0,827	0,459	0,643	0,183	0,827	0,365	0,551	0,186	0,000	0,000	0,000
16	0,000	0,000	1,000	0,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
17	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

Tableau 2.36 – Matrice pondérée des co-occurrences au sein des quasi-cliques énumérées par l'algorithme pour le graphe G_2 avec $\lambda_{min} = \gamma_{min} = \frac{8}{10}$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0,000	1,000	0,700	1,000	1,000	1,000	1,000	0,600	0,700	0,300	1,000	0,500	0,600	0,100	0,900	0,000	0,000
2	1,000	0,000	0,700	1,000	1,000	1,000	1,000	0,600	0,700	0,300	1,000	0,500	0,600	0,100	0,900	0,000	0,000
3	0,875	0,875	0,000	0,875	1,000	0,875	0,875	0,500	0,625	0,375	0,875	0,375	0,375	0,125	0,750	0,125	0,000
4	1,000	1,000	0,700	0,000	1,000	1,000	1,000	0,600	0,700	0,300	1,000	0,500	0,600	0,100	0,900	0,000	0,000
5	0,909	0,909	0,727	0,909	0,000	0,909	0,909	0,545	0,636	0,273	0,909	0,455	0,545	0,091	0,818	0,091	0,000
6	1,000	1,000	0,700	1,000	1,000	0,000	1,000	0,600	0,700	0,300	1,000	0,500	0,600	0,100	0,900	0,000	0,000
7	1,000	1,000	0,700	1,000	1,000	1,000	0,000	0,600	0,700	0,300	1,000	0,500	0,600	0,100	0,900	0,000	0,000
8	1,000	1,000	0,667	1,000	1,000	1,000	1,000	0,000	0,667	0,333	1,000	0,500	0,500	0,000	0,833	0,000	0,000
9	0,875	0,875	0,625	0,875	0,875	0,875	0,875	0,500	0,000	0,375	0,875	0,500	0,375	0,000	0,750	0,000	0,125
10	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,667	1,000	0,000	1,000	0,667	0,000	0,000	0,667	0,000	0,000
11	1,000	1,000	0,700	1,000	1,000	1,000	1,000	0,600	0,700	0,300	0,000	0,500	0,600	0,100	0,900	0,000	0,000
12	1,000	1,000	0,600	1,000	1,000	1,000	1,000	0,600	0,800	0,400	1,000	0,000	0,400	0,000	0,800	0,000	0,000
13	1,000	1,000	0,500	1,000	1,000	1,000	1,000	0,500	0,500	0,000	1,000	0,333	0,000	0,167	1,000	0,000	0,000
14	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,000	0,000	0,000	1,000	0,000	1,000	0,000	1,000	0,000	0,000
15	1,000	1,000	0,667	1,000	1,000	1,000	1,000	0,556	0,667	0,222	1,000	0,444	0,667	0,111	0,000	0,000	0,000
16	0,000	0,000	1,000	0,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
17	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

Tableau 2.37 – Matrice des co-occurrences au sein des quasi-cliques énumérées par l'heuristique pour le graphe G_2 avec $\lambda_{min} = \gamma_{min} = \frac{8}{10}$.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0,000	1,000	0,704	1,000	1,000	1,000	1,000	0,601	0,700	0,301	1,000	0,499	0,598	0,101	0,898	0,000	0,000
2	1,000	0,000	0,704	1,000	1,000	1,000	1,000	0,601	0,700	0,301	1,000	0,499	0,598	0,101	0,898	0,000	0,000
3	0,866	0,866	0,000	0,866	1,000	0,866	0,866	0,497	0,618	0,371	0,866	0,371	0,372	0,124	0,741	0,134	0,000
4	0,993	0,993	0,699	0,000	0,993	0,993	0,993	0,597	0,695	0,299	0,993	0,495	0,595	0,100	0,892	0,000	0,000
5	0,902	0,902	0,733	0,902	0,000	0,902	0,902	0,542	0,631	0,272	0,902	0,450	0,540	0,091	0,810	0,098	0,000
6	1,000	1,000	0,704	1,000	1,000	0,000	1,000	0,601	0,700	0,301	1,000	0,499	0,598	0,101	0,898	0,000	0,000
7	1,000	1,000	0,704	1,000	1,000	1,000	0,000	0,601	0,700	0,301	1,000	0,499	0,598	0,101	0,898	0,000	0,000
8	1,000	1,000	0,671	1,000	1,000	1,000	1,000	0,000	0,668	0,337	1,000	0,501	0,496	0,000	0,830	0,000	0,000
9	0,866	0,866	0,621	0,866	0,866	0,866	0,866	0,497	0,000	0,373	0,866	0,495	0,369	0,000	0,739	0,000	0,134
10	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,672	1,000	0,000	1,000	0,667	0,000	0,000	0,661	0,000	0,000
11	1,000	1,000	0,704	1,000	1,000	1,000	1,000	0,601	0,700	0,301	0,000	0,499	0,598	0,101	0,898	0,000	0,000
12	1,000	1,000	0,604	1,000	1,000	1,000	1,000	0,604	0,802	0,403	1,000	0,000	0,396	0,000	0,795	0,000	0,000
13	1,000	1,000	0,505	1,000	1,000	1,000	1,000	0,498	0,498	0,000	1,000	0,330	0,000	0,168	1,000	0,000	0,000
14	1,000	1,000	1,000	1,000	1,000	1,000	1,000	0,000	0,000	0,000	1,000	0,000	1,000	0,000	1,000	0,000	0,000
15	0,919	0,919	0,615	0,919	0,919	0,919	0,919	0,510	0,611	0,204	0,919	0,406	0,612	0,103	0,000	0,000	0,000
16	0,000	0,000	1,000	0,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000
17	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	1,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000	0,000

Tableau 2.38 – Matrice pondérée des co-occurrences au sein des quasi-cliques énumérées par l'heuristique pour le graphe G_2 avec $\lambda_{min} = \gamma_{min} = \frac{8}{10}$.

On remarque que les quasi-cliques permettent de garder un score élevé pour la co-occurrence des noeuds 1 et 5 là où le score est nul pour les cliques.

Ces scores permettent de rendre compte de l'importance des noeuds relativement les uns aux autres. Ainsi, nous notons que malgré l'arête supprimée dans le graphe G_2, le score de la relation entre les kangourous 1 et 5 reste très élevé, et notamment plus que certaines relations déjà présentes entre des noeuds. Par exemple, on peut prendre la relation entre le noeud 5 et le noeud 16 qui, bien qu'existante dans G_2, a un score très faible dans les matrices de quasi-cliques.

2.5.4 Application aux graphes de Hackers

Dans le cadre d'une collaboration avec l'université de Tucson en Arizona nous avons été amené à générer des graphes et utiliser les quasi-cliques afin d'évaluer les relations entre des hackers sur des canaux IRC. IRC, pour l'anglais *Internet Relay Chat*, est un protocole de communication textuelle accessible sur internet. Il permet la discussion instantanée entre plusieurs utilisateurs sur un même canal de discussion.

La facilité d'utilisation d'IRC en a fait une forme populaire et importante de communication, en particulier parmi les organisations de sécurité, qu'elles soient bienveillantes ou malveillantes, pour partager les connaissances et co-opérer. Certains canaux IRC vont jusqu'à contenir un marché noir [98] où les visiteurs peuvent acheter des connaissances sur des failles de sécurité, des services de piratage informatique, des informations sur des cartes de crédit dérobées, etc. L'analyse des canaux IRC peut aider à comprendre l'esprit et les comportements des cybercriminels ou même à prévoir un éventuel événement de cybercriminalité [97].

La recherche suggère que les pirates se réunissent souvent en organisations via notamment des forums en ligne ou des canaux IRC [10]. Les communautés pirates se spécialisent généralement en fonction des connaissances et capacités de leurs membres ainsi que de leurs intérêts criminels. Ainsi les canaux IRC sont généralement eux aussi spécialisés ; développement de la détection, de l'exploitation ou du partage des vulnérabilités, diffusion des connaissances en piratage ou recrutement de nouveaux membres pour la communauté.

IRC utilise un protocole qui facilite les échanges textuels en temps réels. Par conséquent, contrairement aux forums ou sites web d'échanges, les messages échangés ne sont pas archivés pour être consultés ultérieurement [11]. Ceci implique donc de collecter les messages échangés en temps réels.

Nous distinguerons 4 types de messages différents dans le protocole IRC :

— Les messages *publics*, visibles par l'ensemble des utilisateurs connectés sur le channel IRC
— Les *logs de connexion*, qui sont des messages publics envoyés par le serveur pour signaler la connexion ou déconnexion d'un utilisateur au canal IRC
— Les messages *adressés*, qui sont des messages publics mais dont l'émetteur du message cite directement un utilisateur connecté censé être le destinataire du message. Par exemple si un utilisateur *alpha* pose une question sur le canal, n'importe quel utilisateur *lambda* peut y répondre, mais citera généralement *alpha* pour lui signifier que le message s'adresse premièrement à lui. Un destinataire cité verra alors le message mis en évidence par une technique graphique (généralement la mise en sur-brillance de son pseudo dans le message)
— Les messages *privés*, résultants d'échanges directs entre deux utilisateurs

Un travail de récolte des messages échangés à donc été effectué par le développement de robots se connectant aux canaux IRC et collectant les messages publics (les messages privés n'étant, par définition, pas accessibles). Les messages collectés sont ensuite étudiés par un traitement automatique du langage. Cette étude vise à classifier les messages selon le niveau de dangerosité du message : *normal*, *suspicieux* ou *dangereux*.

Une fois les messages collectés et triés, on peut alors créer des graphes résultants des échanges observés.

Graphes

Pour créer les graphes nous allons, pour chaque canal, considérer les utilisateurs comme noeuds du graphe. Pour créer les arêtes, nous allons tenir une liste d'utilisateurs connectés dans le temps en fonction des *logs de connexion*. Ainsi tout message public envoyé sera considéré comme une arête entre l'émetteur du message et les personnes connectées. On va ainsi pouvoir créer un graphe dirigé, étiqueté en fonction des types de messages échangés et pondéré au niveau des arêtes par le nombre de messages échangés [5].

5. On ajoutera également un poids supplémentaire pour les *messages adressés* d'un utilisateur à un autre

Nous avons créer 5 graphes issus de canaux différents. Ces graphes et leurs principales caractéristiques sont présentés dans le tableau 2.39, les arêtes sont distinguées en fonction des types de messages échangés : normaux (E_0), suspicieux (E_1) ou dangereux (E_2).

graphes	canaux				
	1	2	3	4	5
$\|V\|$	605	712	435	481	528
$\|messages\|$	9 726 448	1 670 016	1 360 963	6 391 941	669 360
$\|E_0\|$	41 475	44 117	23 021	37 940	23 265
$\|E_1\|$	23 035	21 023	11 051	21 868	9 983
$\|E_2\|$	4 033	1 827	1 223	5 085	513
$deg_{moy}(G)$	113,29	94,05	28,86	12,76	65,81
$\Delta(G)$	1 426	1 464	945	1 389	815
diamètre	3	4	4	3	4
coefficient de clustering local	0,88	0,45	0,49	0,70	0,43

Tableau 2.39 – Statistiques des graphes de Hackers issus de différents canaux IRC.

Résultats

Nous avons utilisé l'énumération des quasi-cliques et les co-occurrences au sein de celles-ci pour déterminer une valeur relatant de la relation entre les hackers (et ce malgré le biais lié au fait que l'on ne peut récolter les messages privés). Nous avons donc :

1. filtré les graphes sur les arêtes étiquetés avec le type de message dangereux
2. énuméré les quasi-cliques
3. calculé les co-occurrences au sein des quasi-cliques pour chaque paire de noeuds
4. relevé les interactions suspectes, qui correspondent à un score entre deux utilisateurs de co-occurrences supérieur à 50% alors que ces deux utilisateurs ne sont pas initialement liés dans le graphe

Les résultats de l'énumération des quasi-cliques sont présentés dans le tableau 2.40. Nous y présentons le nombre de cliques et quasi-cliques énumérées en fonction des contraintes λ_{min} et γ_{min}, la valeur moyenne des paramètres pour les quasi-cliques énumérées, la taille de la plus grande quasi-clique énumérée (respectivement de la plus grande clique). Enfin nous donnons le nombre de relations *suspectes*.

graphe	1	2	3	4	5
λ_{min}	0,9	0,7	0,7	0,9	0,7
γ_{min}	0,95	0,7	0,7	0,9	0,7
$\|clique\|$	353	1 022	685	3 078	275
$\|quasi-clique\|$	71 565	2 101 860	1 936 878	130 232	3 758
λ_{mean}	0,912	0,733	0,703	0,928	0,762
γ_{mean}	0,988	0,817	0,778	0,976	0,903
$\|K\|_{max}$	23 (21)	11 (6)	11 (6)	18 (15)	8 (5)
suspectes	0	9	16	0	1

Tableau 2.40 – Résultats de l'énumération des quasi-cliques pour les graphes de Hackers.

Exploitation des résultats

Nous remarquons qu'il est possible de détecter quelques relations suspectes lorsque les paramètres sont bas, ce qui semble logique car on est plus souple sur les quasi-cliques trouvées. Nous notons également que pour ces graphes de données réelles le rapport entre le nombre de quasi-cliques et le nombre de cliques peut varier énormément (de 13 fois plus pour le graphe 5, à 2827 fois plus de quasi-cliques pour le graphe 3).

Notre méthode semble donc permettre de relever des relations suspectes. Afin d'améliorer nos résultats nous envisageons d'utiliser le poids des arêtes pour calculer un nouveau score pour la co-occurrence (reflétant ainsi la densité des interactions entre les utilisateurs).

2.6 Discussion

Notre méthode liste l'ensemble des quasi-cliques maximales sans doublons selon des contraintes de voisinage et de densité. La méthode n'est cependant pas parfaite et ne permet pas de lister des quasi-cliques non-connexes ou qui incluent des cycles supérieurs à 5.

L'algorithme proposé prend cependant un temps considérable, ce qui est dû au calcul des candidats pour l'agrandissement de la quasi-clique.

Nous avons donc développé une heuristique qui améliore grandement le temps de calcul de l'énumération au détriment des quasi-cliques trouvées, l'heuristique pouvant amener du silence dans l'ensemble des quasi-cliques énumérées.

2.7 Conclusion et ouvertures

Le problème de recherche de sous-ensembles denses est un problème très étudié dans la théorie des graphes, notamment lorsqu'elle s'applique aux réseaux. Nous avons proposé une méthode d'énumération de quasi-cliques basée sur l'agrandissement successif d'une quasi-clique courante.

Notre méthode permet de lister l'ensemble des quasi-cliques maximales satisfaisant les contraintes λ et γ, correspondant respectivement à une contrainte de voisinage et une contrainte de densité. Bien qu'exacte notre méthode prend un temps non négligeable. Nous avons donc proposé une heuristique permettant d'arriver plus vite au résultat. Cette heuristique n'est pas exacte et peut générer du silence dans les quasi-cliques énumérées. L'heuristique possède également en contrainte supplémentaire le fait d'avoir un diamètre maximal de 2, ce qui permet d'élaguer certains candidats voisins de la quasi-clique.

Pour l'algorithme, l'expérimentation montre qu'il n'est pas intéressant d'utiliser un pivot :, le gain de temps espéré par l'élagage de l'exploration de certaines solutions ne compense pas le temps nécessaire à énumérer les voisins candidats du pivot ; ces voisins devant être conservés pour lister l'ensemble des quasi-cliques maximales.

Pour l'heuristique, l'expérimentation montre que le fait de ne pas utiliser de pivot permet d'énumérer plus de quasi-cliques, ce qui peut avoir un impact, pour les graphes denses, sur le temps nécessaire à l'énumération de ces quasi-cliques supplémentaires. Dans le cas de graphes de plus faible densité où le nombre de quasi-cliques énumérées sera globalement le même, l'expérimentation montre que le fait de ne pas utiliser de pivot permet là aussi un gain de temps.

Comparé aux autres algorithmes que nous avons implémentés, notre heuristique donne de bons résultats car l'énumération se termine et les quasi-cliques énumérées sont toutes maximales, aucun bruit n'étant produit.

Pour les valeurs des paramètres l'expérimentation montre que l'utilisation de valeurs faibles augmente le temps nécessaire pour énumérer les quasi-cliques alors que le nombre de quasi-cliques diminue. Ceci est une conséquence due au fait que l'on va regrouper plus de noeuds ensemble. Notre expérimentation empirique montre que définir la valeur des paramètres à l'aide de la taille de la clique maximale du graphe offre de bons résultats en termes de quasi-cliques énumérées.

Nous montrons également une possible exploitation des résultats de l'énumération de quasi-cliques pour évaluer les relations entre noeuds du graphe. Nous appliquons cette exploitation de résultats dans le cas concret de l'évaluation de relations entre Hackers.

Il nous semble pertinent de continuer le travail commencé en abordant les points suivants :

— Comparer notre méthode d'évaluation des relations avec celles de recherche de sous-ensembles denses et de prédiction de liens
— Étudier l'impact des paramètres λ et γ sur la qualité des résultats proposés pour l'évaluation des relations entre les noeuds du graphe
— Proposer d'autres méthodes d'élagage conduisant à d'autres heuristiques pour améliorer le temps de calcul ou la qualité des solutions trouvées

— Implémenter et tester l'énumération de quasi-cliques non-connexes
— Tester une répartition initiale des candidats et mesurer les nouveaux temps

Contexte de la problématique : le graphe de ReportLinker

Calcul du graphe des entités nommées de ReportLinker

Des entités nommées

Une entité nommée est un ensemble de mots faisant office de référent pour une entité du monde. Les entités nommées sont généralement associées aux noms propres. Les entités nommées font partie intégrante du traitement automatique du langage naturel.

Les entités nommées représentent des éléments de notre monde considérées comme uniques :

— Des personnes (réelles, fictives, entités vivantes, contemporaines ou historiques). Ex : *Barack Obama, Zeus, Rantanplan*
— Des lieux (endroits localisés géographiquement). Ex : *Lyon, Vénus, Asie*
— Des organisations (sociétés, gouvernements, ONG). Ex : *Apple, Greenpeace, Fond Monétaire International*
— Des entités physiques (produits). Ex : *Hubble, 2CV*
— Des entités théoriques (concepts). Ex : *Réunion du G8, Katrina*

Problématiques liées

Le référencement de ces entités nommées fait apparaître deux principaux problèmes : les situations de polysémie et de métonymie.

Polysémie

La polysémie est une situation rencontrée quand un même mot à plusieurs sens.

Ainsi le terme *Katrina* peut aussi bien référer à une personne qu'à l'ouragan ayant touché les côtes de la Louisiane en 2005.

Métonymie

La métonymie correspond à une figure de style qui utilise un ensemble de mots pour remplacer un autre groupe de mots. Cependant le groupe de mots utilisés reste associé au concept initial par un lien logique.

Ainsi la phrase *« Je vais boire un Beaujolais »* indique que l'on va boire du vin issu du Beaujolais et non pas qu'on va boire l'entité Beaujolais.

Solution

Afin de désambiguïser ces cas de figure et de déterminer à quelle entité nommée l'ensemble de mots sert de référent, une solution est de prendre en considération le contexte autour des mots correspondant à l'entité nommée.

Ainsi les phrases *« Katrina a obligé les habitants de La Nouvelle-Orléans à évacuer la ville »* et *« Katrina a joué aux échecs »* permettent de classer respectivement la première occurrence comme l'ouragan et la deuxième comme une personne.

Les entités nommées sont généralement considérées comme un ensemble ouvert : il est impossible de lister la totalité des entités nommées vu que de nouvelles entités nommées sont créées au cours du temps. Ex : *La découverte de nouvelles exoplanètes au sein du système Trappist-1.*

Présentation du contexte

ReportLinker commercialise un moteur de recherche regroupant des rapports et des études de marché provenant de différentes sources, publiques ou privées. ReportLinker a donc un intérêt particulier pour la détection et le référencement des entités nommées : cela leur permet de définir ce dont parlent les documents, et plus précisément quelles sociétés sont citées dans le document.

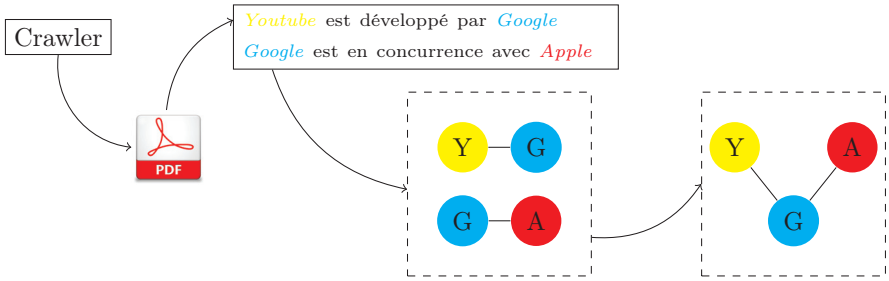
Une solution naturelle est donc de réaliser un graphe des sociétés et de leur environnement.

Construction du graphe des entités nommées

La construction du graphe sociétés de ReportLinker repose sur un enchaînement d'étapes aboutissant à la création d'un graphe d'entités nommées centré sur les sociétés du monde :

1. dans un premier temps des robots, nommés *crawler*, parcourent le web afin de recueillir des données textuelles sur ces sociétés et ce qui les concernent
2. ensuite un traitement automatique du langage est appliqué sur ces données afin de déterminer les entités nommées et leurs contextes (qui sont également des entités nommées)
3. les entités nommées sont ainsi représentées sous la forme de milliards de petits graphes appelés hypothèses
4. ces hypothèses sont ensuite consolidées en un plus grand graphe qui forme le graphe final

La figure présentée en page 129 représente la suite d'étapes aboutissant à la création du graphe de données (des entités nommées).



Le graphe créé est un multi-graphe dirigé, étiqueté et pondéré :

— la pondération est liée au nombre d'occurrence des entités nommées dans les hypothèses, plus une donnée est initialement présente dans les textes, plus les poids des arêtes ou des noeuds seront importants dans le graphe final

— l'étiquette est liée à la typicité des noeuds et des liens : pour certains noeuds on peut déterminer, à l'aide du traitement automatique du langage, si ce sont des entreprises, des lieux, etc.

— la direction s'explique par la typicité des noeuds et des liens : si un *produit* est créé par une *entreprise*, le sens du lien permet de donner une indication supplémentaire. Il y aura en effet un lien *estProduitPar* du *produit* vers l'*entreprise* tandis qu'il y aura un lien *produit* dans le sens opposé

— les arêtes multiples permettent de représenter les différentes interactions entre les noeuds. Par exemple, deux *entreprises* peuvent être en *concurrence* sur un secteur industriel tout en ayant des accords de *collaboration* sur un autre projet. En conséquence il peut exister deux liens entre un noeud u et un noeud v sous la condition que le *type*, représenté par l'étiquette de l'arête $(u,v).type$, soit différent.

Critique du graphe

Le graphe est créé à partir de documents rédigés en anglais et contient des noeuds de différents types, l'ensemble des différents types variant au cours du temps en fonction des besoins de ReportLinker.

La méthode de construction du graphe amène les remarques suivantes :

— Les termes importants au moment de la récupération des données ont un poids plus important dans le graphe car ils ont tendance à être plus cités.

— Des informations peuvent être absentes du graphe, et notamment des relations entre sociétés économiques. Cette situation se rencontre car l'information initiale peut ne pas être suffisamment présente dans les documents utilisés lors de la création des hypothèses.

— La proportion de documents sur le marché économique américain étant importante dans le corpus de documents étudiés, les noeuds ayant des relations avec les États-Unis ont un poids proportionnellement plus important.

On remarque que le graphe n'est pas parfait, il peut notamment y avoir des silences concernant les relations entre noeuds. Le développement de l'algorithme des quasi-cliques a été réalisé dans l'idée de proposer des liens possibles entre sociétés (en utilisant les sous-graphes des sociétés liées à un secteur industriel).

Le principe étant de sélectionner dans le voisinage de noeuds correspondant à un secteur industriel les noeuds de types sociétés et leurs relations entre elles. Sur le sous-graphe ainsi construit, on exécute l'algorithme des quasi-cliques et on peut alors trier les sociétés par co-occurrences communes. Ainsi, si une société x ne possède pas d'arêtes avec une société y mais que x et y possèdent de nombreuses sociétés communes dans leurs voisinages, représentées par l'ensemble S, l'algorithme évaluera la probabilité d'un lien entre x et y selon la densité du sous-ensemble S : plus le graphe induit par le sous-ensemble S sera dense, meilleures seront les quasi-cliques trouvées et meilleur sera le score du potentiel lien entre x et y.

Deuxième partie

Propagation

Chapitre 3

Propagation d'informations dans un graphe d'entités nommées

Résumé

Dans ce chapitre nous étudierons le problème de propagation d'informations au sein d'un graphe économique. Nous présenterons les problèmes et modèles de propagation présents dans la littérature dans la section 3.2. Dans la section 3.3, nous présenterons la problématique liée au problème de propagation dans le graphe économique. Les modèles que nous avons développés pour résoudre le problème seront présentés dans la section 3.4. Nous les testerons ensuite via des expérimentations présentées dans la section 3.5.

3.1 Introduction

La propagation d'informations au sein d'un graphe correspond à l'échange de données entre ses noeuds via ses arêtes. L'idée est que des noeuds possédant une donnée la transmette à leurs voisins qui peuvent faire de même, *propageant* et *diffusant* ainsi l'information dans le réseau.

La figure 3.1 présente un exemple de propagation dans un graphe, les noeuds rouges transmettant leur couleur à leurs voisins au cours du temps.

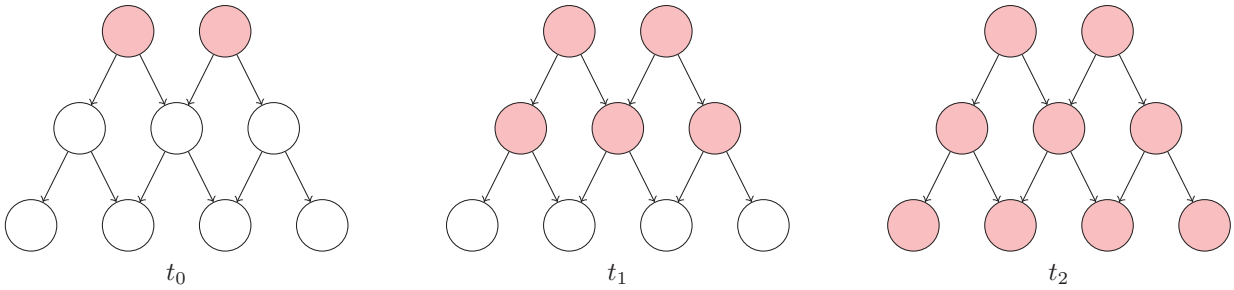


Figure 3.1 – Exemple de propagation dans un graphe.

L'intérêt de modéliser la propagation est d'essayer de comprendre des phénomènes de diffusion tels que le développement d'une maladie ou la transmission de fausses informations.

L'étude de la propagation de ces informations dans des réseaux peut donc porter sur plusieurs cas de figure, ce qui conduit à poser différents problèmes selon les situations rencontrées : variation des temps de propagation, blocage de la diffusion, identification des sources de diffusions, etc.

Les événements économiques et sociaux semblent pouvoir être représentés de manière similaire : une baisse de production de l'orge peut entraîner une augmentation du prix des céréales, ce qui peut impacter les

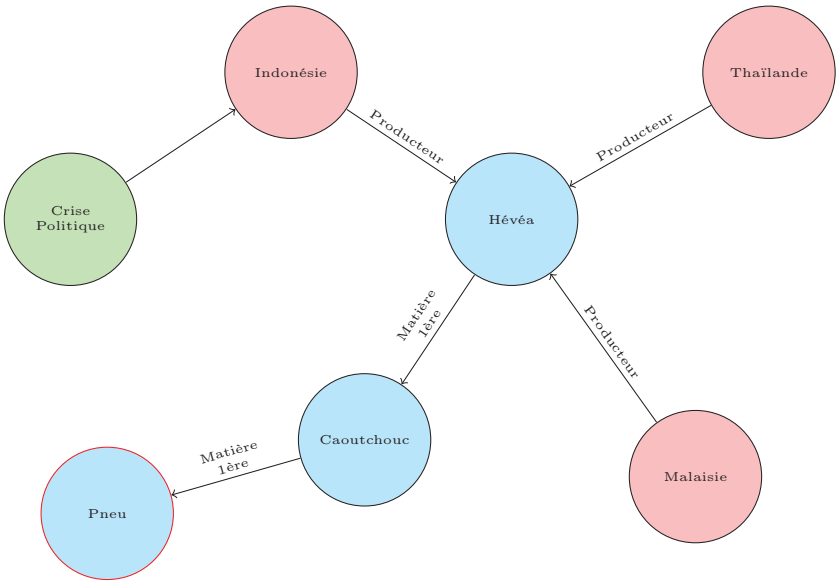


FIGURE 3.2 – Graphe exemple de propagation d'information.

producteurs et distributeurs de bières. Un autre exemple est donné dans la figure 3.2 qui représente l'impact que pourrait donner une crise politique en Indonésie affectant la production d'hévéa et impliquant une crise sur le marché du pneumatique (l'Indonésie étant l'un des principaux producteurs d'hévéa, matière première utilisée pour la fabrication de pneumatiques).

Nous proposons dans ce chapitre de modéliser ce type d'événements sur un graphe hétérogène, c'est-à-dire un multi-graphe dont les noeuds et relations sont étiquetés. Nous appliquerons nos modèles sur le graphe de ReportLinker.

3.2 État de l'art

La propagation d'informations a été étudiée par Kermack et McKendrick [63] dans le cas de diffusion de maladies. Ils proposent un modèle compartimental épidémiologique où l'infection est définie comme un état possible pour un individu, susceptible de passer, selon certaines probabilités d'un état à un autre. Dans leur papier, ils distinguent 3 états possible pour un individu :

— Susceptible : l'individu n'a pas déclaré de maladie
— Infecté : l'individu a été atteint et a contracté la maladie
— Rétabli : l'individu n'est plus malade, ce qui implique soit la mort de l'individu, soit le développement d'une immunité

Certaines études [5, 55] proposent d'autres états pour décrire l'état d'infection d'un individu. Ils différencient notamment l'immunité innée transmise de la mère à l'enfant à celle acquise ou encore un temps d'exposition durant lequel un individu peut être atteint mais ne pas avoir atteint le stade ou la maladie est déclarée.

La figure 3.3 présente une chaîne de Markov représentant les différents échanges possibles entre compartiments où :

— S correspond à l'état susceptible
— I correspond à l'état infecté
— R correspond à l'état retiré, signifiant la mort
— P correspond à l'état protégé, signifiant une immunité acquise
— le nombre sur les arêtes correspond à la probabilité du passage d'un état à un autre

Dans l'exemple donné nous remarquerons qu'il n'est pas possible de passer de certains stades à d'autres. Ainsi, il n'est pas possible de passer du stade infecté à susceptible, l'individu ayant développé une maladie

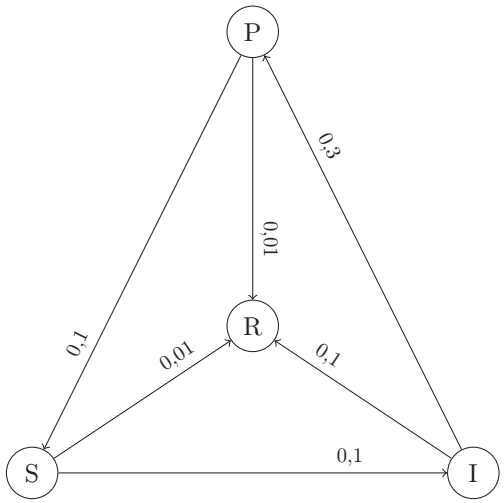


FIGURE 3.3 – Chaîne de Markov représentant les changements d'état possible pour un individu par rapport à une maladie.

étant considéré comme protégé (du moins pendant un certain temps, l'immunité acquise n'étant pas définitive, un individu ayant acquis l'immunité pouvant repasser à l'état susceptible). Il n'est également pas possible de passer de l'état susceptible à l'état protégé, ce qui correspondrait à la vaccination de l'individu.

Dans le cadre de cette thèse nous considérons uniquement l'infection comme une fonction d'étiquetage définissant l'état d'un noeud par rapport à un événement.

Définition 3.1. L'*infection* correspond à une fonction d'étiquetage $I(v)$ de tout noeud $u \in V$.

Nous prendrons en compte seulement les états infectés et susceptibles et la transition de l'état susceptible vers l'état infecté. En effet, dans le cadre d'événements économiques nous considérerons qu'un noeud, une fois en connaissance d'un événement donné, ne peut pas oublier l'information correspondant à un événement. Les noeuds infectés ne pourront donc pas revenir vers l'état susceptible.

Définition 3.2. Un noeud est dit *infecté* lorsqu'il est concerné par un événement.

Définition 3.3. Un noeud est dit *susceptible* lorsqu'il n'est pas concerné par un événement.

En conséquence, un noeud $u \in V$ est soit dans l'ensemble des *Noeuds Infectés* I ou dans l'ensemble des *Noeuds Susceptibles* $\overline{I}$, les ensembles *Infectés* et *Susceptibles* étant disjoints, $I \cap \overline{I} = \emptyset$,$I \cup \overline{I} \equiv V$. À chaque étape de diffusion de l'information, un noeud u est soit infecté, $u \in I$, soit *susceptible* d'être infecté, $u \in \overline{I}$. Un noeud infecté est actif : il est considéré comme contagieux et transmet l'information de l'événement à ses voisins.

L'une des faiblesses des modèles compartimentaux en épidémiologie est de traiter tous les individus de la même manière, sans prendre en compte les comportements locaux des individus. Plusieurs solutions sont proposées pour contourner ce problème : créer des classes d'états pour les individus présentant plus ou moins de risques dans leurs comportements [4] ; prendre en compte les interactions entre individus [87] (en fonction notamment de la répartition du degré des noeuds).

Une autre option est de prendre en compte directement les réseaux et donc les échanges entre individus distincts.

3.2.1 Modèles de propagation dans les réseaux

Les modèles de propagations dans les réseaux reposent généralement sur l'infection du réseau à partir de noeuds sources, qui seraient considérés comme les *patients zéros* dans le cas d'une maladie.

Définition 3.4. Un noeud *source* est un noeud qui est infecté par un événement avant même sa diffusion dans le réseau. Ainsi tout événement e aura un ensemble de noeuds sources S, les noeuds initialement concernés par l'événement.

Il y a deux grandes classes de modèles de propagation, les modèles en cascade et les modèles de seuils.

Le modèle en cascade

Le modèle en cascade [42] repose sur l'idée qu'un noeud infecté $u \in V$ peut aléatoirement infecté un ou plusieurs voisins $v \in \Gamma(u)$ via une fonction probabiliste $p_{(u,v)}$. Si à un instant t un voisin devient infecté, il commence alors lui aussi à potentiellement transmettre l'infection à ses voisins au temps $t+1$:

$$\exists t, u \in I_t, v \in S_t, \forall (u,v) \in E,\ i_{(u,v)t} < p_{(u,v)} \Rightarrow v \in I_{t_{+1}} \tag{3.1}$$

La figure 3.4 représente comment un noeud peut infecter un autre dans le modèle en cascade. La figure présente deux étapes de diffusion : l'étape initial t_0 et l'étape suivante t_1. Chaque arête (u,v) possède une fonction probabiliste p qui représente la probabilité d'infection et qui est représentée par la pondération sur l'arête. Le tirage aléatoire de l'infection $i_{(u,v)}$ au temps initial est représenté par les arêtes pointillées.

Au temps initial t_0, seul le noeud 0, qui est un noeud source, est infecté. Il va donc tenter de transmettre l'infection à ses voisins, les noeuds 1 et 2. En accord avec l'équation 3.1 on remarque que le noeud 1 n'est pas infecté, le tirage aléatoire $i_{(0,1)} = 0{,}4$ étant supérieur à la probabilité d'infection $p_{(0,1)} = 0{,}1$. Cependant, l'infection est transmise au noeud 2, le tirage aléatoire $i_{(0,2)} = 0{,}2$ étant inférieur à la probabilité d'infection $p_{(0,2)} = 0{,}3$. Le noeud 2 devient ainsi infecté au temps t_1 et commence donc à lui aussi effectuer des tirages pour transmettre l'infection.

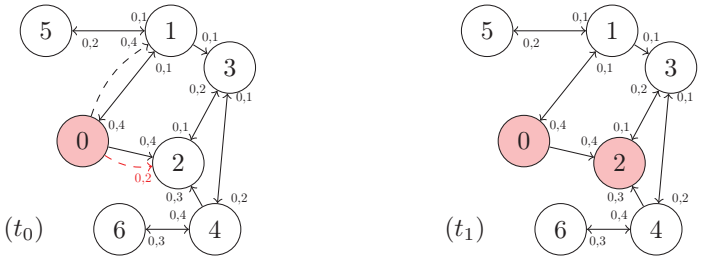


FIGURE 3.4 – Transmission de l'information dans un modèle en cascade.

Le modèle de seuil

Le modèle de seuil [45] repose sur un procédé de percolation.

Définition 3.5. Une *percolation* correspond à une transition d'un état à un autre suite au franchissement d'un seuil.

Ainsi chaque noeud du graphe $u \in V$ se voit associé une fonction seuil $s(u)$ qui lui attribue une valeur numérique. Dans le modèle de seuil, les noeuds infectés $u \in I$ transmettent une information à leur voisin $v \in \Gamma(u)$ sur les arêtes via une pondération $w_{(u,v)}$. Si v n'est pas infecté à une étape t et que la somme du poids des arêtes venant des voisins infectés est supérieure au seuil $s(v)$, v devient infecté :

$$\exists t, v \in S_t, \forall u \in I_t, (u,v) \in E,\ \sum w_{(u,v)} > s(v) \Rightarrow v \in I_{t_{+1}} \tag{3.2}$$

La figure 3.5 représente la propagation de l'information dans un petit réseau. Chaque arête possède une valeur numérique symbolisant le poids de l'information w potentiellement transmise au noeud destinataire si la source de l'arête est infectée. Chaque noeud a sa propre valeur de seuil $s(v)$, présentée dans le tableau 3.1.

Lors de l'étape initiale t_0, seul le noeud 0 est infecté. Selon l'équation 3.2, au temps $t+1$ le noeud 1 devient infecté car il reçoit un poids d'information suffisamment important -$w_{(0,1)} = 0{,}6$) au regard de sa valeur seuil ($s(1) = 0{,}3$). Le noeud 2 n'est quant à lui pas infecté lors de cette étape. Cependant, lors de l'étape t_1, le

noeud 1 va infecté les noeuds 3 et 5. Cette situation va conduire, lors de l'étape t_2, à l'infection du noeud 2. En effet, la somme des informations transmises par ses voisins 0 et 3 ($w_{(0,1)} + w_{(3,1)} = 0{,}2 + 0{,}4 = 0{,}6$) est supérieure à la valeur seuil du noeud 2 ($s(2) = 0{,}5$). La propagation de l'infection s'arrête d'elle-même à l'étape t_3, aucun nouveau noeud n'étant infecté à l'issue de la transmission des informations.

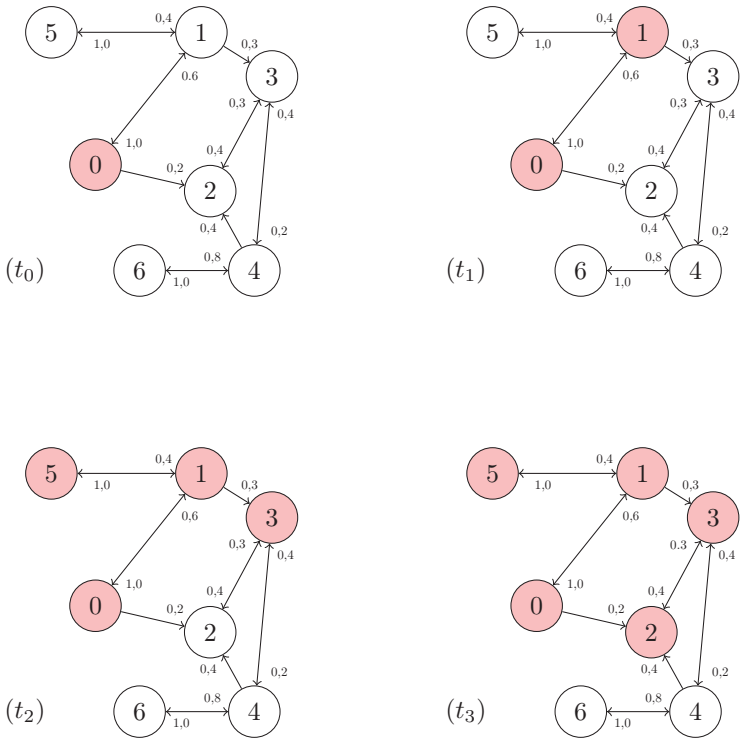


FIGURE 3.5 – Transmission de l'information dans le modèle de seuil.

v	$s(v)$
0	0,4
1	0,3
2	0,5
3	0,2
4	0,6
5	0,3
6	0,2

Tableau 3.1 – Seuils des noeuds.

Généralement le poids des arêtes est calculé aléatoirement en fonction des réseaux. Une autre solution est d'utiliser les poids présents dans les réseaux pondérés pour calculer la propagation [113]. L'étude de la propagation dans de tels réseaux montre que le temps de diffusion dépend du degré des noeuds, du poids des arêtes et des voisins infectés.

La méthode de calcul du seuil est aussi un paramètre important de la propagation. Il peut être fixe [101] ou défini aléatoirement [69, 7]. De nombreuses variations existent, par exemple en fonction des données géographiques [15] ou du degré de voisinage des noeuds dans le réseau [89].

Lu et al. ont démontré que le modèle de seuil est déterministe pour un seuil fixe [76].

Dans le cadre de cette thèse nous cherchons à développer de nouveaux modèles basés sur celui de seuil. En effet, notre objectif premier est d'avoir un modèle qui se comporte toujours de manière identique afin de pouvoir définir quels noeuds sont impactés par un événement donné.

3.2.2 Problèmes de propagation

Parmi les problèmes connus sur l'infection de réseaux, Kempe et al. [60] proposent d'étudier le problème de maximisation de l'influence (en anglais *Influence Maximization Problem*).

Définition 3.6. Le problème de *maximisation de l'influence* consiste à choisir un ensemble V_S de noeuds sources de cardinalité fixe de telle sorte que l'ensemble des noeuds infectés I soit de cardinalité maximale quand le processus de propagation s'arrête.

Le problème d'optimisation est NP-Difficile dans le cas général [60, 61]. Un exemple est montré en figure 3.6. Le nombre de noeuds sources choisi est égal à deux, le modèle est un modèle de seuil dont la valeur est fixée à 0,3 pour chacun des noeuds. Dans l'exemple présenté, avec une cardinalité de l'ensemble des noeuds

sources égale à 2, on voit que les résultats sont sensibles au choix des noeuds sources utilisés pour initialiser le problème : dans le cas de l'exemple de gauche, l'initialisation a lieu sur l'ensemble {4,5} et la propagation s'arrête avec 7 noeuds infectés {1,2,3,4,5,6,8} tandis que si l'initialisation a lieu sur l'ensemble {0,9} la propagation s'arrête avec 5 noeuds infectés {0,1,3,7,9}. Dans ce cas, les résultats sont plus satisfaisants pour l'exemple de gauche.

L'un des exemples applicatifs de ce problème est la publicité, où l'on cherche à faire connaître un produit à un maximum de monde en offrant par exemple ce produit à un nombre limité de personnes initiales, qui se chargeront de sa publicité sur les réseaux sociaux.

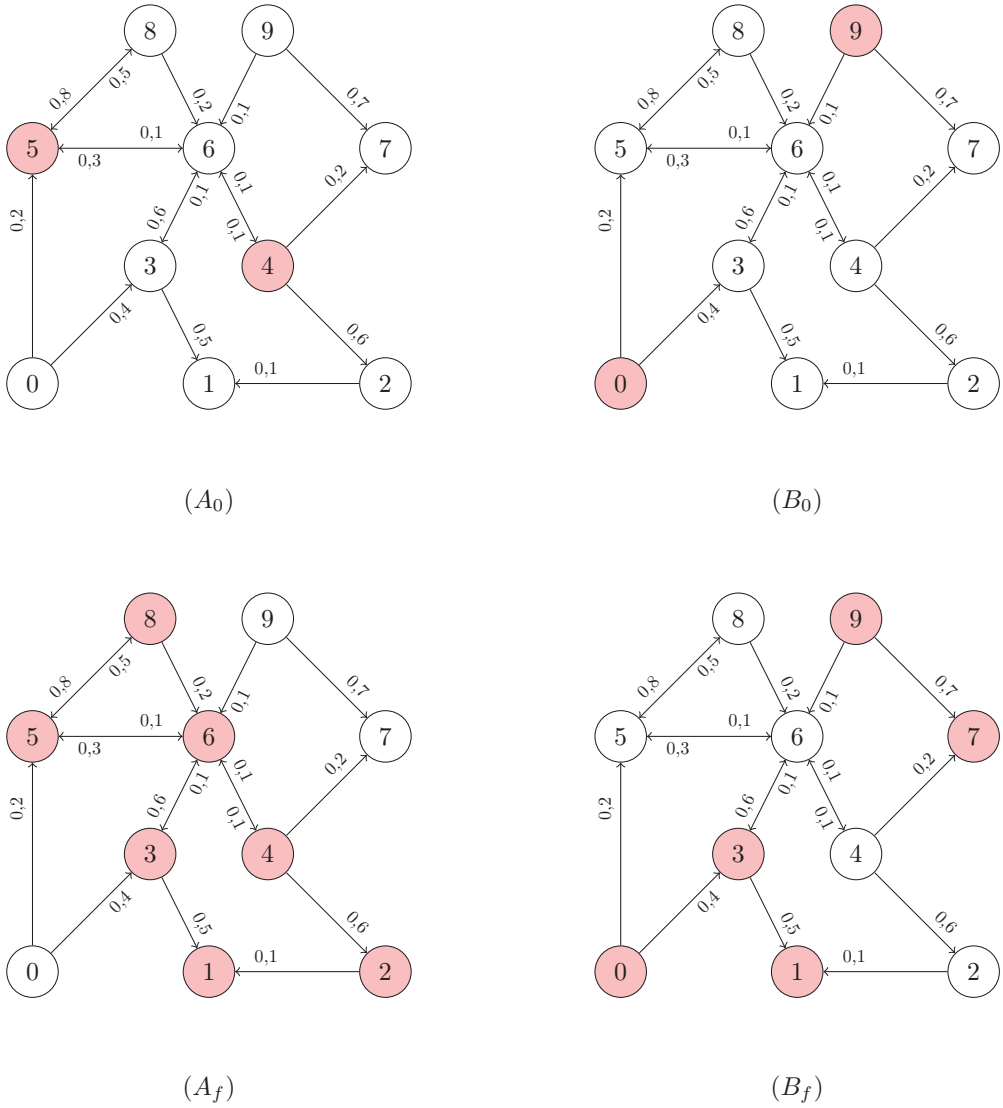


FIGURE 3.6 – Exemple de propagation pour le problème de maximisation de l'influence avec un modèle de seuil où les seuils sont fixés à 0,3 pour tous les noeuds.

Des variantes de ce problème sont également étudiées. Ainsi on peut chercher à minimiser l'influence [118], l'exemple donné étant de celui d'une société qui doit, dans un cas de surcharge, couper l'accès à certains de ses clients. Dans ce cas, elle va chercher à identifier ceux qui auront le moins d'impact sur son image.

Une autre variation très intéressante est l'influence compétitive [13, 44]. Un exemple de l'influence compétitive est celui du vote par référendum. Tout votant ayant le choix de voter oui ou non, les partisans du oui vont s'opposer aux partisans du nom en cherchant à infecter le maximum de votants du réseau. Dans ce

cas de figure Calio et Tagarelli [21] proposent également un état où le votant est infecté mais non actif : il a déjà choisi son vote, mais ne communique pas dessus et n'influence donc pas ses contacts.

Parmi les autres problèmes de informations sur les réseaux on notera également celui d'identification des sources [67] où l'objectif est de déterminer l'ensemble des noeuds sources d'un événement. La résolution de tels problèmes peut par exemple s'appliquer à la détermination des personnes à l'origine de la diffusion d'une fausse informations dans un réseau social.

Nous noterons également, parmi les problèmes d'identification étudiés, ceux qui concernent des noeuds caractéristiques.

On notera notamment le problème des noeuds *influents* [95]. Ce sont des noeuds qui ont un fort impact sur la diffusion car ils ont généralement un grand voisinage et une pondération importante est présente sur leurs arêtes sortantes. Dans le cadre du problème de maximisation de l'influence, ces noeuds ou leurs proches voisins sont par exemple étudier en premier lieu car ils auront un impact certain sur le réseau.

On notera également l'identification de noeuds *critiques* [75], des noeuds par lesquels la diffusion de l'infection à de fortes probabilités de passer. Un des intérêts de l'étude de ces noeuds critiques est d'identifier les personnes à vacciner pour stopper la diffusion d'une maladie. Un exemple est donné en figure 3.7. Dans ce cas précis on voit que seuls les noeuds X et Y connectent les deux communautés. Ainsi, si une maladie se déclare dans la communauté bleue, la vaccination des noeuds X et Y permet d'éviter que la maladie se propage à la communauté rouge.

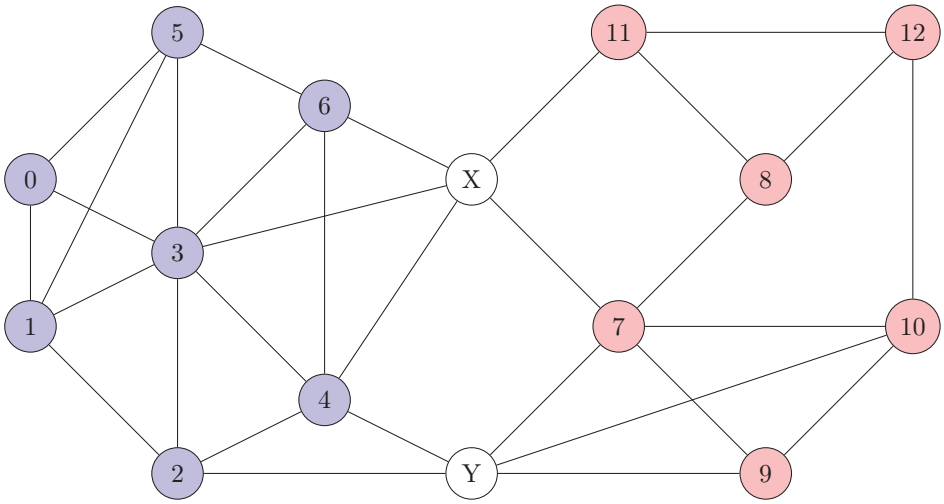


FIGURE 3.7 – Graphe exemple pour l'intérêt de l'identification de noeuds critiques.

L'étude de la propagation dans les réseaux peut aussi viser à évaluer la pondération des liens [43]. En effet, dans des graphes initialement non pondérés, on peut chercher à définir une pondération sur les arêtes en fonction des propagations observées.

Un problème très intéressant est proposé par Mehmood et al. [81], le problème de *cascade typique* (en anglais *Typical Cascade Problem*) : il s'agit de trouver, pour un ensemble de noeuds sources donnés, quels noeuds seront les plus susceptibles d'être infectés dans le cas d'une propagation sur le modèle de cascades. Pour résoudre ce problème, le papier propose d'utiliser plusieurs propagations aléatoires, et de calculer la distance de Jaccard [1] entre chacune d'elles. L'ensemble des noeuds minimisant les distances de Jaccard est considéré comme le résultat de la propagation typique issue des noeuds sources.

La figure 3.8 présente différentes propagations pour un même graphe et un ensemble de noeuds sources $\{X,Y\}$. L'intérêt de l'étude du problème de cascade typique est de faire ressortir les noeuds souvent impactés à partir de l'ensemble initial. Ainsi, l'ensemble de noeuds $\{1,2,3,6\}$, qui est constitué des noeuds présents 2 ou 3 fois dans les propagations figure 3.8, peut être considéré comme un ensemble de noeuds typiquement infectés après une propagation sur le modèle en cascade.

1. La *distance de Jaccard* est une mesure de la différence entre deux ensembles, elle est égale à 0 si les ensembles sont équivalents, 1 si les ensembles sont disjoints.

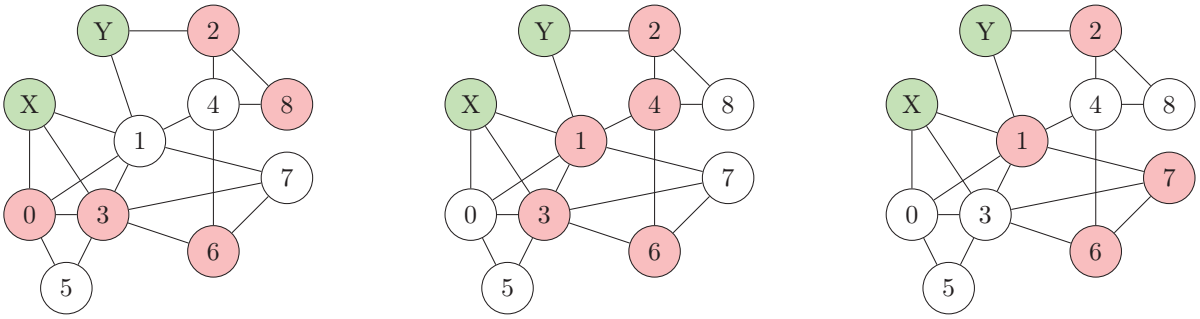


FIGURE 3.8 – Présentation de différentes propagations pour le problème de cascade typique.

Les problèmes de propagation sont étudiés dans les réseaux hétérogènes [72, 73]. Dans ces réseaux les noeuds sont étiquetés, ils possèdent donc un type les classant en catégories qui influencent directement la propagation. Les problèmes étudiés sont généralement des problèmes d'infections ciblées [48] où l'on cherche à atteindre une certaine catégorie de population du graphe. Ce type d'étude peut être rapproché de modèles multi-agents [57], où chaque noeud possède un comportement personnel.

Les problèmes de propagation ont également été étudiés dans les réseaux financiers et notamment dans le milieu bancaire [25, 62]. L'intérêt de l'étude de propagation dans ces réseaux est d'étudier les faillites découlant des différents échanges boursiers [82].

La topologie de ces réseaux est particulièrement étudiée. Ainsi Brusco et Castiglionesi [20] étudient l'interdépendance et la robustesse de deux régions différentes liées par un marché de dépôts interbancaires tandis que Kantemirova et al. [58] proposent une étude de la formation de réseaux économiques.

D'autres graphes économiques ont été construits, par exemple des graphes de sociétés [6] ou d'échanges commerciaux entre pays [37]. Sur ces graphes les noeuds sont de même type (sociétés, pays), et ne sont donc pas hétérogènes dans le sens d'un étiquetage de noeuds.

3.3 Problème de classification de l'infection

Dans le cas de ReportLinker nous avons un multi-graphe dirigé, étiqueté et pondéré qui représente un monde socio-économique. L'idée est d'utiliser un modèle de propagation afin de déterminer, pour un événement donné, quels noeuds sont impactés. On se rapproche donc du problème de *cascades typiques* [81], où le but est de déterminer quels noeuds sont généralement impactés à partir de sources définies. La principale différence vient du fait que l'événement sera lui aussi étiqueté. En effet, deux événements de types distincts, ayant les mêmes sources, doivent potentiellement impacter des noeuds différents.

Par exemple, on pourrait prendre en considération deux événement ayant pour noeuds sources une entreprise et une ville. Le premier événement serait de type *changement de QG* tandis que le second serait de type *fermeture d'usine*. Bien que les noeuds sources soient identiques on ne s'attend pas à ce que les noeuds infectés soit les mêmes.

La figure 3.9 représente cette situation. L'étiquetage des noeuds est défini en fonction de la couleur. Pour l'ensemble des noeuds sources $\{X,Y\}$ on remarque que la propagation observée/souhaitée est différente en fonction du type d'événement. Ainsi l'événement de type *changement de QG* aura un impact plus important sur les lieux affiliés aux noeuds sources tandis que l'événement de type *Fermeture d'usine* aura un impact plus important sur les noeuds économiques liés aux noeuds sources.

Afin de déterminer si les modèles testés proposent de bonnes modélisations de la propagation, dans le graphe de ReportLinker, nous avons défini le problème de *classification de l'infection*.

Définition 3.7. Le *problème de classification de l'infection* consiste à déterminer, pour tout noeud cible $u \in C$, si $u \in I$ pour un événement e.

Ainsi, un événement est défini par un ensemble de noeuds sources et une étiquette que nous nommerons le *type* (de l'événement).

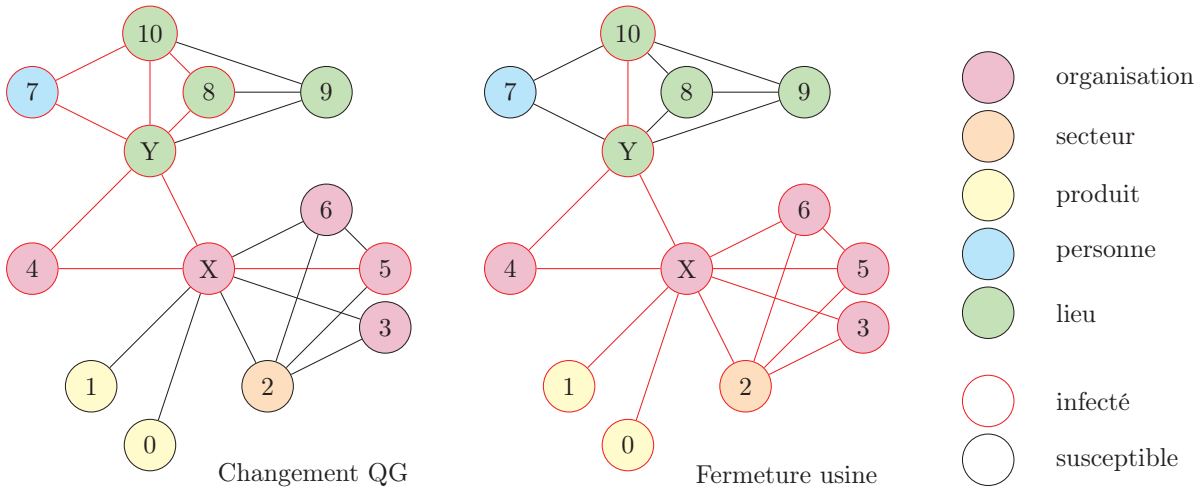


FIGURE 3.9 – Exemple de propagations pour deux événements différents avec des noeuds sources identiques.

Définition 3.8. Un *événement* e dans le *problème de classification de l'infection* est défini par :
— un ensemble $e.S$ ou S qui représente les noeuds sources de l'événement
— une étiquette $e.type$ qui correspond au type de l'événement

Nous définissons également un ensemble de noeuds cibles C. Les noeuds appartenant à cet ensemble serviront de témoins et permettront de dire si le modèle de propagation a infecté les bons noeuds

Définition 3.9. Un noeud *cible* $u \in C$ est un noeud dont on veut que l'état soit à une certaine valeur à la fin de la propagation. Ainsi, l'ensemble des noeuds cibles C est composé de deux sous-ensembles disjoints C_I et $C_{\overline{I}}$ représentant respectivement l'ensemble des noeuds que l'on souhaite voir infectés ou non à la fin de la propagation.

Ainsi, un modèle de propagation devant résoudre le problème de classification de l'infection prend en entrée un événement e et un ensemble C. Il fournit en sortie deux ensembles de noeuds, I_{t_f} et $\overline{I}_{t_f}$, correspondant à l'état des noeuds, infecté ou non, lorsque la propagation s'arrête à l'étape finale t_f.

Nous pouvons alors évaluer la performance de chaque modèle en comparant les ensembles de noeuds retournés par le modèle au résultat attendu en utilisant l'ensemble des noeuds cibles.

On classe ainsi les noeuds en 4 ensembles : vrais positifs, faux positifs, faux négatifs et vrais négatifs.

Définition 3.10. Un noeud n appartient à l'ensemble des *vrais positifs* VP s'il est infecté à la fin de la propagation et qu'il devait être infecté, $n \in C_I, n \in I_{t_f} \Rightarrow n \in VP$.

Définition 3.11. Un noeud n appartient à l'ensemble des *faux positifs* FP s'il est infecté à la fin de la propagation alors qu'il ne devait pas être infecté, $n \in C_{\overline{I}}, n \in I_{t_f} \Rightarrow n \in FP$.

Définition 3.12. Un noeud n appartient à l'ensemble des *faux négatif* FN s'il est n'est pas infecté à la fin de la propagation alors qu'il devait être infecté, $n \in C_I, n \in \overline{I}_{t_f} \Rightarrow n \in FN$.

Définition 3.13. Un noeud n appartient à l'ensemble des *vrai négatif* VN s'il est n'est pas infecté à la fin de la propagation et qu'il ne devait pas être infecté, $n \in C_{\overline{I}}, n \in \overline{I}_{t_f} \Rightarrow n \in VN$.

Ces ensembles nous permettent de calculer deux mesures, la précision et le rappel.

Définition 3.14. La *précision* offre une évaluation du bruit en calculant le pourcentage de noeuds bien classés parmi les noeuds infectés :

$$Précision = \frac{|VP|}{|VP| \cup |FP|} \tag{3.3}$$

La valeur de la précision varie donc de 0 à 1, 0 signifiant qu'aucun noeud devant être infecté ne l'a été tandis que 1 signifie qu'aucun noeud n'a été considéré comme infecté à tort.

Définition 3.15. Le *rappel* offre une évaluation du silence en calculant le pourcentage de noeuds infectés parmi ceux qui sont censés l'être :

$$Rappel = \frac{|VP|}{|VP| \cup |FN|} \tag{3.4}$$

La valeur du rappel varie donc de 0 à 1, 0 signifiant qu'aucun noeud devant être infecté ne l'a été tandis que 1 signifie qu'aucun noeud n'a été considéré comme non-infecté à tort.

La moyenne harmonique de ces mesures, appelée la *F-mesure*, permet d'évaluer la réponse d'un modèle face à une instance du problème de classification de l'infection.

Définition 3.16. La *F-mesure* offre une évaluation du problème de classification de l'infection :

$$F - mesure = 2 \cdot \frac{précision \cdot rappel}{précision + rappel} \tag{3.5}$$

La valeur de la F-mesure varie donc de 0 à 1, 0 signifiant qu'aucun noeud devant être infecté ne l'a été tandis que 1 signifie que l'ensemble des noeuds a été correctement classifié.

La figure 3.10 présente les résultats sur les graphes de deux modèles de diffusion différents, les résultats pour le problème de classification de l'infection sont présentés dans le tableau 3.2. Dans l'exemple proposé, on constate que le modèle 1 donne de meilleurs résultats aussi bien en termes de bruit que de silence. En conséquence c'est ce modèle qui obtient le meilleur résultat, représenté par la valeur de la F-mesure.

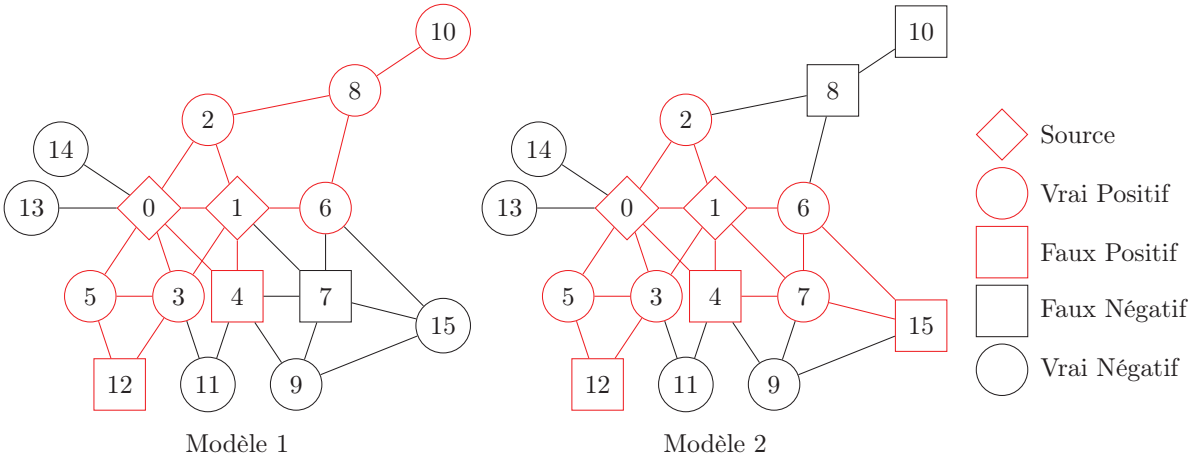


FIGURE 3.10 – Exemple de propagations sur deux modèles différents pour un même événement.

modèles	VP	FP	FN	VN	Précision	Rappel	F-mesure
1	6	2	1	5	0,75	0,86	0,8
2	5	3	2	4	0,63	0,71	0,67

Tableau 3.2 – Résultats de deux modèles pour le problème de classification de l'infection en fonction de la figure 3.10.

3.4 Modèles pour le problème de classification de l'infection

Pour résoudre le problème de classification de l'infection, nous avons commencé par développer des modèles basés sur le modèle de seuil linéaire [45]. Ce choix a été motivé par l'adaptation d'un graphe fixe à un

événement typé par des fonctionnalités permettant de modifier le graphe en fonction des types d'infections. En effet, le graphe de ReportLinker étant déjà pondéré, nous pouvons utiliser une variation des poids, déjà présents sur les arêtes et les noeuds, pour maximiser les résultats obtenus par les modèles de diffusions pour le problème de classification de l'infection.

Le modèle de seuil étant déterministe pour un seuil fixe [76], cela nous garantit des résultats toujours identiques. Ceci nous permet de paramétrer au mieux les fonctionnalités que nous avons développés pour permettre aux modèles d'adapter le graphe au type de l'événement.

3.4.1 Fonctionnalités des modèles

Les fonctionnalités développées dans le cadre de nos modèles sont des fonctions paramétriques permettant d'utiliser les données présentes dans le graphe telles que le poids, le *type* des liens ou des noeuds ou encore la distance aux noeuds sources. Ces fonctionnalités ont été crées afin d'améliorer le score de chaque modèle pour le problème de classification de l'infection.

Calcul du seuil

Afin d'adapter la valeur de seuil de chaque noeud u en fonction de l'événement, nous utilisons le *type* de l'événement $e.type$ et le *type* du noeud $u.type$.

En fonction des paramètres fournis, la fonctionnalité de calcul de seuil produit une pondération $p(e.type,u.type)$ sous la forme d'un pourcentage. Ce pourcentage s'applique alors sur la valeur du seuil de chaque noeud $s(u)$, définissant ainsi un nouveau seuil $s_n(u)$ adapté au type de l'événement. L'équation 3.6 présente la méthode de calcul du nouveau seuil s_n pour le noeud u, pondéré en fonction du type de l'événement e.

$$\forall u \in V, e,\ s_n(u) = p(e.type,u.type) \cdot s(u) \tag{3.6}$$

La fonction de pondération $p(e.type,u.type)$ va chercher la valeur du pourcentage dans une matrice définie. S'il n'existe pas d'entrée pour la valeur des paramètres, ce sont des valeurs seuil par défaut qui sont appliquées. L'un des intérêts du problème de classification de l'information est de définir les bonnes pondérations de seuil en fonction des types d'événements étudiés.

La figure 3.11 propose un exemple de propagation sur laquelle on va utiliser la fonctionnalité de calcul de seuil. En effet, si l'on se place dans le contexte d'un événement e de type *acquisition d'une société*, les noeuds sources de l'événement $e.S$ seront des noeuds de types *organisation*. On va donc chercher à favoriser les noeuds de type *organisation*, car il y a des chances que les organisations devant être impactées dans le cadre du problème de classification de l'infection soient des sociétés ayant un lien avec les sources $e.S$. En contre-partie, les noeuds de types *lieu* et *personne* ont moins de chances d'être concernés par ce type d'événement, on va donc augmenter la valeur seuil de ces types de noeuds. Ce cas de figure est représenté par la matrice présentée dans le tableau 3.3. Les noeuds de types *produit* et *secteur* n'étant pas pris en compte dans la matrice, le modèle prendra la valeur par défaut, soit 100% de la valeur seuil.

La conséquence de ces variations de seuil est présentée dans la figure 3.11. Avant l'utilisation de la pondération du seuil le noeud *IBM* n'est pas atteint alors qu'il doit théoriquement l'être et le noeud *USA* est atteint alors qu'il ne doit pas l'être (partie gauche). L'utilisation de la fonctionnalité de calcul de seuil adapté à l'événement permet de corriger ces problèmes. En effet, l'application de cette fonctionnalité permet de mieux catégoriser l'état final de ces noeuds, augmentant ainsi le score du modèle (pour problème de classification de l'infection).

e.type	*u.type*	pondération (%)
acquisition d'une société	*organisation*	70
acquisition d'une société	*lieu*	120
acquisition d'une société	*personne*	120

Tableau 3.3 – Matrice de pondération pour la fonction de calcul de seuil.

FIGURE 3.11 – Exemple représentant l'influence de la pondération sur le calcul du seuil.

Calcul du poids des arêtes

Afin d'adapter la propagation sur les arêtes (u,v) en fonction de l'événement, nous utiliserons le *type* de l'événement $e.type$, le *type* de l'arête $(u,v).type$ ainsi que les types des noeuds de l'arête ($u.type$ et $v.type$).

En fonction des paramètres fournis, la fonctionnalité d'adaptation du poids des arêtes produit une pondération $p(e.type,u.type,(u,v).type,v.type)$ sous la forme d'un pourcentage. Ce pourcentage s'applique alors sur le poids de l'arête $w_{(u,v)}$, définissant ainsi un nouveau poids $w_{n(u,v)}$ adapté au type de l'événement. L'équation 3.7 présente la méthode de calcul du nouveau poids w_n pour l'arête (u,v) pondéré en fonction du type de l'événement e.

$$\forall u,v \in V,(u,v) \in E,e,\ w_{n(u,v)} = p(e.type,u.type,(u,v).type,v.type) \cdot w_{(u,v)} \tag{3.7}$$

La fonction de pondération $p(e.type,u.type,(u,v).type,v.type)$ va chercher la valeur du pourcentage dans une matrice définie. S'il n'existe pas d'entrée pour la valeur des paramètres, ce sont les poids par défaut qui sont appliqués. L'un des intérêts du problème de classification de l'information est de définir les bonnes pondérations d'arête en fonction des types d'événements étudiés.

La figure 3.12 propose un exemple de propagation sur laquelle on va utiliser la fonctionnalité de calcul du poids de l'arête. En effet, si l'on se place dans le contexte d'un événement e de type *acquisition d'une société*, les noeuds sources de l'événement $e.S$ seront des noeuds de type *organisation*. On va donc chercher à favoriser les relations entre organisations, et plus spécifiquement celles qui sont en relation de *concurrence*. En contre-partie les relations de type *organisation* vers *lieu* ont moins de chance d'être significatives dans le cadre d'un événement de type *acquisition d'une société*. En conséquence, on va diminuer le poids de ces liens. Ce cas de figure est représenté par la matrice du tableau 3.4.

La conséquence de ces variations de seuil est présentée dans la figure 3.12. Avant l'utilisation de la pondération du poids de l'arête, le noeud *IBM* n'est pas atteint alors qu'il doit l'être et le noeud *L.A.* est atteint alors qu'il ne doit pas l'être (partie gauche). En conséquence, l'utilisation de la fonctionnalité de calcul du poids de l'arête adapté à l'événement permet de corriger ces problèmes, augmentant ainsi le score du modèle qui applique cette fonctionnalité au problème de classification de l'infection.

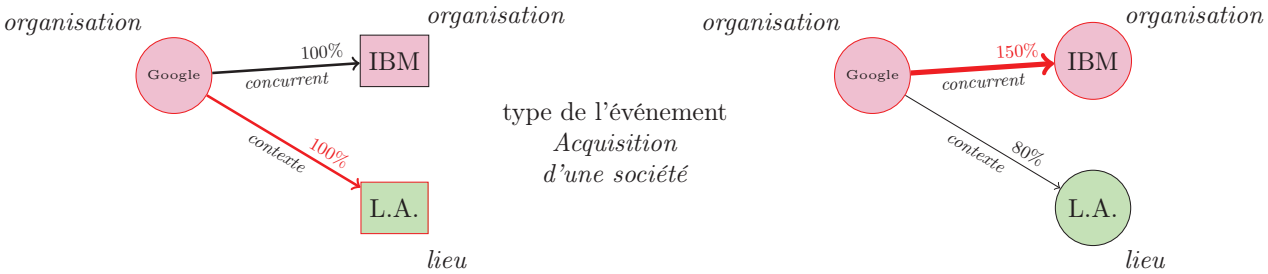


FIGURE 3.12 – Exemple représentant l'influence de la pondération sur le calcul du poids de l'arête.

e.type	*u.type*	*(u,v).type*	*v.type*	pondération (%)
acquisition d'une société	*organisation*	*concurrent*	*organisation*	150
acquisition d'une société	*organisation*	*contexte*	*organisation*	120
acquisition d'une société	*organisation*	*contexte*	*lieu*	80

Tableau 3.4 – Matrice de pondération pour la fonction de calcul du poids de l'arête.

Valeur d'infection

Afin de représenter l'impact de l'événement sur un noeud u nous avons proposé de prendre en considération le rapport entre la valeur des infections entrantes et celle du seuil : plus un noeud u est impacté, plus il impactera les autres noeuds.

Nous définirons l'*exposition* $e(u)$ d'un noeud comme la somme du poids de ses arêtes entrantes dont ses voisins sont infectés :

$$\forall u,v \in V,(v,u) \in E,v \in I_t,\ e(u) = \sum w_{n(v,u)}$$

Ainsi l'*infection* $i_t(u)$ d'un noeud sera définie par le rapport entre l'exposition et le seuil du noeud u au temps t. L'équation 3.8 présente la méthode de calcul de l'infection $i_t(u)$ pour le noeud u au temps t.

$$i_t(u) = \frac{e_t(u)}{s(u)} \tag{3.8}$$

L'infection sera ensuite utilisée comme coefficient sur les arêtes sortantes de u au temps t_{+1}. On peut alors reprendre le calcul de l'exposition, présenté dans l'équation 3.9, pour l'adapter à la valeur de l'infection des chacun des noeuds voisins infectés.

$$\forall u,v \in V,(v,u) \in E,v \in I_t,\ e_t(u) = \sum (w_n(v,u) \cdot i_{t-1}(v)) \tag{3.9}$$

Un noeud sera donc infecté dès lors que sa valeur d'infection sera supérieure à 1. En conséquence, on évaluera l'infection des noeuds sources à 1 lors de la première étape de propagation de l'information.

La figure 3.13 propose un exemple de propagation sur laquelle on va utiliser la prise en compte de la valeur de l'infection. Si initialement (partie gauche) le noeud 2 est infecté il ne ne fait que transmettre l'information qu'il est infecté à son voisin (le noeud 3), il ne l'infecte pas.

L'utilisation de la valeur d'infection change la situation (partie droite). En effet on calcule alors l'infection du noeud 2 via le rapport entre son exposition, égale à 0,4, et celle de son seuil, égale à 0,2. Le noeud 2 est donc bien infecté, avec une valeur d'infection de $\frac{0,4}{0,2} = 2.0$. Par conséquent il transmettra une infection deux fois plus importante sur ses arêtes sortantes. Ainsi, l'arête le liant au noeud 3 aura un poids doublé.

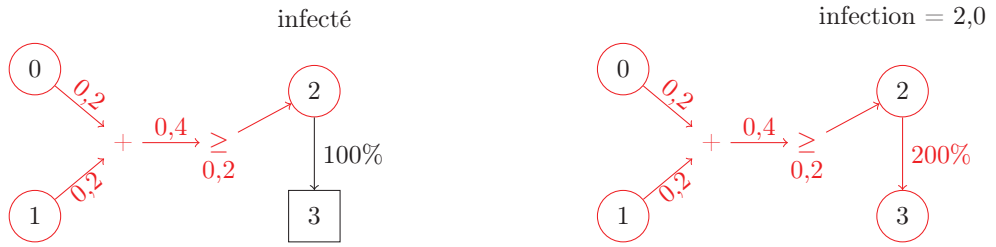


FIGURE 3.13 – Exemple représentant la prise en compte de la fonctionnalité de valeur d'infection.

L'une des conséquences de l'utilisation de la valeur d'infection est qu'une propagation dans un réseau peut être infinie. En effet, dans toute composante fortement connexe infectée du graphe, les noeuds infectés vont voir leur score d'infection continuer à croître.

La figure 3.14 présente un exemple où la propagation est infinie. En effet, si l'on regarde la composante fortement connexe qui contient les noeuds 1, 2 et 3, on remarque que ces noeuds ont une influence les uns sur les autres. En conséquence, dès que le noeud 1 est infecté il transmet l'infection au noeud 2, qui va le transmettre au noeud 3, qui transmettra au noeud 1. Dans le cas d'une propagation sans valeur d'infection, la propagation s'arrêterait à l'étape *(d)*, aucun nouveau noeud n'étant infecté. Dans le cas d'une propagation avec valeur d'infection, la propagation se poursuivra à l'infinie car après l'infection du noeud 3 et transmission de la valeur d'infection au noeud 1, le noeud 1 ré-évaluera sa valeur d'infection. Cette dernière ayant changé, il transmettra sa nouvelle valeur d'infection, ce qui changera celle du noeud 2, qui transmettra au noeud 3 et ainsi de suite.

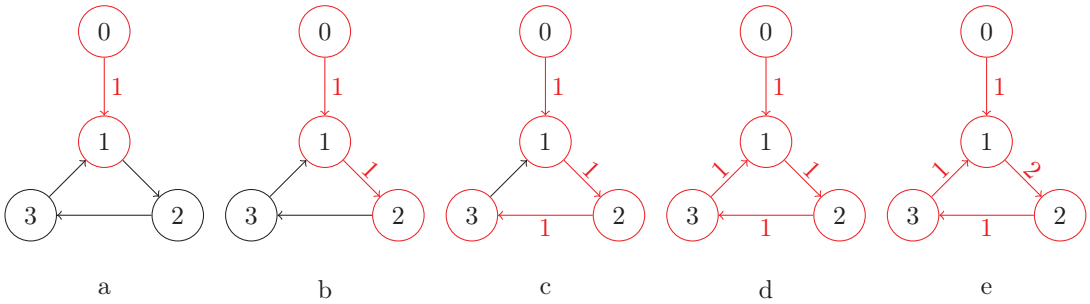


FIGURE 3.14 – Exemple de propagation infinie due à la fonctionnalité de la valeur d'infection.

Pour éviter la situation où une propagation se continue de manière infinie due aux changements de score d'infection des noeuds, plusieurs solutions peuvent être envisagées :

— Stopper la propagation après un certain nombre d'étapes
— Stopper la propagation dès lors qu'il n'y a pas de nouveaux noeuds infectés après une certaine étape
— Ajouter une fonctionnalité d'infection maximale, ce qui permet une terminaison [2]

Fonction d'utilité

Afin de représenter la distance entre l'événement e et un noeud u nous utilisons une fonction d'utilité qui représente la proximité entre l'événement et le noeud. L'idée est de diminuer proportionnellement la valeur d'infection transmise sur les arêtes en fonction de la distance du noeud à l'origine de l'événement.

Nous définirons donc l'utilité d'un noeud v comme une fonction $u(v)$ prenant en compte la distance du noeud v à l'événement e. La distance est représentée par le plus court chemin pcc entre un noeud de l'ensemble des noeuds sources $e.S$ et le noeud v. La fonction d'utilité est donc un fonction mathématique, différente pour chaque modèle, qui renvoie une valeur en fonction de l'entier calculé à partir du plus court chemin $pcc(v,e.s)$. Le calcul de la fonction d'utilité est présenté dans l'équation 3.10.

$$\forall v \in V, u \in e.S, pcc(u,v) \in \mathbb{R},\ u(v) = math(pcc(u,v)) \tag{3.10}$$

Cette fonction d'utilité est ensuite appliquée sur les arêtes sortantes du noeud infecté, changeant l'évaluation de l'exposition du noeud distant. Ce calcul est présenté dans l'équation 3.11.

$$\forall u,v \in V, (v,u) \in E, v \in I_t,\ e_t(u) = \sum \left(\frac{w_n(v,u) \cdot i_{t-1}(v)}{u(v)} \right) \tag{3.11}$$

La figure 3.15 présente un exemple de l'application de la fonction d'utilité. Dans l'exemple, la fonction mathématique choisie est celle de la fonction carré 2^n, appliquée sur la distance à la source moins 1 ($n = pcc - 1$). Ainsi, sans l'utilisation de la fonction d'utilité (partie gauche), tous les noeuds se trouvent infectés. Avec l'utilisation de la fonction d'utilité (partie droite), on remarque que l'exposition des noeuds diminue avec la distance au noeud source 0. Ainsi l'exposition ne change pas pour le noeud 1, $2^0 = 1$, elle est divisée par 2 pour le noeud 2, $2^1 = 2$, et par 4 pour le noeud 3, $2^2 = 4$. Dans ce cas de figure, le noeud 3 ne reçoit plus une exposition suffisante pour être infecté.

2. Dans le pire des cas l'ensemble des noeuds du graphes sont infectés avec la valeur d'infection maximale.

FIGURE 3.15 – Exemple représentant la prise en compte de la fonctionnalité d'utilité.

Le calcul du plus court chemin entre un noeud v et l'ensemble des noeuds sources $e.S$ repose sur la distance minimale avec tout noeud de l'ensemble $e.S$ en passant par un chemin constitué uniquement par des noeuds infectés. Ainsi dans la figure 3.16 *(a)*, la plus courte distance pour le noeud 2 est 1 (le chemin à partir du noeud source 1, passant par le noeud 3, étant plus long). Dans la figure 3.16 *(b)*, la distance la plus courte pour le noeud 4 est de 3, bien que dans le graphe il soit de 2 (cependant le noeud 3 n'étant pas infecté, il ne peut pas être pris en compte dans le calcul du plus court chemin). Si le noeud 3 venait à être infecté plus tard, la valeur de la plus courte distance du noeud 4 changerait alors, ce qui entraînerait une augmentation de l'exposition des noeuds voisins du noeud 4 en accord avec l'équation 3.11.

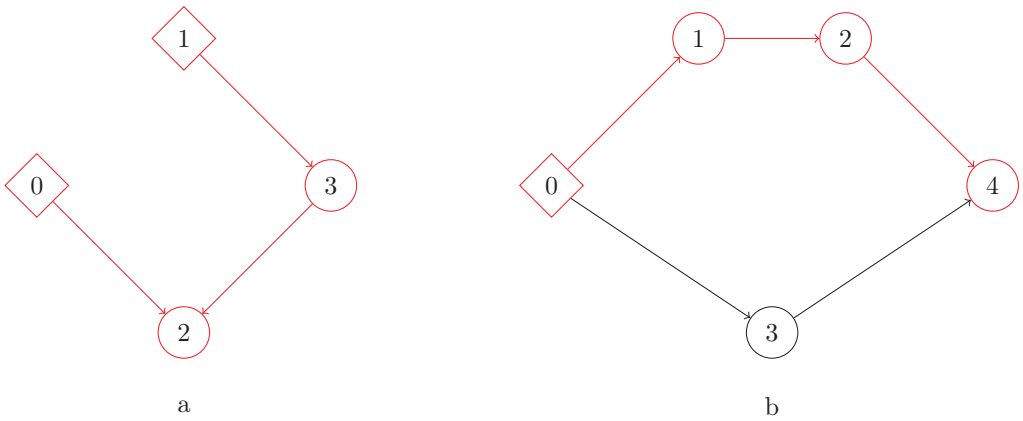


FIGURE 3.16 – Exemples du plus court chemin pour un noeud en fonction de l'ensemble des noeuds sources.

3.4.2 Méthode de diffusion

Pour diffuser l'infection, nous utilisons le modèle Pregel [77]. Le modèle Pregel permet l'échange d'informations entre noeuds du graphe. Un noeud actif envoie des messages à ses voisins. Les messages sont agrégés en un seul message par noeud, ce message étant utilisé pour déterminer quels noeuds seront ensuite actifs. Dans notre cas, les noeuds actifs seront les noeuds infectés. Ainsi, par convention, lors de la première itération, ce seront les noeuds sources de l'événement qui seront actifs.

Le modèle Pregel fonctionne par itérations, chacune d'elles étant divisée en 3 étapes :

1. Les noeuds infectés envoient un message à leurs voisins, correspondant au rapport entre leur infection et la distance à la source
2. Les messages sont agrégés en un seul pour chacun des noeuds, déterminant ainsi l'exposition des noeuds
3. Chaque noeud compare la valeur du message agrégé et celui de son seuil, ce qui permet de définir s'il sera actif dans la prochaine itération

La figure 3.17 présente les étapes du modèle Pregel. Chaque itération passe par chacune des 3 étapes présentées. La première étant celle où les noeuds infectés transmettent à l'ensemble de leur voisin l'infection.

La seconde étape consiste au calcul, pour chacun des noeuds du graphe, de leur exposition en accord avec l'équation 3.11. La troisième et dernière étape de l'itération étant celle de la comparaison entre la valeur d'exposition calculée à l'étape précédente et celle du seuil du noeud. Si la valeur d'exposition est supérieure ou égale à son seuil, le noeud sera infecté à la prochaine itération. Il transmettra donc sa valeur d'infection à l'étape 1. Dans le cas contraire, le noeud ne sera pas infecté et ne transmettra aucune information.

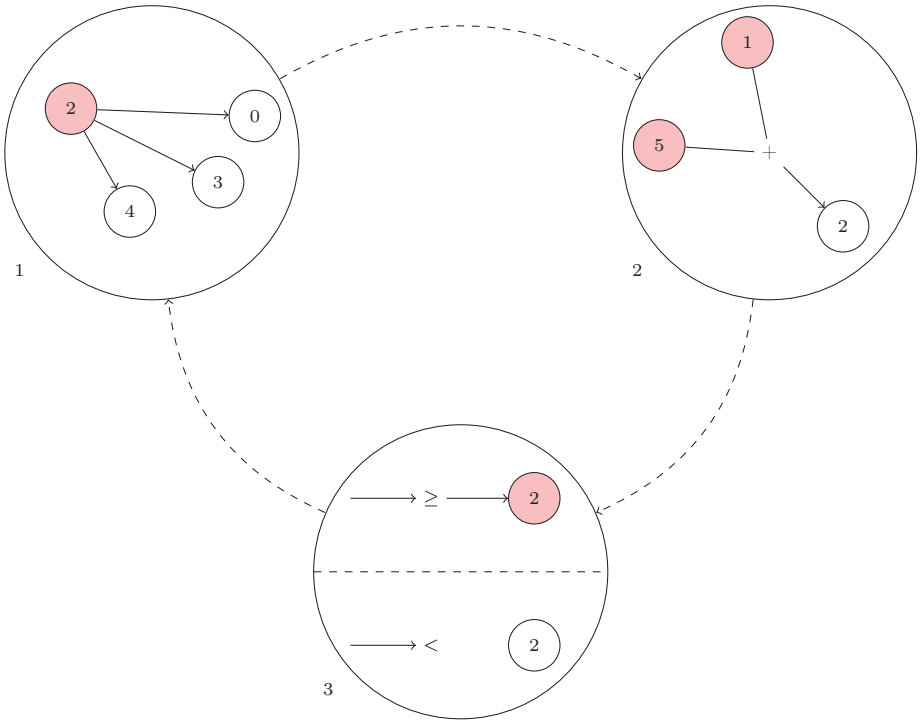


FIGURE 3.17 – Étapes Pregel pour le modèle de seuil

Le modèle Pregel s'arrête lorsque plus aucun noeud n'est actif. Dans notre situation, les noeuds actifs sont ceux qui sont infectés, ce qui implique que le calcul n'est pas censé se terminer. Pour terminer le calcul nous pouvons utiliser soit une limitation du nombre d'étapes, soit une terminaison calculée en fonction des noeuds infectés. Dans le second cas la terminaison est obtenue lorsqu'aucun nouveau noeud n'a été infecté ou alors lorsque les valeurs d'exposition n'ont pas changé entre deux étapes.

3.4.3 Modèle linéaire de seuil hybride

Du fait des caractéristiques intrinsèques au graphe (une pondération étant utilisée aussi bien sur les arêtes que sur les noeuds), nous avons été amené à développer deux nouveaux modèles de seuils qui se distinguent par leur méthode de calcul.

Le premier modèle se nomme le modèle de seuil hybride. Le calcul du seuil dans ce modèle repose sur un pourcentage de la somme totale des liens entrants. Ce pourcentage est appliqué après avoir préalablement changé la valeur du poids des arêtes en accord avec les différents types (voir équation 3.7).

L'équation 3.12 présente le calcul du seuil pour le modèle de seuil hybride.

$$\forall u,v \in V,(u,v) \in E,v \in \Gamma(u),e,\ s(u) = p(e.type,u.type) \cdot \sum w_n(v,u) \tag{3.12}$$

3.4.4 Modèle de seuil adapté

Le second modèle que nous avons développé se nomme le modèle de seuil adapté. Contrairement aux modèles de seuils basés sur les liens entrants, le calcul du seuil dans ce modèle repose sur un pourcentage appliqué au poids du noeuds (le pourcentage étant défini en fonction du type du noeud et de l'événement).

L'équation 3.13 présente le calcul du seuil pour le modèle de seuil adapté.

$$\forall u \in V, e,\ s(u) = p(e.type, u.type) \cdot w(u) \tag{3.13}$$

Étant donné que les valeurs de seuils sont calculées de manière différente pour les noeuds, la fonction de pondération ne doit pas être la même pour les deux modèles.

Ainsi, pour le modèle de seuil hybride, la pondération sera calculée avec un pourcentage relativement faible, représentant la quantité de voisins du noeud devant être infectés pour que le noeud le devienne à son tour. Dans le cas d'un événement neutre sur un noeud de type neutre, la valeur par défaut est fixée à 20% de la somme des poids des liens entrants.

Pour le modèle de seuil adapté, la pondération est différente car elle se rapporte au poids du noeud. Ainsi, dans le cas d'un événement neutre sur un noeud de type neutre, la valeur par défaut est fixée à 100% du poids du noeud.

3.5 Expérimentations

Nous avons souhaité évaluer nos modèles de diffusion sur le graphe de données de ReportLinker. Pour cela nous avons défini plusieurs événements avec des analystes de ReportLinker. Les analystes devaient définir, dans la liste des noeuds cibles, quels noeuds étaient impactés pour un événement donné.

3.5.1 Données

Le graphe de données d'entités nommées de ReportLinker est un multi-graphe dirigé, étiqueté et pondéré composé de 3,8 millions de noeuds et de 9,6 millions d'arêtes. Le graphe utilisé représente une situation économico-sociale en 2016. Les noeuds sont typés, ils peuvent être des *organisations* (IBM, Mitsubishi, etc.), des *lieux* (fleuve Mississippi, Europe, etc.), des *personnes* (Tim Cook, etc.), des *produits* (roundup, playstation, etc.) ou des *secteurs industriels* (banque, agriculture, etc.).

Les caractéristiques principales du graphe sont présentées dans le tableau 3.5. La première ligne présente le nombre de noeuds, d'arcs, le plus grand degré (le noeud avec le plus de voisins) et le nombre moyen de voisins par noeud. La seconde ligne présente le coefficient de clustering, le diamètre et le nombre de noeuds et d'arcs dans la principale composante connexe.

$\|V\|$	$\|E\|$	Deg_{max}	Deg_{moy}
3 770 959	9 609 323	222 879	5.10
cluster. coef.	diameter	$\|V\| \in CC_{max}$	$\|E\| \in CC_{max}$
0.012	15	956 631	7 500 833

Tableau 3.5 – Caractéristiques du graphe de ReportLinker.

Les arêtes possèdent également un type pour représenter les interactions entre deux noeuds. Les arêtes sont dirigées, principalement parce qu'une arête entre deux noeuds de types différents ne peut pas avoir le même type. Considérons une relation entre un produit p et une compagnie c ; l'arête (p,c) sera par exemple de type *est produit par*, tandis que l'arête (c,p) sera dans ce cas de type *fabrique le produit*. Il peut également y avoir plusieurs arêtes entre deux mêmes noeuds, sous condition qu'elles soient d'un type différent : une compagnie c_1 peut avoir des accords de coopération avec une compagnie c_2 pour un premier projet et être, dans le même temps, en compétition avec c_2 sur un autre projet. En conséquence, nous aurions deux arcs de c_1 vers c_2, avec des poids et types respectifs. Dans cette situation, si c_1 est infecté, l'information passera par les deux arcs mais sera pondérée de manière différente selon les propriétés locales des arcs (à savoir leur poids et la pondération effectuée en accord avec la fonctionnalité de pondération des arêtes).

Définition des noeuds cibles

Pour évaluer la qualité des modèles, nous avons défini avec les analystes une liste de 3 792 noeuds considérés comme importants : ce sont nos noeuds cibles dans le problème de classification de l'infection. Ces noeuds ont été définis par l'intersection de deux ensembles :

— un premier ensemble composé des 10 000 noeuds les plus importants en termes de poids de noeuds $w(u)$

— un second ensemble composé des 10 000 noeuds les plus importants en termes de poids de liens entrants $w(v,u)$

Pour chaque événement, les analystes doivent donc définir quels noeuds cibles sont concernés. Ce sont ces noeuds qui permettent d'évaluer les modèles (pour le problème de classification) une fois que la diffusion s'est arrêtée.

3.5.2 Intérêt des fonctionnalités

Nous avons premièrement cherché à évaluer les fonctionnalités que nous avons développées pour les modèles. Pour tester ces fonctionnalités nous avons défini, avec l'aide de deux analystes, deux événements : *Kartina* et *Bayer-Monsanto*

L'ouragan Katrina

Nous avons souhaité simuler les conséquences de l'impact de l'ouragan *Katrina* qui a frappé la *Nouvelle-Orléans* en 2005. Nous avons défini l'événement comme une infection avec :

— un ensemble de noeuds sources $e.S$ composé des noeuds

  — *Nouvelle-Orléans, lieu*

  — *Louisiane, lieu*

— *catastrophe naturelle* comme type d'événement $e.type$

Le rachat Bayer-Monsanto

Nous avons cherché à simuler les conséquences de l'acquisition de *Monsanto* par *Bayer*. Nous avons défini l'événement comme une infection avec :

— un ensemble de noeuds sources $e.S$ composé des noeuds

  — *Bayer, organisation*

  — *Monsanto, organisation*

— *relation économique* comme type d'événement $e.type$

Définition des instances du problème

Le tableau 3.6 présente, pour chacun des événements, le nombre de noeuds cibles désignés par les analystes comme devant être infectés ou non. La somme des deux ensembles de noeuds cibles est égale à 3 790 car nous retirons les noeuds sources de l'ensemble des noeuds cibles pour calculer la qualité de la classification (ce qui est le cas pour les deux événements testés).

Événement	C_I	$C_{\overline{I}}$
Katrina	55	3 735
Bayer-Monsanto	92	3 698

Tableau 3.6 – Classification souhaitée pour les événements.

Nous pouvons voir que l'événement *Katrina* concerne moins de noeuds cibles que *Bayer-Monsanto*. Cette situation s'explique par le graphe utilisé. En effet, le graphe est défini pour représenter des interactions économiques entre des compagnies de différents secteurs industriels. En conséquence, le nombre de noeuds cibles de type *organisation* ou *secteur* est plus important que ceux de type *lieux*.

Présentation des modèles utilisés

Nous avons testé quatre modèles pour résoudre le problème de classification de l'infection :

— **Modèle de seuil linéaire (Lt)** : Basé sur l'article de Granovetter [45], nous avons testé le modèle *Lt*, pour l'anglais *Linear Threshold*. Chaque noeud possède son propre seuil. Le seuil est défini par un pourcentage s'appliquant sur la somme du poids des arêtes entrantes et est le même pour chacun des noeuds indépendamment du *type* de l'événement.

— **Modèle de seuil linéaire dynamique (Dlt)** : Basé sur l'article de Litou et al. [74], nous avons testé le modèle *Dlt*, pour l'anglais *Dynamic Linear Threshold*. Chaque noeud a son propre seuil fixé par un pourcentage sur le poids des arêtes entrantes. *Dlt* propose d'adapter le poids des arêtes dans le temps en exploitant une distribution suivant une Loi de Poisson : *Dlt* permet aux noeuds de renoncer à l'infection et de re-calculer leurs seuils en fonction du temps. Dans notre cas le temps est représenté par le nombre d'itérations écoulées.

— **Modèle linéaire de seuil hybride (Hlt)** : Nous avons testé notre modèle *Hlt*, pour l'anglais *Hybrid Linear Threshold*, basé sur le modèle *Lt*. Dans ce modèle nous recalculons le poids des arêtes et de l'information transmise en fonction du *type* de l'événement, des noeuds et des arêtes. Le seuil est calculé selon un pourcentage sur le poids des arêtes entrantes qui est défini selon le *type* de l'événement et du noeud. Nous proposons plusieurs versions du modèle *Hlt* dans lesquelles nous avons activé ou non différentes fonctions.

— **Modèle de seuil adapté (At)** : Nous avons testé notre second modèle, *At*, pour l'anglais *Adapted Threshold*. Dans ce modèle nous calculons le seuil non pas avec les arêtes entrantes mais avec un pourcentage, défini selon le *type* de l'événement et celui du noeud, qui s'applique au poids initial du noeud. Nous proposons plusieurs versions du modèle *At* dans lesquelles nous avons activé ou non différentes fonctions.

Le tableau 3.5.3 présente les différentes versions des modèles testés pour cette expérience. La colonne seuil indique la manière dont le seuil est défini, fixe correspondant à un seuil relatif identique pour l'ensemble des noeuds, pondéré à un seuil défini en fonction du type des noeuds. Les colonnes *arêtes pondérées*, *score d'infection* et *distance* indique l'utilisation (✓) ou non ( ) de ces fonctionnalités. Les fonctionnalités activées sont définies pour un modèle, c'est-à-dire que les valeurs sont identiques pour un même modèle mais ne le sont pas pour un autre. Par exemple, nous n'utilisons pas la même fonction mathématique pour ré-évaluer le score en fonction de la distance dans les modèles *Hlt* et *At*.

Modèle	seuil	arêtes pondérées	score d'infection	distance
Lt	fixe			
Dlt	fixe			
Hlt_1	fixe	✓		
Hlt_2	pondéré	✓		
Hlt_3	fixe	✓	✓	
Hlt_4	pondéré	✓	✓	
Hlt_5	fixe	✓	✓	✓
Hlt_6	pondéré	✓	✓	✓
At_1	pondéré		✓	
At_2	pondéré		✓	✓
At_3	pondéré	✓	✓	
At_4	pondéré	✓	✓	✓

Tableau 3.7 – Présentation des différents modèles et des fonctionnalités utilisées.

Résultats

Nous avons testé nos modèles avec plusieurs valeurs de paramètres et présentons les meilleurs résultats pour chacun de ces deux événements. Les valeurs ont été définies à la main, en fonction des requêtes des analystes, c'est-à-dire en cherchant à maximiser la valeur de la F-mesure pour le problème de classification.

Le résultat des diffusions de l'information sur les noeuds du graphes est présenté dans les tableaux 3.8 et 3.9. Les colonnes $|I|$ et $|I \cap C|$ donnent respectivement le nombre de noeuds infectés pour le graphe et le

nombre de noeuds cibles infectés. Les colonnes VP, FP et FN représentent le nombre de noeuds correctement classifiés ou non par les modèles.

Nous notons que *Dlt*, avec le changement du poids des arcs en fonction du temps, impacte moins de noeuds que *Lt*.

Pour le modèle *Hlt*, nous remarquons que lorsque le seuil est fixé en fonction de leur type, moins de nœuds sont infectés. Ceci s'explique par le fait que nous pouvons choisir plus facilement les types de noeuds qui doivent être infectés. En conséquence, si le modèle infecte moins de nœuds, la propagation sera moins importante et le nombre d'autres noeuds impactés sera moindre. Il est intéressant de noter que, pour l'événement *Katrina*, la propagation n'a pas lieu si nous n'ajoutons pas la fonctionnalité de score d'infection. Ceci s'explique par le fait que nous utilisons les mêmes valeurs pour le seuil des noeuds pour toutes les propagation avec le modèle *Hlt*. Or, dans le cas de l'événement *Katrina*, le poids initial, qui correspond à la somme du poids des noeuds sources, n'est pas suffisamment important pour permettre une diffusion dans le graphe. Lorsque cette dernière fonctionnalité est ajoutée, l'événement affecte davantage de nœuds, car nous transmettons une valeur plus grande sur les arêtes. Cependant, avec plus de nœuds impactés, nous notons que le bruit augmente également. Pour réduire ce bruit, l'utilisation de la fonctionnalité de distance est utile, car elle diminue les noeuds infectés distants, qui sont généralement moins visés que les voisins directs.

Pour le modèle *At*, nous remarquons qu'adapter le poids de l'arête en fonction du type est utile car cela permet d'avoir un impact uniquement sur certains types de noeuds. Comme pour le modèle *Hlt*, la fonctionnalité de distance permet de réduire le bruit.

Si nous n'utilisons que le meilleur résultat pour chaque modèle, nous remarquons que le modèle *At* impacte une plus grande fraction de noeuds cibles sur l'ensemble des noeuds infectés que les autres modèles. En effet, pour les deux événements, *At* termine la propagation avec moins de noeuds infectés que les autres modèles, mais en proportion, cela a un impact sur de nombreux noeuds cibles. Nous remarquons que le modèle *Hlt* infecte également plus de nœuds importants que le modèle *Lt* duquel il dérive.

Modèle	$\|I\|$	$\|I \cap C\|$	**VP**	**FP**	**FN**
Lt	750	5	4	1	51
Dlt	518	4	4	0	51
Hlt_1	706	6	4	2	51
Hlt_2	37	0	0	0	55
Hlt_3	4 319	190	19	171	36
Hlt_4	1 072	35	9	26	46
Hlt_5	1 194	29	8	21	47
Hlt_6	405	12	7	5	48
At_1	395 688	3 789	55	3 734	0
At_2	10 303	1 966	55	1 911	0
At_3	291 843	3 788	55	3 733	0
At_4	89	39	18	21	37

Tableau 3.8 – Résultats de propagations pour l'événement Katrina.

Les tableaux 3.10 et 3.11 et les figures 3.18 et 3.19 présentent les résultats pour le problème de classification de l'infection pour les événements *Katrina* et *Bayer-Monsanto*. Seule la meilleur version des modèles *Hlt* et *At* est représentée sur les figures. La colonne F-mesure, calculée à partir de la moyenne harmonique de la précision et du rappel, donne une évaluation de chaque modèle.

Les modèles *Lt* et *Dlt* donnent de bons résultats pour la précision, ce qui est dû au fait que ces modèles infectent moins de noeuds importants que les autres. Généralement les noeuds infectés sont les voisins qui sont fortement liés aux noeuds sources. Par conséquent, le bruit est très faible, ce qui est notamment le cas pour le modèle *Dlt* et l'événement Katrina où seulement des noeuds cibles devant être infectés le sont. Ces bons résultats en termes de bruit ont cependant des conséquences sur le silence et la valeur du rappel est donc faible pour les deux modèles.

Pour le modèle *Hlt* nous notons que nous avons de meilleurs résultats pour le silence. L'ajout de la fonctionnalité des arêtes pondérées permet de mieux adapter le graphe au type de l'événement. Nous notons également que l'utilisation du score d'infection permet d'améliorer les résultats (sous condition que nous

Modèle	$\|I\|$	$\|I \cap C\|$	VP	FP	FN
Lt	1 691	25	16	9	76
Dlt	1 244	18	14	4	78
Hlt_1	2 486	43	23	20	69
Hlt_2	1 312	14	11	3	81
Hlt_3	18 619	918	89	829	3
Hlt_4	4 384	188	54	134	38
Hlt_5	6 019	251	60	191	32
Hlt_6	1 693	59	32	27	60
At_1	248 899	3 788	92	3 696	0
At_2	491	121	51	70	41
At_3	148 615	3 767	92	3 675	0
At_4	328	72	50	22	42

Tableau 3.9 – Résultats de propagation pour l'événement Bayer-Monsanto.

utilisons la valeur de seuil pondéré). Pour les deux événements, nous notons que la fonctionnalité de distance permet de réduire le bruit, ce qui augmente par conséquent la précision et donc la valeur de la F-mesure.

Les résultats du modèle *At* montrent que son comportement est opposé à celui des modèles *Lt* et *Dlt*, c'est-à-dire que le silence est plus faible tandis que le bruit est plus haut. Seule l'activation des fonctionnalités de pondération des arêtes et de distance permet d'augmenter la F-mesure pour les deux événements.

Finalement, si nous comparons les meilleurs résultats de chaque modèle, nous notons que le modèle *At*, du fait qu'il favorise l'infection de noeuds cibles, produit de meilleurs résultats pour le problème de classification de l'infection. Nous notons également que les fonctionnalités développées permettent au modèle *Hlt* de surpasser le modèle de seuil linéaire standard.

Modèle	précision	rappel	F-mesure
Lt	0.8	0.07	0.13
Dlt	1.0	0.07	0.14
Hlt_1	0.66	0.07	0.13
Hlt_2	0.0	0.0	0.0
Hlt_3	0.1	0.35	0.16
Hlt_4	0.26	0.16	0.20
Hlt_5	0.28	0.15	0.19
Hlt_6	0.58	0.13	0.21
At_1	0.01	1.0	0.03
At_2	0.03	1.0	0.05
At_3	0.01	1.0	0.03
At_4	0.47	0.33	0.38

Tableau 3.10 – Résultats des modèles pour le problème de classification appliqué à l'événement Kartina.

Nous remarquons également que les résultats sont toujours meilleurs pour l'événement *Bayer-Monsanto* que pour l'événement *Katrina*, ceci s'expliquant par le fait que le graphe traite majoritairement de relations économiques. La partie locale du graphe proche de Bayer et de Monsanto sera en conséquence mieux représentée et détaillée, que ce soit en termes de relations ou de poids.

On remarque notamment le changement de scores entre les deux événements pour les modèles *Lt* et *Dlt*, preuve que les fonctionnalités ne sont pas les seules à avoir de l'influence sur les résultats, mais que la qualité du graphe est aussi à prendre en compte.

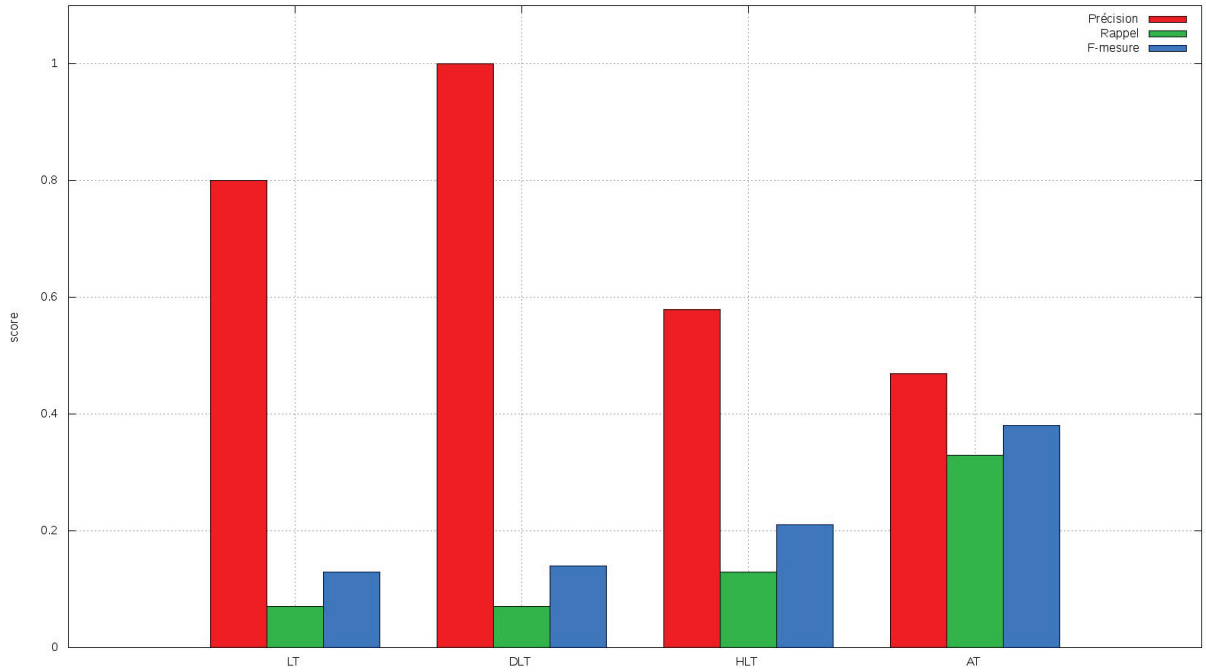


FIGURE 3.18 – Résultats des modèles pour le problème de classification appliqué à l'événement Katrina.

Modèle	précision	rappel	F-mesure
Lt	0.64	0.17	0.27
Dlt	0.78	0.15	0.25
Hlt_1	0.53	0.25	0.34
$HltT_2$	0.79	0.12	0.21
Hlt_3	0.10	0.97	0.18
Hlt_4	0.29	0.59	0.39
Hlt_5	0.24	0.65	0.35
Hlt_6	0.54	0.35	0.42
At_1	0.02	1.0	0.05
At_2	0.42	0.55	0.48
At_3	0.02	1.0	0.05
At_4	0.69	0.54	0.61

Tableau 3.11 – Résultats des modèles pour le problème de classification appliqué à l'événement Bayer-Monsanto.

3.5.3 Paramétrisation des pondérations

Nous avons ensuite tenter de trouver de bonnes pondérations pour les seuils en fonction des types de noeuds et des types d'événements, et ce pour les modèles *Hlt* et *At*. Pour cela, les analystes ont défini des classifications souhaitées pour 10 événements, répartis en deux ensembles en fonction du type des noeuds initiaux. Il y a 3 événements traitant de relations entre *secteur* et *lieux* et 7 événements traitants de relations entre deux *organisations*. Ces événements sont présentés dans le tableau 3.12.

Nous avons ensuite défini des valeurs initiales pour les pondérations de seuils des noeuds en fonction du type d'événement et du noeud. Ensuite nous avons cherché à optimiser la valeur de la F-mesure de l'ensemble des événements en en faisant varier les valeurs des pondérations de seuils. Pour cela nous avons implémenté un recuit simulé [66].

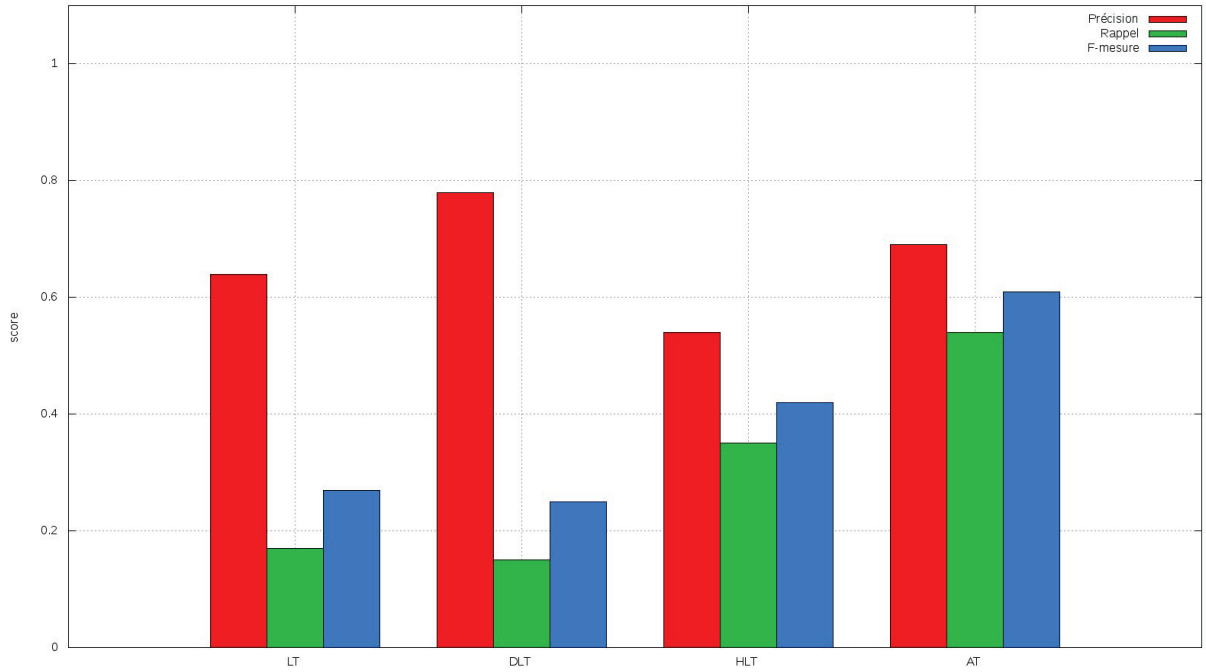


FIGURE 3.19 – Résultats des modèles pour le problème de classification appliqué à l'événement Bayer-Monsanto.

Événement	type	$S \cap C$	C_I	$C_{\overline{I}}$
oilReductionSaudiArabia	lieuSecteur	2	18	3 772
electricCarNorway	lieuSecteur	2	78	3 712
RussiaFirstOnWeapons	lieuSecteur	2	9	3 781
BayerMonsanto	orgaOrga	2	92	3 698
IBMRedHat	orgaOrga	1	17	3 774
CMACeva	orgaOrga	0	10	3 782
commandeAirbusJetBlue	orgaOrga	1	26	3 765
commandeAirbusDeltaAirLines	orgaOrga	1	18	3 773
commandeAirbusEutelsat	orgaOrga	1	13	3 778
chuteSFRAltice	orgaOrga	0	13	3 779

Tableau 3.12 – Classification souhaitée pour les événements.

Nous avons en tout exécuté 200 fois la propagation pour chacun des événements et des modèles *At* et *Hlt*. Nous avons défini une fonction score qui mesurait la moyenne des F-mesure des événements par type. En conséquence, le fonctionnement de l'amélioration de la pondération pour les seuils des noeuds est le suivant :

1. on change aléatoirement la valeur d'un seuil, en gardant sa valeur proche de ce qu'elle était avant
2. on teste chacun des événements avec les nouvelles valeurs de seuils
3. on calcule la nouvelle moyenne de la F-mesure que l'on compare à la valeur précédente :
   — si la nouvelle moyenne est meilleure que la précédente on garde la nouvelle valeur de seuil
   — si la nouvelle moyenne est inférieure à un certain seuil (qui devient de plus en plus petit en fonction du temps), on garde également la nouvelle valeur, pour éviter de tomber dans un maximum local
   — sinon on annule le dernier changement de la valeur de seuil

Finalement on garde l'ensemble de pondérations qui a apporté les meilleurs résultats.

Les évolutions du score de la F-mesure pour les événements de type *lieuSecteur* et *orgaOrga* sont présentés respectivement dans les figures 3.20 et 3.21. Les valeurs des modèles *Lt* et *Dlt* sont représentées par les droites constantes. On remarque que dans les deux cas le modèle standard *Lt* surpasse le modèle *Dlt*. Nous notons également que les scores sont biens meilleurs pour les événements de relations entre deux *organisations*, ce qui semble confirmer le fait que le graphe est plus précis pour les relations économiques.

On remarque enfin que les modèles *Hlt* et *At* donnent globalement de bons résultats mais peuvent très rapidement subir de grandes variations. En effet, pour les deux catégories d'événements, on se retrouve plusieurs fois avec des chutes importantes de la valeur de la F-mesure en une seule itération. Ceci peut indiquer que nos méthodes de calculs de seuils voisins sont peut être trop importants et/ou que certains types de noeuds ont des impacts importants sur la qualité des résultats fournis par les modèles (en fonction des catégories d'événements donnés). Nous notons également que le modèle *At* semble subir plus fortement ces changements, les pertes de valeurs enregistrées étant plus importantes que pour le modèle *Hlt*.

Les meilleurs résultats des deux modèles sont équivalents pour la F-mesure, ce qui semble indiquer que la fonctionnalité de pondération de seuil permet d'obtenir de meilleurs résultats indépendamment de la manière dont le seuil est calculé.

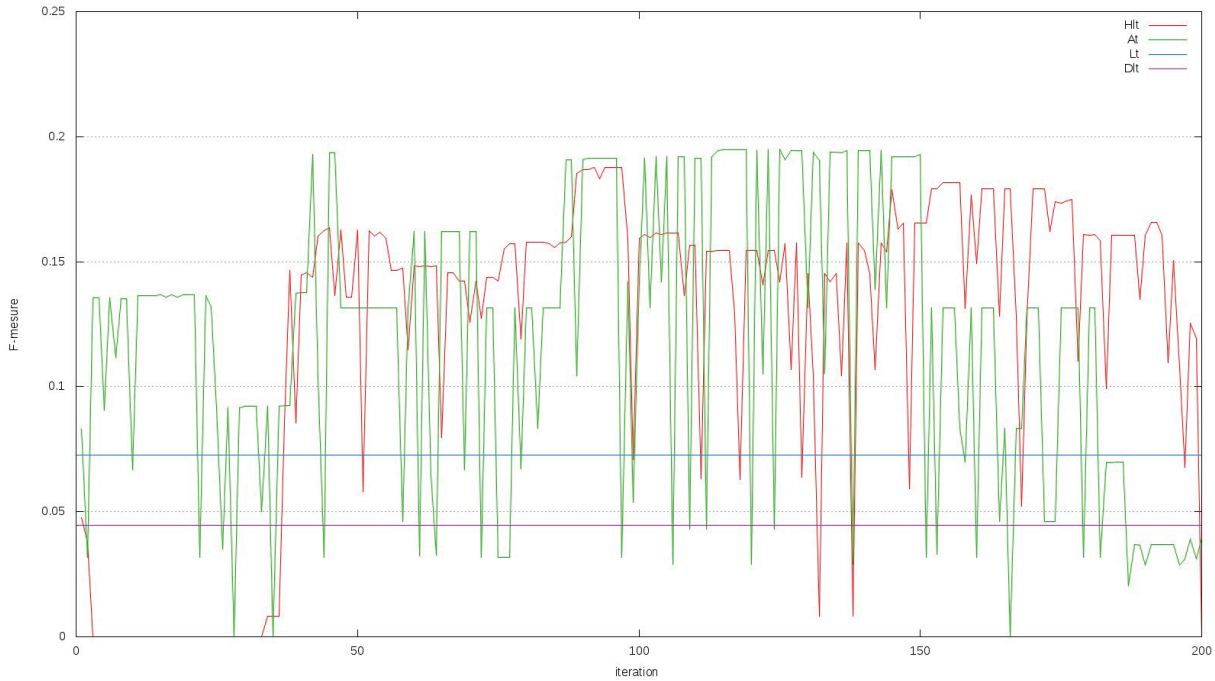


FIGURE 3.20 – Évolution de la F-mesure en fonction des itérations pour les événements de type *lieuSecteur*.

Les résultats détaillés sont présentés dans le tableau 3.13, avec les meilleurs scores obtenus pour l'ensemble des modèles représentés en bleu. Pour les modèles *Hlt* et *At*, le résultat présenté est celui obtenu lorsque la moyenne pour l'ensemble des événements de même type est maximale, bien qu'il soit possible que des scores individuels supérieurs aient été observés dans certaines configurations.

Pour les événements de type *lieuSecteur*, on remarque que de nombreux noeuds sont impactés pour les modèles *Lt* et *Dlt*, avec plus de noeuds (cibles ou non) impactés pour le modèle *Lt*. Avec plus de noeuds cibles impactés, *Lt* obtient de meilleurs résultats pour le rappel tandis que *Dlt* obtient de meilleurs résultats pour la précision (à l'exception du premier événement où la précision est nulle, aucun noeud cible vrai positif n'ayant été détecté). Pour les modèles *Hlt* et *At*, les résultats sont meilleurs, la proportion de noeuds cibles impactés étant plus grande, elle est même proche de 100% des noeuds impactés pour le modèle *At*. La proportion de vrai positifs, dans les noeuds cibles impactés, étant également plus importante, la précision des deux modèles est supérieure à celle de *Dlt*.

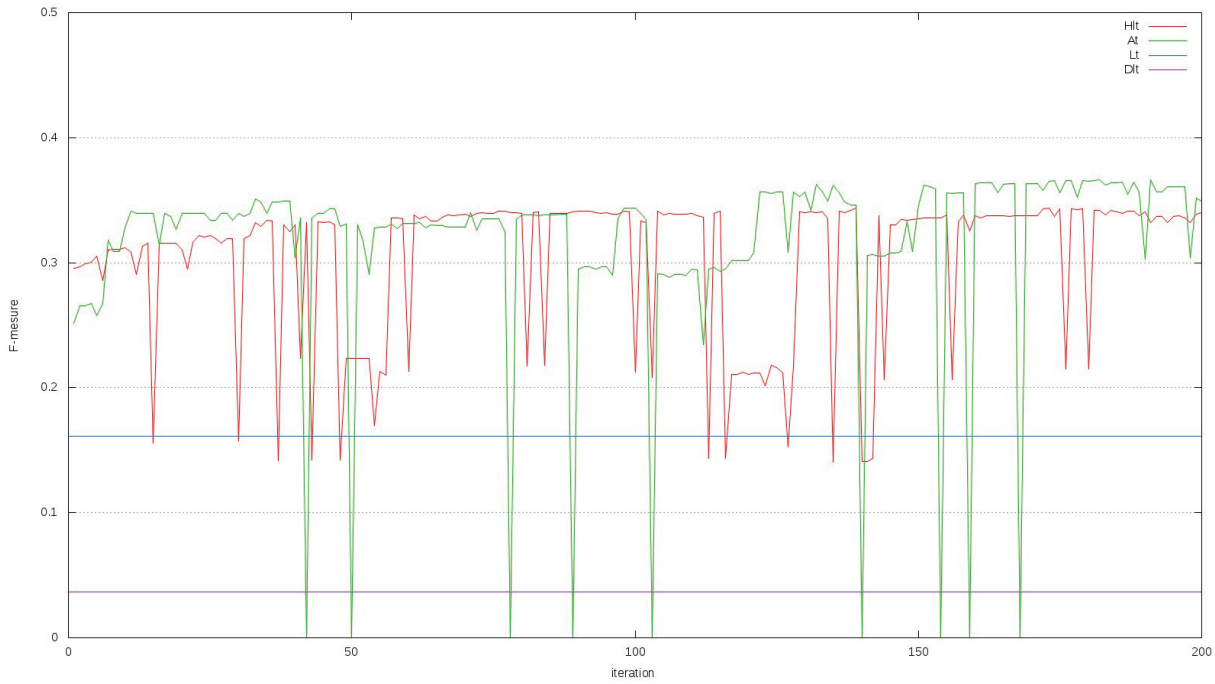


FIGURE 3.21 – Évolution de la F-mesure en fonction des itérations pour les événements de type *orgaOrga*.

Pour les événements de type *orgaOrga*, on remarque ici que le modèle *Dlt* n'offre pas de bons résultats lorsque les événements testés ne sont pas important, ce qui est le cas lorsque l'ensemble des noeuds sources n'est pas inclus dans l'ensemble des noeuds cibles (voir le tableau 3.12). En effet, dans ces cas de figure, à savoir l'ensemble des événements sauf *BayerMonsanto*, aucun noeud appartenant à l'ensemble des vrais positifs n'est détecté, ce qui a comme conséquence de réduire le score de *Dlt* à zéro. Le modèle *Lt* a le même problème mais seulement dans le cas où l'ensemble des noeuds sources est disjoint de l'ensemble des noeuds cibles. En conséquence, il offre de meilleurs résultats que le modèle *Dlt*. Le modèle *Hlt* impacte généralement plus de noeuds que le modèle *Lt*, ce qui lui permet d'impacter plus de noeuds cibles. De ce fait, dans les événements testés, il impacte toujours au moins quelques noeuds appartenant à l'ensemble des vrai positifs. En conséquence, il offre de meilleurs résultats en terme de rappel. Le modèle *At* atteint ici encore une grande proportion de noeuds cibles, ce qui lui permet d'obtenir de bons résultats (notamment pour les événements basés sur des noeuds sources disjoints des noeuds cibles).

Globalement, nous notons que l'optimisation des pondérations de seuils permet d'obtenir un score moyen sur l'ensemble des événements, indépendamment des spécificités individuelles de ces derniers. Cependant, cette généralisation peut avoir comme conséquence de réduire le score d'un événement en particulier, ce qui est le cas pour l'événement *commandeAirbusEutelsat* où c'est le modèle *Lt* qui offre de meilleurs résultats.

3.5.4 Évaluation du score

Un intérêt sous-jacent à l'utilisation de la fonctionnalité de score était pour nous de chercher à quantifier l'impact d'un événement, c'est-à-dire se poser la question de l'importance de l'événement pour l'entité nommée représentée par le noeud. Étant donné que les noeuds sont différents les uns des autres (par le type et le poids), nous avons commencé à traiter cette problématique par la comparaison de scores finaux entre deux événements. Ceci revient à ordonner les événements impactant un noeud les uns par rapport aux autres.

Pour cela, nous avons utilisé les scores obtenus par les noeuds lors de l'expérimentation de pondérations des seuils. Nous avons ensuite comparé, pour des noeuds de mêmes type, les scores obtenus par les modèles *Hlt* et *At*. Nous avons observé à chaque fois deux noeudspour chacun des types de noeuds *Organisation*, *Lieu* et *Secteur*. Les résultats sont présentés dans les tableaux 3.14 et 3.15.

Événement	Modèle	|I|	VP	FP	FN	Précision	Rappel	F-mesure
oilReductionSaudiArabia	lt	8 593	6	156	12	0.04	0.33	0.07
	dlt	2 965	0	43	18	0.00	0.00	0.00
	hlt	2 545	6	79	12	0.07	0.33	0.12
	at	148	13	130	5	0.09	0.72	0.16
electricCarNorway	lt	2 728	3	13	75	0.19	0.04	0.06
	dlt	1 008	1	2	77	0.33	0.01	0.02
	hlt	1 244	18	24	60	0.43	0.23	0.30
	at	45	16	29	62	0.36	0.21	0.26
RussiaFirstOnWeapons	lt	5 303	4	81	5	0.05	0.44	0.09
	dlt	2 615	2	28	7	0.07	0.22	0.10
	hlt	1 395	3	29	6	0.09	0.33	0.15
	at	42	4	36	5	0.10	0.44	0.16
BayerMonsanto	lt	1 691	16	11	76	0.59	0.17	0.27
	dlt	1 244	14	6	78	0.70	0.15	0.25
	hlt	2 812	53	96	39	0.36	0.58	0.44
	at	1 031	69	212	23	0.25	0.75	0.37
IBMRedHat	lt	183	2	2	15	0.50	0.12	0.19
	dlt	50	0	1	17	0.00	0.00	0.00
	hlt	1 747	13	80	4	0.14	0.76	0.24
	at	94	8	20	9	0.29	0.47	0.36
CMACeva	lt	50	0	0	10	0.00	0.00	0.00
	dlt	36	0	0	10	0.00	0.00	0.00
	hlt	178	5	8	5	0.38	0.50	0.43
	at	27	5	0	5	1.00	0.50	0.67
commandeAirbusJetBlue	lt	488	2	3	24	0.40	0.08	0.13
	dlt	125	0	1	26	0.00	0.00	0.00
	hlt	1 835	17	31	9	0.35	0.65	0.46
	at	337	19	54	7	0.26	0.73	0.38
commandeAirbusDeltaAirLines	lt	1 023	4	4	14	0.50	0.22	0.31
	dlt	173	0	1	18	0.00	0.00	0.00
	hlt	1 697	11	35	7	0.24	0.61	0.34
	at	308	12	57	6	0.17	0.67	0.28
commandeAirbusEutelsat	lt	632	2	4	11	0.33	0.15	0.21
	dlt	251	0	1	13	0.00	0.00	0.00
	hlt	1 603	6	50	7	0.11	0.46	0.17
	at	374	10	93	3	0.10	0.77	0.17
chuteSFRAltice	lt	289	0	0	13	0.00	0.00	0.00
	dlt	116	0	0	13	0.00	0.00	0.00
	hlt	256	4	8	9	0.33	0.31	0.32
	at	92	7	21	6	0.25	0.54	0.34

Tableau 3.13 – Résultats des modèles pour le problème de classification.

Les événements *commandeAirbusJetBlue* et *commandeAirbusDeltaAirLines* correspondent à des ventes d'avions tandis que l'événement *commandeAirbusEutelsat* correspond à une vente de satellites.

Pour les noeuds de type *organisation*, *Bombardier* et *Boeing*, on remarque que Boeing est, quel que soit le modèle, toujours plus impacté que Bombardier quel que soit l'événement, ce qui est intéressant car Boeing est généralement considéré comme le principal concurrent d'airbus. Pour ce qui est de l'ordre relatif des événements les uns par rapport aux autres, le modèle *Hlt*, sur ces événements-ci, donne de meilleurs résultats. Par rapport au modèle *At*, il classe l'événement *commandeAirbusEutelsat* en dernière position pour les deux noeuds, ce qui devrait être le cas. En effet, Bombardier et Boeing étant connus pour leur production d'aviation et devraient en conséquence être principalement intéressés par les deux autres événements qui correspondent

à des ventes d'avions. Pour ce qui est de la valeur des scores, même si elles sont pour l'instant vagues et nécessiterais une étude approfondie, elles sont intéressantes. De fait, on ne peut pas dire que Boeing est 3 à 7 fois plus impacté que Bombardier, cependant on peut remarquer que les valeurs sont proches pour un même noeud et des événements différents. Ceci tend à souligner que les événements peuvent être d'importance égale pour les entreprises.

La valeur du score est par contre très intéressante dès lors qu'on observe les noeuds de type *secteur*, *airline* et *satellite*. On remarque alors que la vente de satellites impacte très fortement le secteur industriel des satellites mais bien moins celui de l'aviation tandis qu'on observe le cas contraire pour les événements liés à la vente d'avions. En conséquence, le classement est également bon et est quasiment le même pour toutes les paires noeuds-événements quel que soit le modèle.

Globalement, nous remarquons qu'il n'est pas possible de comparer, dans la version actuelle du calcul des scores, les valeurs finales d'infections entre des noeuds de types différents. La pondération n'étant pas la même, il y a donc un biais initial. C'est par exemple le cas avec le modèle *Hlt* qui favorise l'impact sur les noeuds de type *organisation*, ce qui se remarque par le score élevé qu'ont ces derniers par rapports aux scores des noeuds de type *secteur*.

		Bombardier		Boeing		airline		satellite	
Événement	Modèle	score	pos	score	pos	score	pos	score	pos
	hlt	14.91	2	55.64	2	2.61	1	1.09	3
commandeAirbusJetBlue	at	2.36	2	14.95	2	5.65	1	3.04	2
	hlt	14.95	1	56.02	1	2.0	2	1.1	2
commandeAirbusDeltaAirLines	at	2.3	3	14.84	3	4.61	2	3.03	3
	hlt	14.91	2	55.56	3	1.26	3	10.15	1
commandeAirbusEutelsat	at	2.64	1	15.28	1	3.89	3	20.7	1

Tableau 3.14 – Scores obtenus par les noeuds de types *organisation* et *secteur* en fonction des modèles et des événements.

Lorsque l'on observe les noeuds de type *Lieu*, la *France* et les *USA*, le résultat des scores ne semble, pour le moment, pas bon du tout. En effet, comme observé dans le tableau 3.15, les modèles *Hlt* et *At*, après la simulation pour améliorer la pondération, proposent des résultats à l'opposé l'un de l'autre. Avec le modèle *Hlt*, aucun des deux pays n'est impacté, le score étant inférieur à 1. Or, il est nécessaire que le score d'infection soit supérieur à 1 pour que le noeud soit considéré comme concerné par l'événement. En conséquence, les pays ne transmettrons pas l'information de l'événement. Pour le modèle *At*, la situation est différente, ici les deux noeuds sont toujours concernés par les événements (en dehors des événements *IBMRedHat* et *CMACeva* pour la France). On remarque également que les événements de type différents ne vont pas avoir les mêmes positions dans le classement (car ayant eu des pondérations différentes). Ainsi, les événements de type *lieuSecteur* semblent être moins impactant que ceux de type *orgaOrga*.

Finalement, nous remarquons que l'utilisation d'un score pour évaluer l'impact d'un événement sur une entité semble possible mais que de nombreux problèmes restent à résoudre. La première étape, qui semble permettre de pouvoir obtenir de bons résultats, est de pouvoir classifier les événements de même type pour un noeud unique. Sur l'étude menée, pour des événements de même type et un noeud unique, il semble envisageable de trier ces événements pour ce noeud. Il reste cependant du travail à faire pour les noeuds de type *lieu*, et notamment les pays.

Les étapes à considérer pour la suite du trabail sont un calcul post-propagation du score final, notamment pour combler le biais créé par les pondérations différentes en fonction des types d'événement. Une autre option envisageable est de fournir un score d'infection initial différent aux noeuds sources (pour le moment égale à la valeur du seuil du noeud) qui pourrait là encore être un pourcentage du seuil du noeud déterminé en fonction de l'importance initiale supposée de l'événement.

		France		USA	
Événement	Modèle	score	pos	score	pos
	hlt	0.0	-	0.02	2
oilReductionSaudiArabia	at	3.18	6	4.86	6
	hlt	0.03	5	0.02	2
electricCarNorway	at	1.47	7	1.46	8
	hlt	0.02	6	0.01	3
RussiaFirstOnWeapons	at	1.18	8	1.21	10
	hlt	0.06	4	0.06	1
BayerMonsanto	at	15.04	2	40.32	1
	hlt	0.02	6	0.01	3
IBMRedHat	at	0.67	10	1.56	7
	hlt	0.06	4	0.02	2
CMACeva	at	0.91	9	1.26	9
	hlt	0.12	3	0.02	2
commandeAirbusJetBlue	at	4.76	5	5.76	4
	hlt	0.12	3	0.02	2
commandeAirbusDeltaAirLines	at	4.8	4	4.94	5
	hlt	0.4	1	0.02	3
commandeAirbusEutelsat	at	17.83	1	11.02	2
	hlt	0.23	2	0.01	3
chuteSFRAltice	at	14.31	3	8.02	3

Tableau 3.15 – Scores obtenus par les noeuds de type *lieu* en fonction des modèles et des événements.

3.6 Discussion

Pour répondre à la problématique de modélisation d'impact d'un événement dans le graphe économique de ReportLinker nous avons été amené à formaliser un problème de classification des noeuds ainsi que deux modèles pour répondre au problème. Nous avons obtenu de bons résultats à l'aide de ces modèles mais les expérimentations menées souffrent de différents biais.

Premièrement, les expérimentations menées souffrent d'un problème de robustesse, nous n'avons en effet utilisé qu'un millième des noeuds du graphe comme noeuds cibles, et uniquement des noeuds importants en termes de poids. L'une des conséquences est que l'on possède moins d'information sur des noeuds de plus petites tailles, représentant par exemple des sociétés nationales ou locales. Une autre conséquence est que les pondérations définies lors des expérimentations vont avoir tendance à favoriser ces noeuds cibles, ce qui est le cas avec le modèle *At*. Cependant cette situation a été décidée car il est nécessaire, pour chacun des analystes, d'étudier l'ensemble des noeuds cibles pour chaque événement. Or, mobiliser un analyste a un coût et augmenter la taille de l'ensemble des noeuds cibles aurait eu comme conséquence de diminuer le nombre d'événements à tester (nombre qui reste lui aussi relativement faible). Une solution aurait été de diminuer le nombre d'analystes pour un même événement. Cependant les analystes ont environ un tiers de noeuds en communs pour un même événement, ce qui aurait comme conséquence la perte des noeuds cibles.

Secondement ,nous avons été amenés à définir les valeurs initiales des paramètres en utilisant les types des noeuds sources et des noeuds cibles des analystes. Cela introduit alors un biais humain, certaines fonctionnalités ou valeurs vont avoir tendance à être favorisées en fonction de notre ressenti. Cette situation est particulièrement flagrante dans la première expérimentation sur les événements *Katrina* et *Bayer-Monsanto*. En effet, nous utilisons des pondérations pour les poids des arêtes et pour les seuils qui ont été définis à partir du score maximal du modèle, lorsque ce dernier utilise toutes les fonctionnalités. En conséquence, il est possible d'améliorer les résultats de plusieurs versions des modèles avec des pondérations différentes s'adaptant aux fonctionnalités utilisées. Cette situation s'explique par la combinatoire et le temps mis par chacun des modèles pour la propagation d'un événement. Compte tenu des machines et des modèles utilisés, le temps d'exécution d'une propagation peut évoluer de 2 à 8 minutes. Si l'on prend en compte seulement :

— les deux modèles *At* et *Hlt* avec l'utilisation de la fonction de score et une fonction de distance fixe
— les dix événements présentés dans la seconde expérimentation

— 100 valeurs différentes de pondérations de seuils possibles pour l'ensemble des types de noeuds du graphe
— 1 000 valeurs différentes de pondérations possibles pour l'ensemble des relations du graphe

On se retrouve alors à tester 2 millions de propagations différentes. Soit, avec un temps de calcul moyen d'une minute par propagation, un temps de calcul global proche de 4 années. Il est bien entendu possible de distribuer ces calculs ou d'utiliser des heuristiques mais ceci a un certain coût et n'aboutira pas forcément, dans le cas de certaines techniques, à de bons résultats. Par exemple les techniques d'apprentissage automatiques requièrent initialement un certain nombre de données pour effectuer leur apprentissage. Leur fournir seulement 7 événements, dans le cas de relations entre organisations, ne paraît donc pas suffisant et pourrait aboutir à un résultat de sur-apprentissage. Les pondérations apprise pour résoudre les 7 premiers cas ne donneraient pas forcément de meilleurs résultats pour un nouvel événement de même type.

Troisièmement, la qualité du graphe joue également un rôle important. Le graphe de ReportLinker possède des informations très intéressantes mais reste loin d'être parfait. Par exemple, des relations entre deux sociétés peuvent être absentes et la diffusion de l'information ne sera donc pas représentative, ne passant pas par cette relation non représentée. De même, les entités nommées détectées à partir des sources des documents sont loin d'être parfaites, tout comme le poids calculé à partir des documents. Par exemple, le fond documentaire étant uniquement en anglais et particulièrement fourni sur le marché américain, le poids du noeud représentant les États-Unis représente à lui seul un vingtième de l'ensemble des poids des noeuds du graphe. En conséquence, lorsque les États-Unis sont impactés beaucoup de noeuds du graphe sont très vite impactés pour le modèle *At* (c'est ce qui nous a amené à développer la fonctionnalité de distance). Dans le cas opposé, pour le modèle *Hlt*, le noeud correspondant aux États-Unis est rarement impacté car il faudrait qu'une grande proportion de ses voisins le soit également. Pour répondre à ces problématiques, plusieurs options ont été abordées pour améliorer la qualité du graphe de ReportLinker : travail en amont sur les sources des documents, travail sur le processus de calcul des entités nommées ou utilisation de techniques post-calcul du graphe.

Finalement, nous noterons que modéliser ce type de problème reste particulièrement difficile. Pour ne pas arranger les choses, on se retrouve avec une méta-problématique venant de la combinatoire des trois problèmes relevés précédemment. En effet, un nouveau calcul du graphe changera par exemple le poids des noeuds du graphe et obligera à redéfinir de nouvelles pondérations pour la propagation. Dans le même ordre d'idée, l'amélioration du graphe, par exemple par l'ajout de nouveaux types de noeuds et de relations, entraînera une augmentation de la combinatoire possible pour chacune des valeurs de pondérations sur les arêtes et sur les seuils. L'ajout de nouveaux noeuds à l'ensemble des noeuds cibles aura aussi un effet sur les pondérations utilisées jusque-là et obligera les analystes et les modèles à repasser sur des événements déjà évalués pour prendre en compte ces nouveaux noeuds.

3.7 Conclusion et ouvertures

Les problèmes de diffusion de données dans les graphes sont des problèmes largement étudiés. Dans le cadre du graphe d'entités nommées de ReportLinker nous avons essayé de modéliser les conséquences d'un événement historique sur un graphe de données majoritairement économiques. Pour travailler sur ce cas de figure nous avons proposé un problème. Pour ce problème nous avons défini deux modèles pour donner de meilleurs résultats.

Nous avons donc formalisé un problème, le problème de classification de l'information. Chaque instance de ce problème représente un événement historique et est définie à l'aide d'un type et d'un ensemble de noeuds sources. Le type correspond au genre de l'événement et les noeuds sources représentent les entités à partir desquelles l'événement à commencer à se diffuser. Pour évaluer la qualité des réponses apportées par les modèles de diffusion, un ensemble de noeuds cibles est sélectionné dans le graphe. Ainsi, pour chaque instance du problème, ces noeuds cibles sont répartis en deux sous-ensembles : les entités concernées par l'événement et celles qui ne le sont pas. En conséquence, on évalue la qualité d'un modèle par le nombre de noeuds que le dit-modèle arrive à correctement classifier : ce qui revient à dire si le modèle arrive bien à déterminer quelles entités sont impactées par un événement.

Pour résoudre ce problème, nous avons développé deux modèles de diffusion de l'information. Ces modèles sont inspirés du modèle de seuil linéaire, qui définit, pour chaque noeud du graphe, un seuil à partir duquel l'événement le concerne. Le principe étant qu'un noeud concerné par un événement transmettra l'information à ses voisins dans le graphe. Les modèles que nous avons développé reposent sur un ensemble de fonctionnalités

visant à utiliser les spécificités du graphe de ReportLinker. Nous utilisons donc les types des noeuds et des arêtes ainsi que la distance relative à la source des événements pour déterminer quels noeuds sont concernés par un événement donné.

Les expérimentations menées montrent que les fonctionnalités que nous avons développées permettent d'obtenir de meilleurs résultats que les modèles standards. De plus, l'utilisation d'un score d'infection semble pouvoir fournir une indication de l'impact d'un événement sur une entité.

Nous soulignons également que l'optimisation de ces modèles reste un problème compliqué dû à la combinatoire élevée des différentes possibilités offertes par les fonctionnalités.

L'étude de ce genre de problèmes reste très intéressante et offre plusieurs perspectives de développement futurs. En conséquence, nous souhaitons continuer les travaux initiés en abordant les points suivants :

- Augmenter la taille et diversifier l'ensemble des noeuds cibles, en ajoutant des entités témoins qui pourraient être, dans le cas du graphe de ReportLinker, des villes et organisations de moyenne taille, réparties sur l'ensemble de la planète afin de prendre en compte des plus petits noeuds et les situations géographiques locales
- Tester les modèles sur plus d'événements, notamment afin d'affiner les pondérations utilisées par les fonctionnalités
- Proposer de nouvelles fonctionnalités pour améliorer le score de chaque événement, notamment via :
  - une fonctionnalité de score originel qui évaluerait l'importance initiale supposée de l'événement
  - un calcul post-propagation des scores pour ré-évaluer les valeurs d'infections en fonction du type d'événement
  - d'autres définitions de la valeur d'infection qui prendraient en compte la proportion de voisins infectés au lieu du poids, ce qui serait utilisable pour tous les modèles dérivés de celui du modèle de seuil mais aussi pour d'autres modèles tel que le modèle de diffusion en cascades
- Utiliser les modèles sur des événements venant de se produire, et comparer les résultats obtenus à une évaluation à posteriori des analystes afin de déterminer si le modèle fournit de bonnes prédictions
- Utiliser les fonctionnalités de nos modèles pour adapter les graphes utilisés afin que les propagations simulées soient proches de celles observées
- Expérimenter notre problème de classification de l'infection avec d'autres types de modèles, notamment des modèles à base de cascades indépendantes ou de systèmes multi-agents
- Dans la continuité du point précédent, proposer pour ces autres types de modèles des fonctionnalités similaires à celles que nous avons développées pour tester si leurs ajouts améliorent les résultats
- Évaluer le problème et les modèles sur d'autres types de diffusion, notamment dans des cas de diffusion de maladies
- Proposer une fonctionnalité utilisant des méthodes d'analyse des sentiments pour améliorer et évaluer les résultats proposés : un événement donné est-il positif ou négatif pour telle entité ?

Conclusion

Conclusion

Dans cette thèse nous avons étudié des problèmes théoriques sur les graphes, à travers la recherche et l'énumération de sous-graphes denses (partie I), ainsi que des problèmes appliqués, via la diffusion d'événements dans des graphes d'entités nommées (partie II).

Dans le chapitre 1, nous avons proposé une méthode de recherche et d'énumération de cliques dans des graphes compressés par décomposition modulaire. Notre méthode se déroule en deux temps et repose sur l'exploration de l'arbre de décomposition modulaire ainsi que l'adaptation d'algorithmes sur la représentation compressée des graphes. La première partie de notre méthode consiste à compresser le graphe par décomposition modulaire. La compression fournit un arbre de décomposition modulaire ainsi qu'une liste de voisinage pour les parties de graphes incompressibles. La seconde étape de notre méthode est l'énumération des cliques sur la représentation compressée du graphe.

Les résultats obtenus montrent que notre méthode n'est pas efficace pour la recherche de la clique maximale du graphe. Nous supposons que ces résultats s'expliquent par le fait que les méthodes d'élagage habituelles des algorithmes de recherche de clique maximum basées sur le voisinage ne peuvent pas être appliquées.

Les résultats que nous avons obtenus pour l'énumération de cliques maximales montrent l'intérêt de notre méthode sur des graphes compressés. En effet, lorsqu'un graphe est suffisamment compressé, l'algorithme sur le graphe compressé donne de meilleurs résultats en terme de temps.

Dans le chapitre 2, nous avons proposé un algorithme et une heuristique d'énumération de quasi-cliques maximales. Notre méthode repose sur l'adaptions d'algorithmes d'énumérations de cliques maximales que nous adaptons en ajoutant les contraintes propres aux quasi-cliques.

Nous démontrons que l'algorithme développé permet d'énumérer l'ensemble des quasi-cliques maximales du graphe. Cependant l'algorithme prend un temps considérable à terminer l'énumération. De plus, l'utilisation d'un pivot pour élaguer certaines étapes de l'exploration ne semble pas utile après expérimentation empirique. En effet, le temps de calcul nécessaire à l'utilisation du pivot est plus important que le gain de temps acquis par les élagages.

Nous montrons avons en conséquence proposé une heuristique qui permet d'énumérer plus rapidement des quasi-cliques maximales avec cependant un certain nombre de quasi-cliques qui ne seront pas énumérées. Nous montrons également l'intérêt d'utiliser les quasi-cliques énumérées, via la co-occurrence des noeuds, pour évaluer les relations entre les noeuds du graphe.

Dans le chapitre 3, nous avons étudié la diffusion d'informations dans un graphe économique. L'idée repose sur une exploitation des particularités du graphe de ReportLinker pour modéliser les conséquences d'un événement historique.

Nous avons proposé un problème de classification de l'infection. Chaque instance du problème représente un événement. Chaque événement est représenté par un ensemble de noeud sources et un type d'événement représentant la catégorie de l'événement. Pour chaque instance le but est de définir si les noeuds cibles du graphe sont concernés. Les noeuds cibles représentent des entités nommées majeures du graphe comme par exemple des grosses sociétés, secteurs industriels ou pays. La qualité des approches testées est donc évalué via le nombre de noeuds cibles correctement classifiés.

Pour résoudre ce problème, nous avons proposé deux modèles de propagation de l'information qui reposent sur le modèle de seuil linéaire. Le seuil représente une valeur critique pour le noeud à partir duquel il se considère comme concerné par l'événement. Un noeud concerné par l'événement transmettra à son tour l'information à ses voisins. Pour améliorer les résultats obtenus au problème de classification de l'infection nous avons proposé de nouvelles fonctionnalités adaptées au graphe de ReportLinker. Les expériences conduites avec l'aide d'analystes montrent que les fonctionnalités qui prennent en compte les spécificités du graphe permettent de surpasser le modèle standard pour le problème de classification de l'infection.

Les expériences réalisées dans le cadre de cette thèse montrent qu'il est possible d'améliorer les travaux effectués. Pour cela, il nous semble intéressant d'aborder en priorité les points suivants :

— Proposer un algorithme de recherche et d'énumération de cliques sur la représentation compressée du graphe obtenue par décomposition modulaire (chapitre 1) : nous souhaitons tester si l'enregistrement de certaines informations lors de la compression de graphe peut être exploité, notamment pour la recherche de clique maximum. En effet, l'utilisation d'un poids sur les modules de l'arbre de décomposition modulaire permettrait notamment de développer de nouveaux élagages équivalant à ceux utiliser sur le voisinage dans la recherche de clique maximum.
— Comparer notre méthode d'évaluation des liens par co-occurrence aux méthodes de prédiction de liens (chapitre 2) : nous souhaitons utiliser le score présent dans les matrices pour tester l'apparition de nouveaux liens. On pourrait envisager d'utiliser des graphes temporels pour mesurer, à un instant t, quelles sont les paires de noeuds qui possèdent un haut pourcentage de co-occurrence sans que les noeuds ne soient reliés dans le graphe. La suite serait de tester à des temps $t + x$ si ces paires de noeuds finissent par avoir un lien.
— Augmenter la taille de l'ensemble des noeuds cibles pour affiner les résultats de la classification de l'infection (chapitre 3) : nous envisageons d'augmenter la taille de l'ensemble des noeuds cibles afin de mesurer l'impact des événements sur des entités nommées de plus petites tailles. Les noeuds cibles actuels ne représentent qu'un millième du noeud et seulement des noeuds considérés comme importants. Cependant, pour rendre les modèles plus précis il conviendrait de mesurer également l'impact local des événements. Pour cela, augmenter la taille de l'ensemble des noeuds cibles en y ajoutant par exemple des villes ou des moyennes entreprises semble une bonne idée.
— Tester de nouveaux modèles, avec des fonctionnalités similaires à celles développées, pour la propagation d'événements (chapitre 3) : nos nouveaux modèles reposent sur le modèle de seuil linéaire. Il nous semble donc intéressant de comparer les résultats obtenus à d'autres modèles. De plus, l'adaptation des fonctionnalités développées pour le modèle de seuil permettrait une validation supplémentaire de leur utilité. En effet il semble intéressant de valider ces fonctionnalités par l'évaluation de leur impact sur d'autres modèles comme le modèle en cascade.
— Utiliser une analyse des sentiments pour catégoriser l'impact de la propagation des événements sur les entités nommées (chapitre 3) : nous avons commencé à mesurer l'impact d'un événement via un score d'infection. Nous pensons donc que l'utilisation de sémantique, notamment via les types de noeuds, événements et liens, permettrait de mieux évaluer les scores pour chaque entité nommée.

En conclusion, cette thèse traite de problématiques de graphes appliqués sur des données réelles. Une de nos volontés initiales était d'améliorer le graphe et d'utiliser les connaissances incluses dedans, ce qui a été abordé respectivement avec les quasi-cliques et avec la diffusion d'événements. Nous comptons continuer à travailler sur l'utilisation des données du graphe en approfondissant le travail de propagation. Nous souhaitons dans un premier temps définir, via un apprentissage semi-automatique, les meilleures pondérations possibles par type d'événement. Les résultats obtenus permettront de choisir un modèle parmi ceux évalués. Une fois ce modèle sélectionné nous comptons approfondir l'évaluation pondérée de l'impact d'un événement en proposant une fonctionnalité de tri des événements impactant par noeud. L'idée est de pouvoir donner, pour une entité nommée, les événements qui la concerne en temps réel.

Index

Bibliographie

[1] James Abello, Mauricio GC Resende, and Sandra Sudarsky. Massive quasi-clique detection. In *Latin American Symposium on Theoretical Informatics*, pages 598–612. Springer, 2002.

[2] Micah Adler and Michael Mitzenmacher. Towards compressing web graphs. In *Data Compression Conference, 2001. Proceedings. DCC 2001.*, pages 203–212. IEEE, 2001.

[3] Yong-Yeol Ahn, James P Bagrow, and Sune Lehmann. Link communities reveal multiscale complexity in networks. *nature*, 466(7307) :761, 2010.

[4] Azadeh Alimadad, Vahid Dabbaghian, Suraj K Singhk, and Herbert H Tsang. Modeling hiv spread through sexual contact using a cellular automaton. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 2345–2350. IEEE, 2011.

[5] Roy M Anderson and Robert M May. Population biology of infectious diseases : Part i. *Nature*, 280(5721) :361, 1979.

[6] Vladimir Balash, Alfia Chekmareva, Alexey Faizliev, Sergei Sidorov, Sergei Mironov, and Daniil Volkov. Analysis of news flow dynamics based on the company co-mention network characteristics. In *International Workshop on Complex Networks and their Applications*, pages 521–533. Springer, 2018.

[7] Nicola Barbieri, Francesco Bonchi, and Giuseppe Manco. Topic-aware social influence propagation models. In *Data Mining (ICDM), 2012 IEEE 12th International Conference on*, pages 81–90. IEEE, 2012.

[8] Roberto Battiti and Franco Mascia. Reactive local search for maximum clique : A new implementation. Technical report, University of Trento, 2007.

[9] Roberto Battiti and Marco Protasi. Reactive local search for the maximum clique problem 1. *Algorithmica*, 29(4) :610–637, 2001.

[10] Victor Benjamin and Hsinchun Chen. Securing cyberspace : Identifying key actors in hacker communities. In *Intelligence and Security Informatics (ISI), 2012 IEEE International Conference on*, pages 24–29. IEEE, 2012.

[11] Victor Benjamin, Weifeng Li, Thomas Holt, and Hsinchun Chen. Exploring threats and vulnerabilities in hacker web : Forums, irc and carding shops. In *Intelligence and Security Informatics (ISI), 2015 IEEE International Conference on*, pages 85–90. IEEE, 2015.

[12] Anne Bergeron, Cédric Chauve, Fabien De Montgolfier, and Mathieu Raffinot. Computing common intervals of k permutations, with applications to modular decomposition of graphs. *SIAM Journal on Discrete Mathematics*, 22(3) :1022–1039, 2008.

[13] Shishir Bharathi, David Kempe, and Mahyar Salek. Competitive influence maximization in social networks. In *International Workshop on Web and Internet Economics*, pages 306–311. Springer, 2007.

[14] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics : theory and experiment*, 2008(10) :P10008, 2008.

[15] Gert Jan Boender, Ronald Meester, Edo Gies, and Mart CM De Jong. The local threshold for geographical spread of infectious diseases between farms. *Preventive veterinary medicine*, 82(1-2) :90–101, 2007.

[16] Paolo Boldi and Sebastiano Vigna. The webgraph framework i : compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602. ACM, 2004.

[17] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *IEEE transactions on knowledge and data engineering*, 20(2) :172–188, 2008.

[18] Coen Bron and Joep Kerbosch. Algorithm 457 : finding all cliques of an undirected graph. *Communications of the ACM*, 16(9) :575–577, 1973.

[19] Mauro Brunato, Holger H Hoos, and Roberto Battiti. On effectively finding maximal quasi-cliques in graphs. In *Learning and Intelligent Optimization*, pages 41–55. Springer, 2007.

[20] Sandro Brusco and Fabio Castiglionesi. Liquidity coinsurance, moral hazard, and financial contagion. *The Journal of Finance*, 62(5) :2275–2302, 2007.

[21] Antonio Caliò and Andrea Tagarelli. Trust-based dynamic linear threshold models for non-competitive and competitive influence propagation. *arXiv preprint arXiv :1805.11303*, 2018.

[22] Romain Campigotto, Patricia Conde Céspedes, and Jean-Loup Guillaume. A generalized and adaptive method for community detection. *arXiv preprint arXiv :1406.2518*, 2014.

[23] Christian Capelle. *Decompositions de graphes et permutations factorisantes*. PhD thesis, Montpellier 2, 1997.

[24] Christian Capelle, Michel Habib, and Fabien De Montgolfier. Graph decompositions and factorizing permutations. *Discrete Mathematics and Theoretical Computer Science*, 5(1) :55–70, 2002.

[25] Matteo Chinazzi and Giorgio Fagiolo. Systemic risk, contagion, and financial networks : A survey. *Institute of Economics, Scuola Superiore Sant'Anna, Laboratory of Economics and Management (LEM) Working Paper Series*, 2015.

[26] Aaron Clauset. Finding local community structure in networks. *Physical review E*, 72(2) :026132, 2005.

[27] Jeffrey Dean and Sanjay Ghemawat. Mapreduce : simplified data processing on large clusters. *Communications of the ACM*, 51(1) :107–113, 2008.

[28] Site du dimacs challenge. `http://archive.dimacs.rutgers.edu/pub/challenge/graph/benchmarks/clique/`. Accédé le : 31-10-2018.

[29] David Eppstein and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. In *Experimental Algorithms*, pages 364–375. Springer, 2011.

[30] P Erdős and A Rényi. On random graphs i. *Publ. Math. Debrecen*, 6 :290–297, 1959.

[31] Paul Erdős and András Hajnal. On chromatic number of graphs and set-systems. *Acta Mathematica Academiae Scientiarum Hungarica*, 17(1-2) :61–99, 1966.

[32] TS Evans and Renaud Lambiotte. Line graphs, link partitions, and overlapping communities. *Physical Review E*, 80(1) :016105, 2009.

[33] Torsten Fahle. Simple and fast : Improving a branch-and-bound algorithm for maximum clique. In *European Symposium on Algorithms*, pages 485–498. Springer, 2002.

[34] Gary William Flake, Robert E Tarjan, and Kostas Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1(4) :385–408, 2004.

[35] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5) :75–174, 2010.

[36] Tibor Gallai. Transitiv orientierbare graphen. *Acta Mathematica Hungarica*, 18(1) :25–66, 1967.

[37] Jessica A Gephart and Michael L Pace. Structure and evolution of the global seafood trade network. *Environmental Research Letters*, 10(12) :125014, 2015.

[38] Edgar N Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4) :1141–1144, 1959.

[39] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12) :7821–7826, 2002.

[40] Fred Glover. Tabu search—part i. *ORSA Journal on computing*, 1(3) :190–206, 1989.

[41] Fred Glover. Tabu search—part ii. *ORSA Journal on computing*, 2(1) :4–32, 1990.

[42] Jacob Goldenberg, Barak Libai, and Eitan Muller. Talk of the network : A complex systems look at the underlying process of word-of-mouth. *Marketing letters*, 12(3) :211–223, 2001.

[43] Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. Learning influence probabilities in social networks. In *Proceedings of the third ACM international conference on Web search and data mining*, pages 241–250. ACM, 2010.

[44] Sanjeev Goyal, Hoda Heidari, and Michael Kearns. Competitive contagion in networks. *Games and Economic Behavior*, 2014.

[45] Mark Granovetter. Threshold models of collective behavior. *American journal of sociology*, 83(6) :1420–1443, 1978.

[46] TR Grant. Dominance and association among members of a captive and a free-ranging group of grey kangaroos (macropus giganteus). *Animal Behaviour*, 21(3) :449–456, 1973.

[47] TR Grant. Kangaroo network dataset – KONECT, January 2016.

[48] Furkan Gürsoy. *Targeted and budgeted influence maximization in social networks under deterministic linear threshold model*. PhD thesis, Istanbul Sehir University, 2018.

[49] Michel Habib, Fabien De Montgolfier, and Christophe Paul. A simple linear-time modular decomposition algorithm for graphs, using order extension. In *Algorithm Theory-SWAT 2004*, pages 187–198. Springer, 2004.

[50] Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1) :41–59, 2010.

[51] Michel Habib, Christophe Paul, and Laurent Viennot. Partition refinement techniques : An interesting algorithmic tool kit. *International Journal of Foundations of Computer Science*, 10(02) :147–170, 1999.

[52] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[53] Lars Hagen and Andrew B Kahng. New spectral methods for ratio cut partitioning and clustering. *IEEE transactions on computer-aided design of integrated circuits and systems*, 11(9) :1074–1085, 1992.

[54] Frank Harary and Robert Z Norman. Some properties of line digraphs. *Rendiconti del Circolo Matematico di Palermo*, 9(2) :161–168, 1960.

[55] Herbert W Hethcote. The mathematics of infectious diseases. *SIAM review*, 42(4) :599–653, 2000.

[56] Petter Holme and Beom Jun Kim. Growing scale-free networks with tunable clustering. *Physical review E*, 65(2) :026107, 2002.

[57] Yichuan Jiang and JC Jiang. Diffusion in social networks : A multiagent perspective. *IEEE Transactions on Systems, Man, and Cybernetics : Systems*, 45(2) :198–213, 2015.

[58] Mira Kantemirova, Zaur Dzakoev, Zara Alikova, Sergei Chedgemov, and Zarina Soskieva. Percolation approach to simulation of a sustainable network economy structure. *Entrepreneurship and Sustainability Issues*, 5(3) :502–513, 2018.

[59] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[60] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 137–146. ACM, 2003.

[61] David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. *Theory OF Computing*, 11(4) :105–147, 2015.

[62] Dror Y Kenett and Shlomo Havlin. Network science : a useful tool in economics and finance. *Mind & Society*, 14(2) :155–167, 2015.

[63] William O Kermack and Anderson G McKendrick. Contributions to the mathematical theory of epidemics. ii.—the problem of endemicity. *Proc. R. Soc. Lond. A*, 138(834) :55–83, 1932.

[64] Brian W Kernighan and Shen Lin. An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2) :291–307, 1970.

[65] Jeong Han Kim and Van H Vu. Generating random regular graphs. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 213–222. ACM, 2003.

[66] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *science*, 220(4598) :671–680, 1983.

[67] Mohammed Lalou and Hamamache Kheddouci. Least squares method for diffusion source localization in complex networks. In *International Workshop on Complex Networks and their Applications*, pages 473–485. Springer, 2016.

[68] Eugene L Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Generating all maximal independent sets : Np-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3) :558–565, 1980.

[69] Jure Leskovec, Lada A Adamic, and Bernardo A Huberman. The dynamics of viral marketing. *ACM Transactions on the Web (TWEB)*, 1(1) :5, 2007.

[70] Jure Leskovec and Andrej Krevl. SNAP Datasets : Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[71] Jure Leskovec and Rok Sosič. Snap : A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(1) :1, 2016.

[72] Fa-Hsien Li, Cheng-Te Li, and Man-Kwan Shan. Labeled influence maximization in social networks for target marketing. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, pages 560–563. IEEE, 2011.

[73] Yuchen Li, Dongxiang Zhang, and Kian-Lee Tan. Real-time targeted influence maximization for online advertisements. *Proceedings of the VLDB Endowment*, 8(10) :1070–1081, 2015.

[74] Iouliana Litou, Vana Kalogeraki, Ioannis Katakis, and Dimitrios Gunopulos. Real-time and cost-effective limitation of misinformation propagation. In *2016 17th IEEE International Conference on Mobile Data Management (MDM)*, pages 158–163. IEEE, 2016.

[75] Yang Liu, Bo Wei, Zhen Wang, and Yong Deng. Immunization strategy based on the critical node in percolation transition. *Physics Letters A*, 379(43-44) :2795–2801, 2015.

[76] Zaixin Lu, Wei Zhang, Weili Wu, Bin Fu, and Dingzhu Du. Approximation and inapproximation for the influence maximization problem in social networks under deterministic linear threshold model. In *Distributed Computing Systems Workshops (ICDCSW), 2011 31st International Conference on*, pages 160–165. IEEE, 2011.

[77] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel : a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[78] Fragkiskos D Malliaros and Michalis Vazirgiannis. Clustering and community detection in directed networks : A survey. *Physics Reports*, 533(4) :95–142, 2013.

[79] Hideo Matsuda, Tatsuya Ishihara, and Akihiro Hashimoto. Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theoretical Computer Science*, 210(2) :305–325, 1999.

[80] Ross M McConnell and Jeremy P Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1) :189–241, 1999.

[81] Yasir Mehmood, Francesco Bonchi, and David García-Soriano. Spheres of influence for more effective viral marketing. In *Proceedings of the 2016 International Conference on Management of Data*, pages 711–726. ACM, 2016.

[82] Mattia Montagna and Christoffer Kok. Multi-layered interbank model for assessing systemic risk. Technical report, European Central Bank, 2016.

[83] Saket Navlakha, Rajeev Rastogi, and Nisheeth Shrivastava. Graph summarization with bounded error. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 419–432. ACM, 2008.

[84] Mark EJ Newman. Modularity and community structure in networks. *Proceedings of the national academy of sciences*, 103(23) :8577–8582, 2006.

[85] Mark EJ Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2) :026113, 2004.

[86] Mark EJ Newman and Duncan J Watts. Renormalization group analysis of the small-world network model. *Physics Letters A*, 263(4-6) :341–346, 1999.

[87] Romualdo Pastor-Satorras, Claudio Castellano, Piet Van Mieghem, and Alessandro Vespignani. Epidemic processes in complex networks. *Reviews of modern physics*, 87(3) :925, 2015.

[88] Bharath Pattabiraman, Md Mostofa Ali Patwary, Assefaw H Gebremedhin, Wei-keng Liao, and Alok Choudhary. Fast algorithms for the maximum clique problem on massive sparse graphs. In *Algorithms and Models for the Web Graph*, pages 156–169. Springer, 2013.

[89] Sen Pei, Xian Teng, Jeffrey Shaman, Flaviano Morone, and Hernán A Makse. Collective influence maximization in threshold models of information cascading with first-order transitions. *arXiv preprint arXiv :1606.02739*, 2016.

[90] Patrick Prosser. Exact algorithms for maximum clique : A computational study. *Algorithms*, 5(4) :545–587, 2012.

[91] Nataša Przulj. Graph theory approaches to protein interaction data analysis. *proteins*, 120 :000, 2003.

[92] Wayne Pullan and Holger H Hoos. Dynamic local search for the maximum clique problem. *Journal of Artificial Intelligence Research*, 25 :159–185, 2006.

[93] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[94] Pratha Sah, Lisa O Singh, Aaron Clauset, and Shweta Bansal. Exploring community structure in biological networks with random graphs. *BMC bioinformatics*, 15(1) :220, 2014.

[95] Nikos Salamanos, Elli Voudigari, and Emmanuel J Yannakoudakis. Identifying influential spreaders by graph sampling. In *International Workshop on Complex Networks and their Applications*, pages 111–122. Springer, 2016.

[96] Pablo San Segundo, Diego Rodríguez-Losada, and Agustín Jiménez. An exact bit-parallel algorithm for the maximum clique problem. *Computers & Operations Research*, 38(2) :571–581, 2011.

[97] M Schone, R Esposito, M Cole, and G Greenwald. War on anonymous : British spies attacked cybercriminals, snowden docs show. *NBC News*, 2014.

[98] Sicong Shao, Cihan Tunc, Pratik Satam, and Salim Hariri. Real-time irc threat detection framework. In *Foundations and Applications of Self* Systems (FAS* W), 2017 IEEE 2nd International Workshops on*, pages 318–323. IEEE, 2017.

[99] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 22(8) :888–905, 2000.

[100] Hiroaki Shiokawa, Yasuhiro Fujiwara, and Makoto Onizuka. Fast algorithm for modularity-based graph clustering. In *AAAI*, pages 1170–1176, 2013.

[101] Pramesh Singh, Sameet Sreenivasan, Boleslaw K Szymanski, and Gyorgy Korniss. Threshold-limited spreading in social networks with multiple initiators. *Scientific reports*, 3 :2330, 2013.

[102] Site du projet spark de la fondation apache. `https://spark.apache.org/`. Accédé le : 17-12-2018.

[103] Site du stanford network analysis project, qui propose une librairie c++ pour la manipulation de graphes. `https://snap.stanford.edu/`. Accédé le : 04-12-2018.

[104] Angelika Steger and Nicholas C Wormald. Generating random regular graphs quickly. *Combinatorics, Probability and Computing*, 8(4) :377–396, 1999.

[105] Michael Svendsen, Arko Provo Mukherjee, and Srikanta Tirthapura. Mining maximal cliques from a large graph using mapreduce : Tackling highly uneven subproblem sizes. *Journal of Parallel and Distributed Computing*, 79 :104–114, 2015.

[106] Yuanyuan Tian, Richard A Hankins, and Jignesh M Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 567–580. ACM, 2008.

[107] Hannu Toivonen, Fang Zhou, Aleksi Hartikainen, and Atte Hinkka. Compression of weighted graphs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 965–973. ACM, 2011.

[108] Etsuji Tomita and Toshikatsu Kameda. An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. *Journal of Global optimization*, 37(1) :95–111, 2007.

[109] Etsuji Tomita and Tomokazu Seki. An efficient branch-and-bound algorithm for finding a maximum clique. In *Discrete mathematics and theoretical computer science*, pages 278–289. Springer, 2003.

[110] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, Shinya Takahashi, and Mitsuo Wakatsuki. A simple and faster branch-and-bound algorithm for finding a maximum clique. In *International Workshop on Algorithms and Computation*, pages 191–203. Springer, 2010.

[111] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1) :28–42, 2006.

[112] Armin Töpfer, Tobias Marschall, Rowena A Bull, Fabio Luciani, Alexander Schönhuth, and Niko Beerenwinkel. Viral quasispecies assembly via maximal clique enumeration. *PLoS computational biology*, 10(3) :e1003515, 2014.

[113] Samuel Unicomb, Gerardo Iñiguez, and Márton Karsai. Threshold driven contagion on weighted networks. *Scientific reports*, 8(1) :3094, 2018.

[114] Takeaki Uno. An efficient algorithm for solving pseudo clique enumeration problem. *Algorithmica*, 56(1) :3–16, 2010.

[115] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684) :440, 1998.

[116] Dominic JA Welsh and Martin B Powell. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal*, 10(1) :85–86, 1967.

[117] Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, and Ion Stoica. Graphx : A resilient distributed graph system on spark. In *First International Workshop on Graph Data Management Experiences and Systems*, page 2. ACM, 2013.

[118] Lan Yang, Alessandro Giua, and Zhiwu Li. Minimizing the influence propagation in social networks for linear threshold models. *IFAC-PapersOnLine*, 50(1) :14465–14470, 2017.

[119] Xiao Yang, Jaroslaw Zola, and Srinivas Aluru. Parallel metagenomic sequence clustering via sketching and maximal quasi-clique enumeration on map-reduce clouds. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*, pages 1223–1233. IEEE, 2011.