Coloration de graphes : structures et algorithmes

Benjamin Lévêque

UNIVERSITÉ JOSEPH FOURIER - GRENOBLE I

THÈSE

présentée par

Benjamin LÉVÊQUE

pour obtenir le grade de docteur de l'Université Joseph Fourier

COLORATION DE GRAPHES : STRUCTURES ET ALGORITHMES

Soutenue le 15 octobre 2007

Directeur de thèse : Frédéric MAFFRAY

Spécialité : Recherche Opérationnelle, Combinatoire et Optimisation

Discipline : Mathématiques et Informatique

Composition du jury :

Gérard CORNUÉJOLS	Professeur Université Aix-Marseille II	Rapporteur
Sylvain GRAVIER	Directeur de Recherche CNRS	Examinateur
Yassine LAKHNECH	Professeur Université Grenoble I	Président du jury
Frédéric MAFFRAY	Directeur de Recherche CNRS	Directeur de thèse
Kristina VUŠKOVIĆ	Professeur Université Leeds	Examinateur
Dominique DE WERRA	Professeur EPFL Lausanne	Rapporteur

Thèse préparée au sein des laboratoires
Leibniz-IMAG et G-SCOP (CNRS, INPG, UJF)

Remerciements

Je remercie tout d'abord mon directeur de thèse, Frédéric Maffray, qui m'a permis d'effectuer cette thèse dans les meilleures conditions possibles. J'ai pu bénéficier de ses remarquables connaissances et compétences en théorie des graphes. Notre travail en commun s'est avéré très enrichissant.

Je remercie Gérard Cornuéjols et Dominique de Werra d'avoir accepté de rapporter cette thèse. Je remercie également Sylvain Gravier, Yassine Lakhnech et Kristina Vušković qui m'ont fait l'honneur de prendre part à mon jury de thèse.

Je remercie tous ceux avec qui j'ai travaillé pendant cette thèse, en particulier mes co-auteurs : Jean-Claude Bermond, David Coudert, Kathie Cameron, Jack Edmonds, Yannick Frein, Vincent Jost, David Lin, Frédéric Maffray, Bruce Reed, András Sebő et Nicolas Trotignon.

Je remercie tous mes collègues et amis des laboratoires Leibniz-IMAG et G-SCOP, dont la bonne humeur et la gentillesse ont été pour une grande part dans le plaisir que j'ai eu à effectuer cette thèse.

Je remercie mes parents, Marc et Michèle, tous les deux professeurs de mathématiques dans le secondaire. Ils ont su me donner, ainsi qu'à mes frères Sébastien et Stéphane, ce goût pour la réflexion et l'abstraction. Cela a commencé dès notre plus jeune âge. Marc nous a par exemple appris à jouer à un bridge à 9 cartes lorsque nos petites mains n'arrivaient pas encore à en tenir plus. Les quelques années de collège passées en classe avec Michèle nous ont appris les bases du raisonnement et de la rigueur. Elles nous ont aussi rapprochés du monde de l'enseignement, nous procurant un grand plaisir d'apprendre et de partager.

Finalement, je remercie Gaelle pour sa gentillesse et son amour. Ces quelques années passées ensemble à Grenoble nous ont comblés. Nous les prolongerions volontiers toute une vie.

Table des matières

Introduction

L'origine du problème de la coloration de graphes remonte au XIXème siècle lorsque Francis Guthrie, cartographe anglais, remarqua que quatre couleurs suffisent pour colorier la carte des cantons d'Angleterre, sans donner la même couleur à deux cantons ayant une frontière commune. Il pose alors la question de savoir si quatre couleurs suffisent toujours pour colorier n'importe quelle carte géographique de sorte que deux régions voisines n'aient pas la même couleur. Malgré un énoncé simple, cette conjecture est restée non résolue pendant plus d'un siècle. Il fallut attendre 1976 pour qu'Appel et Haken parviennent à démontrer ce résultat à l'aide d'un ordinateur. Même si le problème des cartographes est résolu, celui des mathématiciens ne l'est pas, car ce théorème traite seulement du cas particulier des graphes planaires.

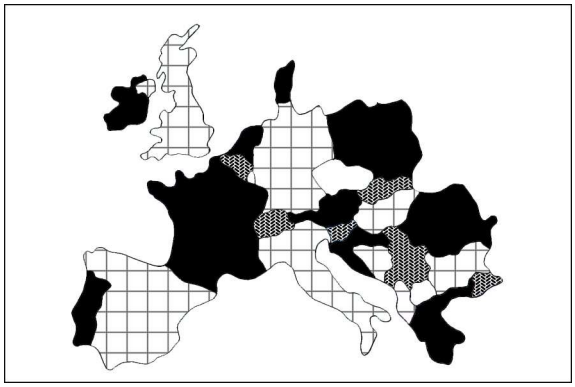


FIG. 1 – Carte de l'Europe coloriée avec 4 couleurs (mer comprise)

Un exemple plus récent d'application du problème de la coloration est celui de l'affectation de fréquences radio. Un opérateur de téléphonie dispose d'un certain nombre d'antennes sur le territoire. Certaines de ces antennes sont trop proches pour pouvoir émettre sur la même fréquence. L'opérateur doit donc attribuer des fréquences différentes aux antennes qui interfèrent entre elles, tout en cherchant à minimiser le nombre total de fréquences

utilisées. Pour le problème des cartes géographiques, il était impossible de dessiner cinq régions du plan qui se touchent toutes les unes les autres, sinon le théorème des quatre couleurs serait faux, mais maintenant, il devient envisageable d'avoir cinq antennes qui interfèrent toutes entre elles.

Ce problème se modélise par un graphe de la manière suivante : chaque antenne est représentée par un point (appelé sommet), et deux sommets sont reliés par une ligne (appelée arête) dès que les deux antennes correspondantes interfèrent entre elles. Le problème d'affectation de fréquences radio revient donc à associer à chaque sommet du graphe une couleur (correspondant à une fréquence) de sorte que deux sommets reliés par une arête ne reçoivent pas la même couleur, le but étant de minimiser le nombre de couleurs utilisées. Une coloration optimale d'un graphe est une coloration qui utilise le moins de couleurs possibles. Depuis une quarantaine d'années, nous soupçonnons qu'il n'existe aucune méthode efficace permettant de trouver une coloration optimale pour un graphe quelconque.

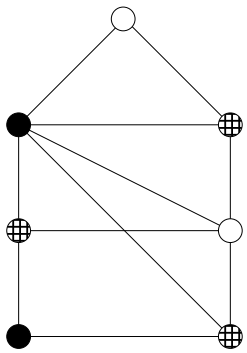


FIG. 2 – Exemple de coloration optimale d'un graphe avec 3 couleurs

En 1960, Claude Berge a défini la classe des graphes parfaits qui est apparue comme étant une classe de graphes assez générale pour laquelle le problème de la coloration pouvait être résolu efficacement. En 1984, Grötschel, Lovász et Schrijver ont démontré qu'il était possible de colorier optimalement les graphes parfaits en temps polynomial. Mais, du point de vue de la théorie des graphes, cette méthode n'est pas complètement satisfaisante car nous ne savons pas interpréter en terme de graphes comment se déroule l'algorithme. Le problème de trouver un algorithme efficace de type combinatoire permettant de colorier la classe des graphes parfaits est ouvert. L'existence de l'algorithme de Grötschel, Lovász et Schrijver motive la recherche d'algorithmes efficaces permettant de colorier les graphes parfaits ou des sous-classes de graphes parfaits.

Voici un résumé des différents chapitres qui composent cette thèse.

Chapitre 1 : Notions de base

Les définitions de théorie de graphes nécessaires à la suite de cette thèse sont introduites dans ce chapitre. Les notions d'algorithme et de complexité algorithmique sont expliquées brièvement. Le problème de la coloration amène à définir la classe des graphes parfaits. Quelques algorithmes classiques permettant de colorier des sous-classes de graphes parfaits sont également présentés.

Chapitre 2 : Contraction

Ce chapitre concerne l'opération de contraction de paire d'amis qui permet de colorier plusieurs sous-classes de graphes parfaits. Une généralisation de la notion de paire d'amis, ainsi qu'une conjecture à ce sujet, sont proposées. La définition d'ordre de contraction permet ensuite d'obtenir des résultats généraux concernant la recherche d'une clique de taille maximum, d'un stable fort, d'une coloration pondérée minimum et d'une clique pondérée maximum.

Chapitre 3 : Graphes sans taureau

Ce chapitre traite de la classe des graphes ne contenant ni taureau, ni trou impair, ni antitrou. En collaboration avec Frédéric Maffray [58], un nouvel algorithme permettant de colorier ces graphes en temps $\mathcal{O}(nm)$ est défini. Cet algorithme est basé sur le fait que ces graphes possèdent un ordre sur les sommets ayant une propriété spéciale. Plus généralement, il est montré que tous les graphes ne contenant ni taureau, ni trou, contiennent un sommet qui n'est pas le milieu d'un chemin sans corde de longueur cinq.

Chapitre 4 : Graphes de Meyniel

La classe des graphes de Meyniel peut être définie comme étant la classe de graphes ne contenant ni trou impair, ni maison. En collaboration avec Kathie Cameron, Jack Edmonds et Frédéric Maffray [12], l'algorithme le plus rapide permettant de colorier les graphes de Meyniel est étendu : il s'applique à n'importe quel graphe en entrée et retourne en temps $\mathcal{O}(n^2)$, soit une coloration optimale et une clique de même taille, soit un sous-graphe interdit dans les graphes de Meyniel.

Chapitre 5 : Graphes d'Artémis

La classe des graphes d'Artémis est définie comme étant la classe des graphes ne contenant ni trou impair, ni antitrou, ni prisme. Frédéric Maffray et Nicolas Trotignon ont montré que ces graphes étaient parfaitement contractiles. En collaboration avec Frédéric Maffray, Bruce Reed et Nicolas

Trotignon [59], l'algorithme sous-jacent à cette preuve est simplifié et accéléré. Un algorithme de complexité $\mathcal{O}(n^2m)$ permettant de colorier les graphes d'Artémis est obtenu.

Chapitre 6 : Extension de pré-coloration

Une variante du problème classique de la coloration considère qu'une partie des sommets a déjà reçu une couleur à priori. Le but est d'étendre cette pré-coloration de manière optimale. En collaboration avec Vincent Jost et Frédéric Maffray [55], il est montré que l'algorithme de coloration des graphes parfaits peut être utilisé pour étendre de manière optimale une pré-coloration quelconque des graphes complémentaires de Meyniel.

Chapitre 1

Notions de base

Les notions de base utilisées dans cette thèse sont présentées dans ce chapitre. Pour disposer de plus d'informations concernant ces notions, le lecteur peut se référer aux ouvrages classiques [4, 20, 24, 72].

1.1 Les graphes

Si V est un ensemble fini quelconque et si k est un entier positif, alors $\binom{V}{k}$ désigne l'ensemble des parties de V qui ont exactement k éléments. Un *graphe* est un couple d'ensembles (V, E) tel que $E \subseteq \binom{V}{2}$. Les éléments de V sont appelés les *sommets* de G, et les éléments de E sont appelés les *arêtes* de G. Afin d'alléger les notations, la paire contenant u et v est notée uv, au lieu de $\{u, v\}$. Si G est un graphe, alors $V(G)$ désigne l'ensemble de ses sommets et $E(G)$ l'ensemble de ses arêtes.

Soient G un graphe et u, v deux sommets de G. Si $uv \in E(G)$ alors u et v sont dit *adjacents* ou *voisins* ; il est aussi dit que u *voit* v. Si $uv \notin E(G)$, alors u et v sont dit *non-adjacents* ou *non-voisins* ; il est aussi dit que u *manque* v.

Par définition, les graphes définis dans cette thèse sont *sans boucle*, c'est-à-dire qu'un sommet v ne peut jamais être son propre voisin (car $\{v, v\}$ ne possédant qu'un seul élément, il n'appartient pas à $\binom{V}{2}$) ; ils sont *non-orientés*, c'est-à-dire que pour tous sommets u et v, $uv = vu$; ils sont *simples*, c'est-à-dire qu'entre deux sommets u et v, il existe 0 ou 1 arête, et non un nombre quelconque d'arêtes ; ils sont *finis*, c'est-à-dire que V possède un nombre fini d'éléments. Le nombre de sommets d'un graphe est noté n et son nombre d'arêtes m.

Soit G un graphe. Si $v \in V(G)$, alors l'ensemble des voisins de v est noté $N(v)$. Remarquons que $v \notin N(v)$. Le *degré* d'un sommet v est défini par

$d(v) = |N(v)|$. Si $A \subseteq V(G)$, alors $N(A)$ désigne l'ensemble des sommets de G ayant au moins un voisin dans A. Remarquons que A et $N(A)$ ne sont pas nécessairement disjoints.

Soit G un graphe. Le *graphe complémentaire* de G est le graphe noté $\overline{G}$, défini par $V(\overline{G}) = V(G)$ et $E(\overline{G}) = \{uv\,;\, u \in V(G),\, v \in V(G) \text{ et } uv \notin E(G)\}$. Soient F et G deux graphes. Un *isomorphisme* entre F et G désigne toute fonction φ bijective, associant à chaque sommet de F un sommet de G et telle que $uv \in E(F) \Leftrightarrow \varphi(u)\varphi(v) \in E(G)$. S'il existe un isomorphisme entre F et G, alors F et G sont dit *isomorphes*. Un graphe est *auto-complémentaire* s'il est isomorphe à son complémentaire.

Soient G un graphe et W un ensemble de sommets de G. Le *sous-graphe de G induit par W* est noté $G[W]$, c'est le graphe dont l'ensemble des sommets est W et dont l'ensemble des arêtes est $\binom{W}{2} \cap E(G)$. Une *clique* ou *graphe complet* est un graphe vérifiant $E(G) = \binom{V(G)}{2}$. La *taille* d'une clique est son nombre de sommets. Un *triangle* est une clique de taille trois. Un *stable* est un graphe vérifiant $E(G) = \emptyset$. Une *clique de G* est un graphe qui est à la fois une clique et un sous-graphe induit de G. Plus généralement, pour tout graphe "bidule" (clique, stable, trou ...), un *"bidule" de G* est un graphe qui est à la fois un "bidule" (ou qui est isomorphe à un "bidule") et un sous-graphe induit de G. Un graphe G *contient* un "bidule" si et seulement s'il existe un sous-graphe induit de G qui est un "bidule" de G. Un graphe G est *sans "bidule"* si et seulement s'il n'existe aucun "bidule" qui soit un sous-graphe induit de G ; il est dit aussi que G est "bidule"-*libre*. Un graphe G est *biparti* si et seulement s'il est possible de partitionner $V(G)$ en deux ensembles A et B tels que $G[A]$ et $G[B]$ sont des stables.

1.1.1 Chemins et cycles

Soit G un graphe. Un *chemin* de G est une suite $P = (v_1, v_2, \ldots, v_k)$ de sommets deux à deux distincts vérifiant, pour tout $1 \leq i \leq k-1$, $v_iv_{i+1} \in E(G)$. Une arête de G de la forme v_iv_{i+1} avec $1 \leq i \leq j$ est appelée *arête du chemin*. Une arête de G de la forme v_iv_j avec $|i-j| > 1$ est appelée *corde du chemin*. Une corde v_iv_j est *courte* si $|i-j| = 2$. La longueur du chemin est égale à son nombre d'arêtes. Les sommets v_1 et v_k sont les *extrémités du chemin* ; les autres sommets sont dits *intérieurs*. Les sommets de l'ensemble $\{v_{\lfloor \frac{k+1}{2} \rfloor}, v_{\lceil \frac{k+1}{2} \rceil}\}$ sont dits *milieu(x) du chemin*. Le sous-chemin de P induit par les sommets intérieurs est noté P^*.

Le graphe dont les sommets sont $v_1, v_2, \ldots, v_k$ et dont l'ensemble des arêtes est $\{v_iv_{i+1}; 1 \leq i \leq k-1\}$ est un chemin sans corde, noté v_1-v_2-$\cdots$-v_k. Le graphe P_k désigne un chemin sans corde sur k sommets, il est donc de

longueur $k-1$.

Un *cycle* de G est une suite $C = (v_1, v_2, \ldots, v_k)$ de sommets deux à deux distincts vérifiant, pour tout $1 \leq i \leq k$, $v_i v_{i+1} \in E(G)$, où l'addition des indices est effectuée modulo k. Une arête de G de la forme $v_i v_{i+1}$ avec $1 \leq i \leq j$ est appelée *arête du cycle*. Une arête de G de la forme $v_i v_j$ avec $|i-j| > 1$ est appelée *corde du cycle*. Une corde $v_i v_j$ est *courte* si $|i-j| = 2$. Deux cordes $v_i v_j$, $v_k v_l$ sont dites *croisées* si elles sont disjointes et si sur chaque chemin du cycle reliant v_i à v_j, il y a exactement un sommet de v_k, v_l. Elles sont dites *non-croisées* dans le cas contraire. La longueur d'un cycle est égale à son nombre d'arêtes.

Un *trou* est un graphe avec au moins cinq sommets pouvant être ordonnés de manière à former un cycle sans corde. Un *antitrou* est le complémentaire d'un trou. Un trou ou un antitrou est *pair* (resp. *impair*) s'il contient un nombre pair de sommets (resp. impair). Pour $k \geq 5$, le graphe C_k désigne un trou sur k sommets. Un *carré* est un cycle sans corde avec quatre sommets, noté C_4.

La relation $\sim$ entre les sommets d'un graphe G est définie de la manière suivante : $u \sim v$ si et seulement s'il existe un chemin de G d'extrémités u et v. C'est une relation d'équivalence. Les *composantes connexes* de G sont les sous-graphes induits par les classes d'équivalence de $\sim$. Si G possède une seule composante connexe alors G est dit *connexe*. Si $\overline{G}$ est connexe, alors G est dit *co-connexe*. Les *composantes co-connexes* de G sont les complémentaires des composantes connexes de $\overline{G}$. Pour tout graphe G, l'un de G ou $\overline{G}$ est connexe.

Soient G un graphe et u, v deux sommets de G. La *distance* de u à v est la longueur d'un chemin sans corde de G de longueur minimale, d'extrémités u et v. Si P est un chemin sans corde de G ayant une extrémité u, le sommet u' de P *le plus proche* de u ayant une certaine propriété, est le sommet u' de P vérifiant la propriété et minimisant la distance entre u et u' calculée dans le chemin P (et non pas dans le graphe G).

1.1.2 Quelques graphes particuliers

Le *domino* est le graphe ayant six sommets a, b, c, d, e, f et dont les arêtes sont $ab, bc, cd, de, ef, fa, be$ (voir figure 1.1).

Le *taureau* est le graphe ayant cinq sommets a, b, c, d, e et dont les arêtes sont ab, bc, cd, de, bd (voir figure 1.2). Un tel taureau est noté *a-bcd-e*. Dans un taureau *a-bcd-e*, les sommets b, d sont appelés les *oreilles* du taureau. Le taureau est un graphe auto-complémentaire $\overline{\textit{a-bcd-e}} = \textit{b-eca-d}$.

Une *maison* est un cycle impair de longueur au moins cinq avec une seule corde, qui est courte (voir figure 1.3). Dans une maison, le sommet qui forme

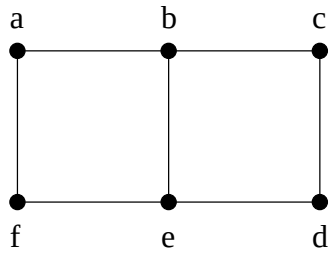


FIG. 1.1 – Le domino

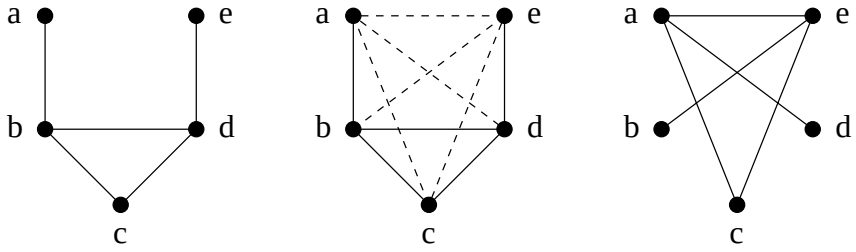


FIG. 1.2 – Le taureau, un graphe auto-complémentaire

un triangle avec l'unique corde est appelé le *toit de la maison*. La *maisonnette* est la plus petite maison, c'est aussi le graphe $\overline{P_5}$.

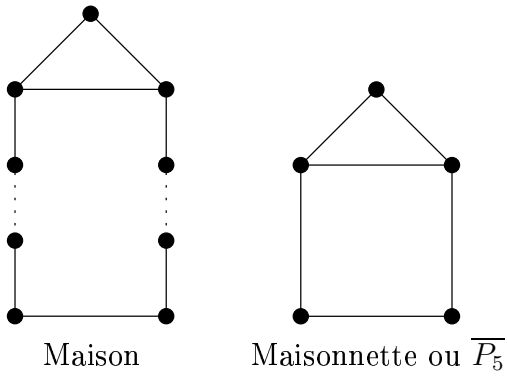


FIG. 1.3 – Les maisons

Un *prisme* est un graphe dont les sommets se partitionnent en trois ensembles induisant des chemins sans corde P_1, P_2, P_3 tels que pour $i = 1, 2, 3$, P_i est de longueur au moins 1, d'extrémités a_i et b_i, tels que $\{a_1, a_2, a_3\}$ et $\{b_1, b_2, b_3\}$ induisent des triangles et tels que, pour $1 \leq i < j \leq 3$, il n'existe aucune arête entre des sommets de P_i et P_j autre que celles des triangles (voir figure 1.4). Un prisme est dit *pair* (resp. *impair*) si les trois chemins P_1, P_2, P_3 sont de longueur paire (resp. impaire).

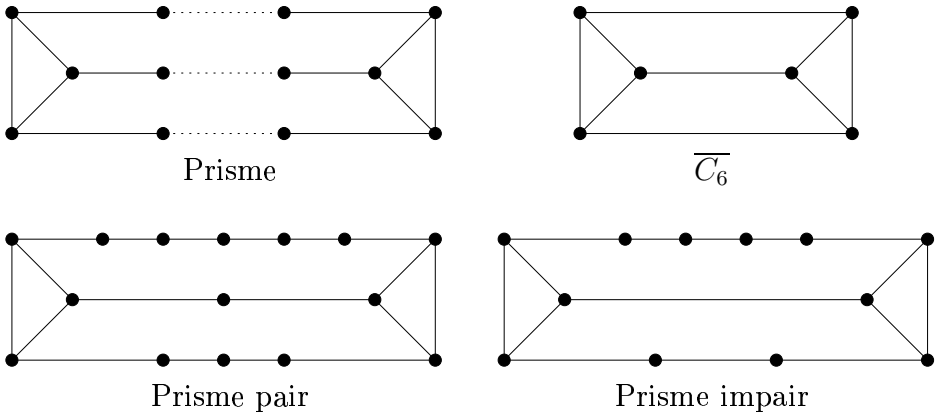


FIG. 1.4 – Les prismes

1.1.3 Algorithmes sur les graphes

La notion d'*algorithme* est définie formellement dans [72]. De manière simple, il s'agit d'une suite d'opérations permettant de résoudre un problème par le calcul. Parmis les plus célèbres, l'algorithme qui se trouve dans le livre 7 des Eléments d'Euclide [27] permet de trouver le plus grand commun diviseur de deux nombres.

La notion d'efficacité d'un algorithme est étudiée depuis le milieu du XX^{e} siècle avec le développement de la théorie de la complexité. La notation de Landau $\mathcal{O}$ est utilisée pour donner une estimation de la vitesse d'exécution des algorithmes. Un algorithme est de *complexité* $\mathcal{O}(f(n))$, où f est une fonction mathématique, s'il existe une constante c telle que le nombre d'opérations nécessaires pour exécuter l'algorithme sur une entrée de taille n est inférieure à $c \times f(n)$.

Un algorithme est *efficace* ou *polynomial* si la fonction f est un polynôme fonction de n, c'est-à-dire s'il existe un entier k tel que l'algorithme est de complexité $\mathcal{O}(n^k)$. La classe des problèmes *NP-complet* [32] est une classe de problèmes pour lesquels s'il est possible de résoudre l'un d'entre eux en temps polynomial, alors il est possible de résoudre tous les autres. Il est conjecturé qu'il n'existe pas d'algorithme polynomial permettant de résoudre un problème NP-complet.

La complexité d'un algorithme polynomial dont l'entrée est un graphe s'exprimera sous la forme $\mathcal{O}(P(n,m))$, où P est un polynôme fonction du nombre n de sommets et du nombre m d'arêtes. Le nombre maximum d'arêtes d'un graphe sur n sommets est $\frac{n(n-1)}{2}$. Il est donc possible d'exprimer une complexité fonction de n et m uniquement en fonction de n : un algorithme de complexité $\mathcal{O}(P(n,m))$ est aussi de complexité $\mathcal{O}(P(n,n^2))$.

Voici un exemple d'algorithme sur les graphes : le *parcours en largeur* ou BFS (pour *Breadth-First Search*). L'algorithme BFS construit un ordre particulier sur les sommets du graphe en utilisant une file. Une *file* est une structure de données dans laquelle il est possible d'ajouter et de retirer des éléments un par un. Lorsqu'un élément est retiré d'une file, c'est toujours l'élément le plus ancien qui est retiré (celui qui y a été mis le plus tôt). Voici une description formelle de l'algorithme BFS.

ALGORITHME BFS
ENTRÉE : Un graphe G sur n sommets et un sommet v de G.
SORTIE : Un ordre σ sur les sommets de G.
CALCUL :
- Soit F une file vide, ajouter v à F, soit $\sigma(v) = n$ et $i = n - 1$;
- tant que F est non vide :
  - retirer le sommet a de F qui maximise σ ;
  - pour chaque voisin non-numéroté b de a, ajouter b à F, poser $\sigma(b) = i$ et $i = i - 1$.

COMPLEXITÉ : $\mathcal{O}(n + m)$

Cet algorithme permet par exemple de détecter si un graphe est connexe. Pour cela il suffit de tester la valeur de i à la fin de l'algorithme. Si $i = 0$, tous les sommets du graphe ont été numérotés et le graphe est connexe. Si $i > 0$, le graphe n'est pas connexe.

1.1.4 Coloration et graphes parfaits

Soient G un graphe et k un entier. Une *coloration* de G utilisant k couleurs est une partition de $V(G)$ en ensembles $S_1, \ldots, S_k$ tels que pour tout $1 \leq i \leq k$, le graphe $G[S_i]$ est un stable. Si $v \in S_i$, le sommet v a *reçu* la couleur i. Le *nombre chromatique* de G, noté $\chi(G)$, est le plus petit entier k tel que G possède une coloration en k couleurs. Une coloration de G utilisant $\chi(G)$ couleurs est dite *optimale*.

Décider si un graphe possède une coloration en k couleurs est un problème NP-complet [56]. Il n'existe donc sans doute pas d'algorithme polynomial permettant de colorier tous les graphes de manière optimale. Nous restreindrons donc notre recherche à des classes de graphes particulières.

Soit G un graphe. La taille d'une clique de taille maximum de G est notée $\omega(G)$. Il est clair que si un graphe G contient une clique de taille k, il faut au moins k couleurs pour le colorier. Donc, tout graphe G vérifie : $\omega(G) \leq \chi(G)$. Il est naturel de se demander s'il y a égalité entre le nombre chromatique et la taille d'une plus grande clique. Ce n'est pas le cas et le plus

petit contre-exemple est C_5 : $\chi(C_5) = 3$ et $\omega(C_5) = 2$. Plus généralement, n'importe quel trou impair vérifie $\chi(C_{2k+1}) = 3$ et $\omega(C_{2k+1}) = 2$. Ghouila-Houri [33] a également remarqué que n'importe quel antitrou impair vérifie $\chi(\overline{C_{2k+1}}) = k+1$ et $\omega(\overline{C_{2k+1}}) = k$. Donc, l'égalité $\chi = \omega$ est fausse en général, mais il est intéressant de se demander pour quels graphes elle demeure exacte. Dans un graphe G, une coloration de G et une clique de G sont dites *de même taille* si la taille de la clique est exactement égale au nombre de couleur de la coloration, dans ce cas, la clique est de taille $\omega(G)$ et la coloration utilise $\chi(G)$ couleurs.

En 1960, Berge [2] a défini la classe des graphes *parfaits* comme étant la classe des graphes G tels que tout sous-graphe induit H de G vérifie $\chi(H) = \omega(H)$. Cette classe de graphes est apparue comme étant une classe de graphes assez générale sur laquelle le problème de la coloration pouvait être résolu efficacement. En 1984, Gröstchel, Lovász et Schrijver, [35] ont donné un algorithme polynomial qui trouve, pour tout graphe parfait, une coloration et une clique de même taille. Déterminer le nombre chromatique d'un graphe est NP-complet en général, mais devient donc polynomial sur la classe des graphes parfaits.

Les recherches ayant conduit à l'algorithme de Gröstchel, Lovász et Schrijver, ont joué un rôle central dans le développement de branches importantes de l'optimisation, comme la programmation semi-définie (voir [76]). Cet algorithme utilise la méthode des ellipsoïdes de Kachiyan [57] qui est très difficile à implémenter en raison de problèmes d'instabilité numérique. Par ailleurs, du point de vue de la théorie des graphes, cet algorithme n'est pas complètement satisfaisant. Il serait préférable d'avoir un algorithme qui tire parti des propriétés combinatoires des graphes. Il est donc considéré que la recherche d'un algorithme purement combinatoire permettant de colorier les graphes parfaits de manière efficace est une question ouverte. Cette thèse se place précisément dans le cadre de cette question.

Les trous impairs et les antitrous impairs ne sont pas des graphes parfaits. Ainsi, dès qu'un graphe contient l'un d'eux, il n'est pas non plus parfait. Les graphes sans trou impair et sans antitrou impair sont donc la plus grande classe de graphes dont il est possible d'espérer la perfection. Un graphe est dit *de Berge* si et seulement s'il ne contient ni trou impair, ni antitrou impair.

Au début des années 1960, Berge a énoncé deux conjectures célèbres [3]. La première, dite *conjecture faible des graphes parfaits*, a été étudiée par Fulkerson [30], puis finalement démontrée par Lovász en 1971 [61, 62]. C'est ce que nous appelons maintenant le *théorème des graphes parfaits* :

Théorème 1.1 ([61, 62]) *Un graphe est parfait si et seulement si son complémentaire est parfait.*

La deuxième conjecture de Berge, dite *conjecture forte des graphes parfaits*, a fait l'objet de nombreuses recherches avant d'être finalement démontrée par Chudnovsky, Robertson, Seymour et Thomas en 2002 [14]. C'est ce que nous appelons maintenant le *théorème fort des graphes parfaits* :

Théorème 1.2 ([14]) *Un graphe est parfait si et seulement s'il est de Berge.*

Le problème de la reconnaissance des graphes parfaits a lui aussi été résolu. En 2002, Chudnovsky, Cornuéjols, Liu, Seymour et Vušković [13] ont donné un algorithme de complexité $\mathcal{O}(n^9)$ permettant de reconnaître les graphes parfaits.

1.2 Quelques algorithmes de coloration

Dans cette partie, quelques algorithmes de coloration sont présentés, ils seront utilisés directement ou indirectement dans la suite de cette thèse. Chaque algorithme est illustré par un exemple de graphe pour lequel la coloration obtenue peut ne pas être optimale. Le graphe $\overline{P_6}$ est utilisé à plusieurs reprises comme contre-exemple (voir figure 1.5). Dans le chapitre 3, nous verrons un algorithme qui permet de colorier de manière optimale la classe des graphes contenant ni trou impair, ni antitrou, ni taureau. Cette classe contient le graphe $\overline{P_6}$.

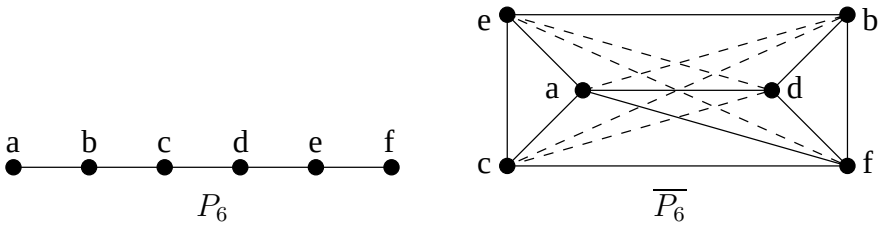


FIG. 1.5 – Le chemin sans corde à six sommets et son complémentaire

1.2.1 L'algorithme COLOR

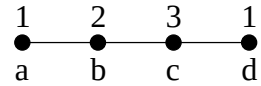
L'algorithme COLOR est l'algorithme le plus naturel permettant de colorier les sommets d'un graphe. Étant donné un ordre sur les sommets du graphe, cet algorithme parcourt les sommets selon cet ordre et donne à chaque sommet la plus petite couleur non attribuée à ses voisins. Voici une description formelle de l'algorithme.

ALGORITHME COLOR
ENTRÉE : Un graphe G avec n sommets et un ordre σ sur ses sommets.
SORTIE : Une coloration des sommets de G.
CALCUL :
Pour $i = 1, \ldots, n$:
- choisir le sommet non colorié x qui est minimum pour σ ;
- colorier x avec la plus petite couleur qui n'est pas présente dans son voisinage.

COMPLEXITÉ : $\mathcal{O}(n + m)$.

Il est facile de voir que pour tout graphe, il existe un ordre tel que l'algorithme COLOR appliqué à celui-ci donne une coloration optimale. Un tel ordre peut être obtenu à partir d'une coloration optimale en ordonnant les sommets par ordre croissant de leur couleur.

La figure 1.6 donne un exemple d'ordre des sommets de P_4 tel que la coloration obtenue par l'algorithme COLOR n'est pas optimale.



Ordre σ : a $<$ d $<$ b $<$ c

FIG. 1.6 – Exemple d'exécution de l'algorithme COLOR

Soient G un graphe et σ un ordre sur les sommets de G. L'ordre σ est dit *parfait* si, pour tout sous-graphe induit H de G, la coloration obtenue par l'algorithme COLOR sur le graphe H et l'ordre σ restreint à H est optimale.

Un graphe G est dit *parfaitement ordonnable* s'il possède un ordre parfait. Chvátal [16] a montré que cette définition est équivalente au fait qu'il n'existe pas de P_4 induit de G, noté v_1-v_2-v_3-v_4, tel que $v_1 < v_2$ et $v_4 < v_3$.

1.2.2 L'algorithme LEXBFS−COLOR

L'algorithme LEXBFS (Lexicographic Breadth-First Search) de Rose, Tarjan et Lueker [78] est un algorithme de complexité $\mathcal{O}(n + m)$ qui a été défini dans le but de reconnaître les graphes triangulés. Un graphe est dit *triangulé* s'il ne contient ni trou, ni carré.

LEXBFS parcourt les sommets d'un graphe et les numérote un par un de n à 1. Lors de l'étape générale, chaque sommet non numéroté a une étiquette, correspondant à l'ensemble des numéros de ses voisins déjà numérotés. Un ordre lexicographique est défini sur les étiquettes : l'étiquette $L(a)$ est *strictement plus grande* que l'étiquette $L(b)$ s'il existe un entier $i \in L(a) \setminus L(b)$

tel que $\forall j > i$, $j \in L(a) \cap L(b)$ ou $j \notin L(a) \cup L(b)$. Le prochain sommet qui est numéroté est le sommet dont l'étiquette est maximale selon l'ordre lexicographique. Voici une description formelle de l'algorithme.

ALGORITHME LEXBFS
ENTRÉE : Un graphe G.
SORTIE : Un ordre σ sur les sommets de G.
CALCUL :
- Pour tout sommet a de G, soit $L(a) = \emptyset$;
- pour $i = n, \ldots, 1$:
  - choisir un sommet a d'étiquette maximale et poser $\sigma(a) = i$;
  - pour chaque voisin non numéroté v de a, ajouter i à $L(v)$.

COMPLEXITÉ : $\mathcal{O}(n + m)$.

Soit G un graphe. Un sommet v de G est dit *simplicial* si et seulement si $N(v)$ induit une clique de G. Un *ordre d'élimination simplicial* est un ordre $(v_1, \ldots, v_n)$ des sommets de G tel que, pour $1 \leq i \leq n$, le sommet v_i est simplicial dans le graphe $G[v_1, \ldots, v_i]$. Rose, Tarjan et Lueuker [78] ont montré qu'un graphe est triangulé si et seulement si l'ordre produit par l'algorithme LEXBFS est un ordre d'élimination simplicial.

L'algorithme LEXBFS−COLOR désigne l'algorithme COLOR appliqué à l'ordre inverse de celui obtenu par l'algorithme LEXBFS. Une coloration obtenue par l'algorithme LEXBFS−COLOR sur la classe des graphes triangulés est optimale. Roussel et Rusu [80] ont généralisé ce résultat en démontrant l'optimalité sur la classe des graphes i-triangulés. Un graphe est dit *i-triangulé* si tout cycle impair de longueur supérieure ou égale à cinq contient au moins deux cordes non-croisées.

Le graphe $\overline{P_6}$ est un exemple de graphe sur lequel la coloration obtenue par l'algorithme LEXBFS−COLOR n'est pas optimale (voir figures 1.7 et 1.8).

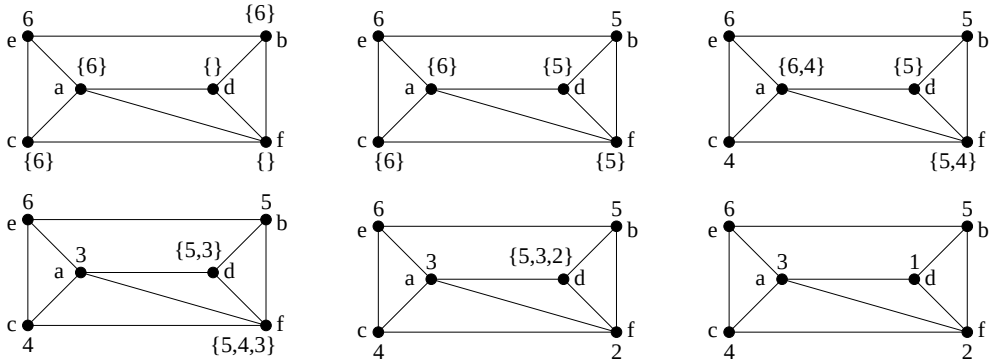
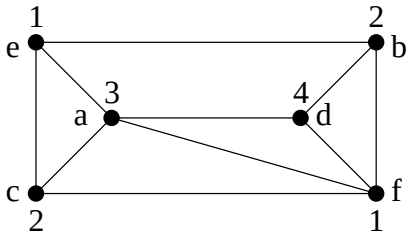


FIG. 1.7 – Exemple d'exécution de l'algorithme LEXBFS



Ordre inverse obtenu par LexBFS (voir figure 1.7) : e < b < c < a < f < d

FIG. 1.8 – Exemple d'exécution de l'algorithme LEXBFS−COLOR

1.2.3 L'algorithme COSINE

L'algorithme COSINE de Hertz [46] est un algorithme de complexité $\mathcal{O}(nm)$ qui a été défini dans le but de colorier les graphes de Meyniel [45]. Un graphe est dit de *Meyniel* si tout cycle impair de longueur supérieure ou égale à cinq contient au moins deux cordes. Les graphes de Meyniel sont exactement les graphes ne contenant ni trou impair, ni maison.

Cet algorithme construit itérativement les classes de couleurs. Pour construire la classe de couleur c, l'algorithme sélectionne des sommets jusqu'à ce que tous les sommets du graphe aient un voisin colorié c. A chaque étape, le sommet colorié c est un sommet qui n'a pas de voisin colorié c et qui maximise le nombre de voisins en commun avec les sommets déjà coloriés c. Voici une description formelle de l'algorithme.

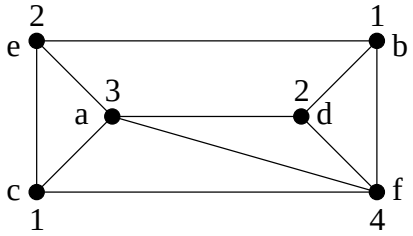
ALGORITHME COSINE
ENTRÉE : Un graphe G.
SORTIE : Une coloration des sommets de G.
CALCUL :
- Soit $c = 1$;
- tant qu'il existe des sommets non coloriés :
  - s'il existe un sommet non colorié qui n'a pas de voisins coloriés c :
    - soit A l'ensemble des sommets non coloriés qui ont un voisin colorié c ;
    - choisir un sommet non colorié v qui n'a pas de voisin colorié c et qui a le nombre maximum de voisins dans A ;
    - colorier v avec la couleur c ;
  - sinon, poser $c = c + 1$.

COMPLEXITÉ : $\mathcal{O}(mn)$.

Cette formulation de l'algorithme COSINE diffère de la description origi-

nale de Hertz [45]. Hertz explique son algorithme en terme de contraction de sommets, mais il nous paraît plus simple de ne pas parler de cette notion dans la description de l'algorithme.

Le graphe $\overline{P_6}$ est un exemple de graphe sur lequel la coloration obtenue par l'algorithme COSINE n'est pas optimale (voir figure 1.9).



Ordre de coloration : c, b, d, e, a, f,

FIG. 1.9 – Exemple d'exécution de l'algorithme COSINE

1.2.4 L'algorithme LEXCOLOR

L'algorithme LEXCOLOR de Roussel et Rusu [81], de complexité $\mathcal{O}(n^2)$, a lui aussi été défini dans le but de colorier les graphes de Meyniel.

Cet algorithme associe des étiquettes $label_x$ de longueur n à chaque sommet x de G. Pour chaque couleur $1 \leq c \leq n$, si x n'a pas de voisin colorié c, alors $label_x(c)$ est égal à 0 ; si x a un voisin colorié c, alors $label_x(c)$ est égal à l'entier i tel que le premier voisin de x colorié c est le $(n\text{-}i)$-ième sommet colorié du graphe. Un ordre lexicographique (inverse) sur les étiquettes est défini de la manière suivante : $label_x <_{Lex} label_y$ si et seulement s'il existe une couleur c telle que $label_x(c) < label_y(c)$ et $\forall\, c' > c$, $label_x(c') = label_y(c')$.

A chaque étape, l'algorithme sélectionne un sommet non colorié dont l'étiquette est maximale pour l'ordre lexicographique, lui donne la plus petite couleur non présente dans son voisinage et itère ce procédé jusqu'à ce que tous les sommets soient coloriés. Voici une description formelle de l'algorithme.

ALGORITHME LEXCOLOR
ENTRÉE : Un graphe G avec n sommets.
SORTIE : Une coloration des sommets de G.
CALCUL :
- Pour tout sommet x et toute couleur c, soit $label_x(c) = 0$;
- pour $i = 1, \ldots, n$:
  - choisir un sommet non colorié x qui maximise $label_x$ pour $<_{Lex}$;
  - colorier x avec la plus petite couleur c qui n'est pas présente dans son voisinage ;

– pour tout voisin non colorié y de x, si $label_y(c) = 0$, poser $label_y(c) = n - i$.

COMPLEXITÉ : $\mathcal{O}(n^2)$.

Cette version de LEXCOLOR diffère un peu de la version originale de Roussel et Rusu [81]. Lorsque x a un voisin colorié c, l'entier $label_x(c)$ a été défini dans [81] comme étant l'entier i tel que le premier voisin de x colorié c soit le (n-i)-ième sommet colorié avec la couleur c (à la place du (n-i)-ième sommet colorié de couleur quelconque). Pour une couleur c, l'ordre entre les $label_x(c)$ de chaque sommet est le même dans les deux versions de l'algorithme, donc l'ordre lexicographique est le même et il n'y a pas de différence entre les deux exécutions de l'algorithme. Cette modification permet juste de simplifier la description de l'algorithme.

Le graphe $\overline{P_6}$ est un exemple de graphe sur lequel la coloration obtenue par l'algorithme LEXCOLOR n'est pas optimale (voir figure 1.10).

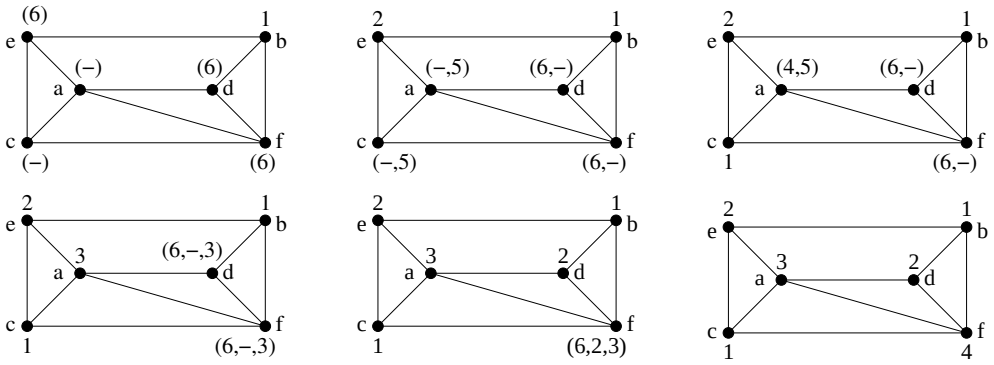


FIG. 1.10 – Exemple d'exécution de l'algorithme LEXCOLOR

Chapitre 2

Contraction

Ce chapitre concerne l'opération de contraction de paire d'amis qui permet de colorier plusieurs sous-classes de graphes parfaits. Une généralisation de la notion de paires d'amis, ainsi qu'une conjecture à ce sujet, sont proposées. Selon la manière dont la contraction est effectuée, plusieurs propriétés intéressantes peuvent apparaître. Ces propriétés seront utilisées dans les chapitres 3 et 4.

2.1 Paire d'amis et paire P_4-libre

Etant donnés deux sommets x, y de G, l'opération de *contraction* consiste à retirer x et y, puis à ajouter un nouveau sommet noté xy relié à chaque sommets de $N(x) \cup N(y)$. Le graphe obtenu est noté G/xy.

Lemme 2.1 *Soit $\{x, y\}$ une paire de sommets non-adjacents dans un graphe G, alors $\omega(G) \leq \omega(G/xy)$ et $\chi(G) \leq \chi(G/xy)$.*

Preuve. Soit Q une clique de G. Dans ce cas, la clique Q contient au plus l'un de x, y. Si Q ne contient ni x, ni y, l'ensemble Q est aussi une clique de G/xy. Si Q contient exactement l'un de x, y, alors en remplaçant ce sommet par xy, nous obtenons une clique de G/xy de même taille. Donc $\omega(G) \leq \omega(G/xy)$.

A partir d'une coloration de G/xy, une coloration de G, utilisant le même nombre de couleurs, est obtenue en donnant à x et y la couleur du sommet xy, et en ne changeant pas la couleur des autres sommets. Donc $\chi(G) \leq \chi(G/xy)$. □

Lemme 2.2 *Soit G un graphe. S'il existe une suite de graphes $G_0, \ldots, G_k$ telle que $G = G_0$, pour $1 \leq i \leq k$, le graphe G_i est obtenu à partir du graphe*

G_{i-1} par une contraction de paire de sommets non-adjacents, et G_k est une clique. Alors $\chi(G) \leq |G_k|$.

Preuve. D'après le lemme 2.2, nous avons $\chi(G) \leq \chi(G_k) = |G_k|$. □

Une *paire d'amis* (resp. une *paire P_4-libre*) est une paire de sommets non-adjacents tels qu'il n'existe pas de chemin sans corde les reliant, de longueur impaire (resp. de longueur 3). Remarquons qu'une paire d'amis est une paire P_4-libre.

La notion de paire d'amis a été introduite par Meyniel [70] et largement étudiée ensuite [28]. Les paires P_4-libre sont apparues dans la littérature de manière plus implicite. Les définitions et les résultats de cette thèse qui y font référence sont nouveaux. Certains de ces résultats sont des conséquences directes de preuves sur les paires d'amis, dans lesquelles les auteurs utilisent uniquement le fait qu'il n'y a pas de P_4 entre les sommets considérés, dans ce cas une référence à l'article d'origine est mentionnée.

Lorsqu'il s'agira de montrer qu'une contraction vérifie une propriété, nous montrerons, si possible, que cette propriété est vérifiée lors d'une contraction de paire P_4-libre. Alors, cette propriété sera en particulier vraie pour la notion plus restrictive de paire d'amis. A l'inverse, lorsqu'il s'agira de contracter un graphe, nous montrerons, si possible, qu'il s'agit d'une contraction de paire d'amis, cette contraction étant aussi une contraction de paire P_4-libre.

Fonlupt et Uhry [29] ont montré que la contraction de paire d'amis préserve la taille maximum d'une clique. La preuve de ce résultat utilise seulement le fait que la paire de sommets considérée est une paire P_4-libre. Nous en déduisons donc le lemme suivant :

Lemme 2.3 ([29]) *Soit $\{x, y\}$ une paire P_4-libre dans un graphe G, alors $\omega(G) = \omega(G/xy)$.*

Preuve. D'après le lemme 2.1, nous avons $\omega(G) \leq \omega(G/xy)$. Inversement, soit Q une clique de G/xy de taille $\omega(G/xy)$. Si Q ne contient pas le sommet xy, alors Q est aussi une clique de G. Si Q contient le sommet xy, alors $Q\backslash\{xy\}$ est une clique de G. L'un de x, y est adjacent à tout Q, sinon Q contiendrait deux sommets a, b tels que a voit x et pas y, b voit y et pas x, alors le chemin x-a-b-y est un chemin sans corde de longueur 3 reliant x à y, une contradiction. Donc G contient une clique de taille $\omega(G/xy)$ et $\omega(G/xy) \leq \omega(G)$. □

Fonlupt et Uhry [29] ont aussi montré que la contraction de paire d'amis préserve le nombre chromatique. Ce résultat n'est pas vrai pour la contraction d'une paire P_4-libre : la contraction des deux extrémités d'un P_5 donne un C_5 alors que $\chi(P_5) = 2$ et $\chi(C_5) = 3$.

Lemme 2.4 ([29]) *Soit $\{x, y\}$ une paire d'amis dans un graphe G, alors $\chi(G) = \chi(G/xy)$.*

Preuve. D'après le lemme 2.1, nous avons $\chi(G) \leq \chi(G/xy)$. Inversement, considérons une coloration de G utilisant $\chi(G)$ couleurs. Si x et y ont reçu la même couleur, alors nous en déduisons une coloration de G/xy utilisant le même nombre de couleurs. Supposons maintenant que x a reçu la couleur 1 et y la couleur 2. Soit H le sous-graphe biparti induit par les sommets de couleurs 1 et 2. Les sommets x et y sont dans deux composantes connexes différentes de H, sinon il existerait un chemin sans corde impair les reliant. Nous pouvons échanger les couleurs 1 et 2 dans la composante connexe de H contenant x sans affecter y. Nous obtenons une coloration de G dans laquelle x et y ont reçu la même couleur, nous en déduisons une coloration de G/xy. Donc $\chi(G/xy) \leq \chi(G)$. □

Pour un graphe quelconque, le problème de décider si une paire de sommets n'est pas une paire d'amis est un problème NP-complet [6], mais il devient polynomial sur la classe des graphes parfaits. Soient G un graphe parfait et $\{x, y\}$ une paire de sommets non-adjacents de G. Soit G' le graphe obtenu à partir de G en ajoutant un nouveau sommet adjacent à x et y. La paire $\{x, y\}$ est une paire d'amis de G si et seulement si G' est un graphe parfait. L'algorithme de reconnaissance des graphes parfaits [13] permet donc de détecter si, dans un graphe parfait, deux sommets forment une paire d'amis en temps $\mathcal{O}(n^9)$.

La détection de paire P_4-libre est beaucoup plus simple. Il est possible de vérifier en temps $\mathcal{O}(n^2)$ si, dans un graphe quelconque, deux sommets non-adjacents forment une paire P_4-libre. Il suffit de considérer tous les couples de sommets et de vérifier s'ils sont le milieu d'un P_4 reliant les deux sommets testés.

Un graphe est dit *contractile* (resp. *P_4-libre-contractile*) si c'est une clique, ou s'il contient une paire d'amis (resp. une paire P_4-libre) dont la contraction donne un graphe contractile (resp. P_4-libre-contractile). La classe des graphes contractiles a été définie par Bertschi [5]. Remarquons qu'un graphe contractile est P_4-libre-contractile.

Lemme 2.5 *Soit G un graphe P_4-libre-contractile, alors $\chi(G) = \omega(G)$*

Preuve. Il existe une suite $G_0, \ldots, G_k$ de graphes telle que $G = G_0$, pour $1 \leq i \leq k$, le graphe G_i est obtenu à partir du graphe G_{i-1} par une contraction de paire P_4-libre, et G_k est une clique. Le lemme 2.3 appliqué successivement à cette suite de graphes implique que $\omega(G) = \omega(G_k)$. Le lemme 2.2 implique

que $\chi(G) \leq |G_k| \leq \omega(G_k)$. Donc $\omega(G) \leq \chi(G) \leq \omega(G_k) = \omega(G)$ et par conséquent $\chi(G) = \omega(G)$. □

La preuve du lemme 2.5 donne un algorithme permettant d'obtenir une coloration et une clique de même taille pour tout graphe P_4-libre-contractile, à condition de savoir contracter le graphe jusqu'à obtenir une clique. C'est l'existence de cette méthode qui est à l'origine de la définition des graphes contractiles par Bertschi [5]. Les graphes P_4-libre-contractiles étant plus généraux, nous pensons que c'est la notion de paire P_4-libre et non pas de paire d'amis qui aurait dû être à la base de la définition de Bertschi. Comme ce n'est pas le cas, nous jonglerons avec ces deux notions tout le long de ce chapitre.

Un graphe G est dit *parfaitement contractile* (resp. *parfaitement P_4-libre-contractile*) si tout sous-graphe induit de G est contractile (resp. P_4-libre-contractile). La classe des graphes parfaitement contractiles a elle aussi été définie par Bertschi [5]. Remarquons qu'un graphe parfaitement contractile est parfaitement P_4-libre-contractile.

Lemme 2.6 *Un graphe parfaitement P_4-libre-contractile est un graphe parfait.*

Preuve. Soit G un graphe parfaitement P_4-libre-contractile. Par définition, tout sous-graphe H de G est P_4-libre-contractile. D'après le lemme 2.5, le sous-graphe H vérifie $\chi(H) = \omega(H)$. Donc G est un graphe parfait. □

Les définitions des graphes parfaitement contractiles et parfaitement P_4-libre-contractiles ne nous permettent pas de savoir comment contracter ces graphes, ni de décider si un graphe quelconque est dans l'une de ces classes. Plusieurs résultats et conjectures seront formulés à ce propos dans la suite de ce chapitre. Auparavant, voici une série de lemmes qui seront utilisés pour montrer qu'après avoir contracté une paire de sommets, le graphe contracté conserve certaines propriétés.

Lemme 2.7 *Soit $\{x, y\}$ une paire P_4-libre dans un graphe G.*

(i) Si G ne contient pas d'antitrou, alors G/xy ne contient pas d'antitrou autre que C_5 ou $\overline{C}_6$.

(ii) Si G ne contient pas d'antitrou impair, alors G/xy ne contient pas d'antitrou impair autre que C_5.

Preuve. Supposons que G/xy contient un antitrou $A \geq 7$. Si A ne contient pas xy, alors A est aussi un antitrou de G. Nous pouvons donc supposer que A contient xy. Notons $a_0 = xy, a_1, \ldots, a_k$, avec $k \geq 6$, les sommets de A de

sorte que a_i manque a_j si et seulement si $|i-j| = 1 \bmod k+1$. Dans G, les sommets a_1, a_k ne voient ni x, ni y. Soit $\overline{X} = A\backslash(N(x) \cup \{a_1, a_k, xy\})$ et $\overline{Y} = A\backslash(N(y) \cup \{a_1, a_k, xy\})$. Chaque sommet de $a_2, \ldots, a_{k-1}$ voit au moins l'un de x ou y, donc $\overline{X} \subseteq N(y)$,$\overline{Y} \subseteq N(x)$. Pour tout $a \in \overline{X}$, $b \in \overline{Y}$, nous avons a manque b, car sinon x, b, a, y induit un P_4 entre x et y. Donc $\overline{X} \cup \overline{Y}$ est constitué d'un ensemble de sommets consécutifs de A d'au plus 3 sommets. Puisque $k \geq 6$, nous pouvons supposer que $a_2 \notin (\overline{X} \cup \overline{Y})$. Si $\overline{X} = \emptyset$, alors $A \cup \{x\}\backslash\{xy\}$ est un antitrou de G de même taille que A. Si $\overline{Y} = \emptyset$, alors $A \cup \{y\}\backslash\{xy\}$ est un antitrou de G de même taille que A. Soit i le plus petit entier tel que $a_i \in (\overline{X} \cup \overline{Y})$. Nous avons $i \geq 3$ et nous pouvons supposer par exemple que $a_i \in \overline{X}$ et $a_{i+1} \in \overline{Y}$. Si i est impair, alors $y, a_1, \ldots, a_{i+1}$ est un antitrou impair de G. Si i est pair, alors $i \geq 4$ et $x, a_1, \ldots, a_i$ est un antitrou impair de G. □

La preuve du lemme montre en fait un résultat un peu plus fort. Si, dans un graphe G, une contraction de paire P_4-libre crée un antitrou A de taille ≥ 7 qui n'est pas déjà présent dans G, alors, quelle que soit la parité de A, le graphe G contient un antitrou impair.

Lemme 2.8 *Soit $\{x, y\}$ une paire P_4-libre dans un graphe G. Si G ne contient pas de triangle, alors G/xy ne contient pas de triangle.*

Preuve. Supposons que G/xy contient un triangle T. Si T ne contient pas xy, alors T est aussi un triangle de G. Si T contient xy, alors $(T \cup \{x, y\})\backslash\{xy\}$ est un P_4 entre x et y dans G. □

Lemme 2.9 ([28]) *Soit $\{x, y\}$ une paire d'amis dans un graphe G. Si G ne contient pas de trou impair, alors G/xy ne contient pas de trou impair.*

Preuve. Supposons que G/xy contient un trou impair H. Si H ne contient pas xy, alors H est aussi un trou impair de G. Si H contient xy, alors soit $(H \cup \{x\})\backslash\{xy\}$, soit $(H \cup \{y\})\backslash\{xy\}$ est un trou impair de G, car sinon $H \cup \{x, y\})\backslash\{xy\}$ est un chemin impair sans corde entre x et y dans G. □

2.2 Sous-graphes exclus

Lorsque nous nous intéressons à une classe de graphes définie comme ayant une certaine propriété, nous cherchons souvent à caractériser cette classe en terme de sous-graphes exclus. C'est par exemple ce que proposait la

conjecture forte des graphes parfaits due à Berge et résolu par Chudnovsky, Robertson, Seymour et Thomas. Une conjecture similaire existe pour les graphes parfaitement contractiles et nous en proposons une nouvelle pour les graphes parfaitement P_4-libre-contractiles. Ces conjectures sont toujours ouvertes.

Commençons par exhiber quelques familles de graphes qui ne sont pas dans les classes considérées.

Lemme 2.10 *Un antitrou n'est pas P_4-libre-contractile*

Preuve. Dans un antitrou, toute paire de sommets non-adjacents est reliée par un P_4. Il n'y a donc pas de paire P_4-libre. □

Lemme 2.11 *Un trou impair n'est pas P_4-libre-contractile*

Preuve. Supposons qu'il existe un entier k impair tel qu'il existe une séquence de contractions de paires P_4-libre menant d'un trou de taille k à une clique Q. Un trou ne contient pas de triangle, donc d'après le lemme 2.8, la clique non plus et $|Q| \leq 2$. D'après le lemme 2.2, $\chi(C_k) \leq |Q| \leq 2$, ce qui contredit $\chi(C_k) = 3$. □

Lemme 2.12 ([60],[28]) *Un prisme impair n'est pas contractile*

Les prismes impairs ne sont pas contractiles. Par contre ils sont P_4-libre-contractiles, excepté l'antitrou $\overline{C_6}$. Le graphe de la figure 2.1 montre comment contracter des paires P_4-libre dans le prisme impair à huit sommets pour aboutir à une clique. A chaque étape, la paire de sommets contractée est représentée en blanc. Cette méthode s'adapte facilement pour contracter les prismes impairs de plus grande taille.

Reed a proposé d'appeler graphes de *Grenoble* les graphes contenant ni trou impair, ni antitrou, ni prisme impair. Les lemmes 2.10, 2.11 et 2.12 permettent de déduire le corollaire suivant concernant les graphes parfaitement contractiles.

Corollaire 2.13 ([28]) *Un graphe parfaitement contractile est un graphe de Grenoble.*

La conjecture des graphes parfaitement contractiles, formulée par Everett et Reed [28, 75], affirme que la réciproque du corollaire 2.13 est vraie.

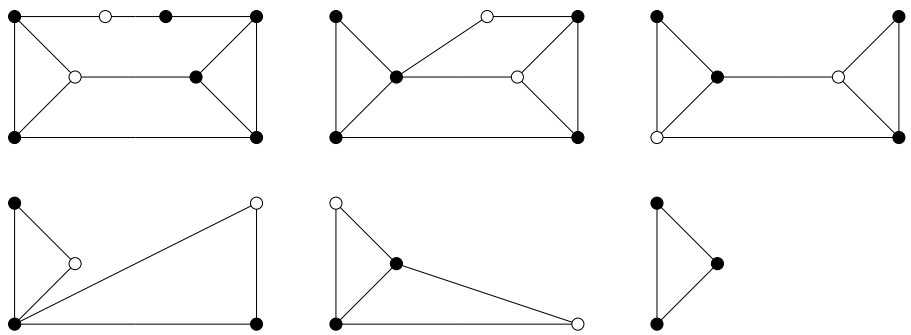


FIG. 2.1 – Exemple de contraction du prisme impair à huit sommets

Conjecture 2.14 ([75]) *Un graphe est parfaitement contractile si et seulement s'il est un graphe de Grenoble.*

Everett et Reed [75] ont de plus conjecturé qu'il est possible de contracter les graphes parfaitement contractiles en restant dans cette classe.

Conjecture 2.15 ([75]) *Tout graphe parfaitement contractile qui n'est pas une clique contient une paire d'amis dont la contraction laisse le graphe parfaitement contractile.*

Adoptons maintenant l'idée que la notion de paire P_4-libre devrait se substituer à celle de paire d'amis dans la définition des graphes parfaitement contractiles. La classe des graphes parfaitement P_4-libre-contractiles est la classe des graphes sur laquelle la méthode de coloration par contraction est possible pour tout sous-graphe induit. Les lemmes 2.10 et 2.11 permettent de déduire un corollaire analogue au corollaire 2.13 pour les graphes parfaitement P_4-libre-contractiles.

Corollaire 2.16 *Un graphe parfaitement P_4-libre-contractile ne contient ni trou impair, ni antitrou.*

Nous formulons une conjecture analogue à la conjecture 2.14 pour la classe des graphes parfaitement P_4-libre-contractiles. Nous conjecturons que la réciproque du corollaire 2.16 est vraie.

Conjecture 2.17 *Un graphe est parfaitement P_4-libre-contractile si et seulement s'il ne contient ni trou impair, ni antitrou.*

La paire d'amis est considérée comme un cas particulier de paire P_4-libre qui permettrait peut-être de contracter les graphes de Grenoble en restant dans cette classe. Les conjectures 2.14 et 2.15 d'Everett et Reed deviennent :

Conjecture 2.18 ([75]) *Tout graphe de Grenoble qui n'est pas une clique contient une paire d'amis dont la contraction donne un graphe de Grenoble.*

Le résultat s'approchant le plus de cette conjecture est dû à Maffray et Trotignon [66]. Reed a proposé d'appeler graphes d'*Artémis* les graphes ne contenant ni trou impair, ni antitrou, ni prisme. Une paire d'amis $\{a, b\}$ dans un graphe G est dite *spéciale* si le graphe G/ab ne contient pas de prisme. Maffray et Trotignon [66] ont montré que cette notion de paire d'amis spéciale est un cas particulier de paire d'amis qui permet de contracter les graphes d'Artémis en restant dans cette classe :

Théorème 2.19 ([66]) *Tout graphe d'Artémis qui n'est pas une clique contient une paire d'amis spéciale dont la contraction donne un graphe d'Artémis.*

L'algorithme sous-jacent à la preuve du théorème 2.19 est simplifié et accéléré dans le chapitre 5.

Un théorème de ce type avait déjà été formulé pour une sous-classe des graphes d'Artémis. Un graphe est *faiblement triangulé* [37] s'il ne contient ni trou, ni antitrou. Une 2-*paire* est une paire de sommets non-adjacents telle que tous les chemins sans corde les reliant sont de longueur 2. Hayward, Hoàng et Maffray [41] ont montré que cette notion de 2-paire est un cas particulier de paire d'amis spéciale qui permet de contracter les graphes faiblement triangulés en restant dans cette classe :

Théorème 2.20 ([41]) *Tout graphe faiblement triangulé qui n'est pas une clique contient une 2-paire dont la contraction donne un graphe faiblement triangulé.*

Nous voyons donc que les paires P_4-libre définissent le cadre général dans lequel l'algorithme de contraction fonctionne. Les paires d'amis, les paires d'amis spéciales et les 2-paires sont des raffinements successifs de cette notion qui permettent de contracter certaines classes de graphes sans en sortir (ceci restant à prouver pour les graphes de Grenoble).

Il ne peut pas exister de résultat analogue à la conjecture 2.15 pour les graphes parfaitement P_4-libre-contractiles. En effet, une contraction de paire P_4-libre qui n'est pas une contraction de paire d'amis crée un trou impair. Les prismes impairs différents de $\overline{C_6}$ sont donc des exemples de graphes pour lesquels la contraction P_4-libre fonctionne, mais pour lesquels il est impossible de rester dans la classe des graphes parfaits. Le deuxième graphe de la figure 2.1 contient un C_5. Lors de la contraction des graphes parfaitement P_4-libre-contractiles, nous sommes donc sûr de sortir de cette classe. Cela

n'exclut pas le fait qu'il peut exister un algorithme polynomial permettant de contracter ces graphes. Il arrive le même genre de problème avec d'autres classes. Par exemple, Hertz [45] a été amené à définir les graphes de *quasi-Meyniel* pour parvenir à contracter les graphes de Meyniel. De manière analogue, il est envisageable de définir une classe de graphes quasi-parfaitement P_4-libre-contractiles contenant quelques trous impairs bien placés et de réaliser la contraction dans cette classe.

Un résultat de Chvátal, Rusu et Sritharan [18] nous conforte en faveur de cette possibilité. Un chemin sans corde v_1-$\cdots$-v_k est dit *simplicial* s'il ne s'étend pas à un chemin sans corde v_0-v_1-$\cdots$-v_k-v_{k+1}. Il est facile de voir que, étant donné un graphe G et un entier positif k, si tout sous-graphe non vide de G contient un chemin simplicial sur au plus k sommets, alors G ne contient pas de trou de taille $\geq k+3$. Chvátal, Rusu et Sritharan [18] ont montré la réciproque.

Théorème 2.21 ([18]) *Pour tout entier positif k, un graphe ne contient pas de trou de taille $\geq k+3$ si et seulement si chacun de ses sous-graphes non vides contient un chemin simplicial sur au plus k sommets.*

Appliqué pour $k=2$ dans le graphe complémentaire, cela donne le corollaire suivant.

Corollaire 2.22 *Tout graphe sans antitrou contient une paire P_4-libre.*

2.3 Ordre de contraction

Plusieurs classes de graphes sont connues pour être parfaitement contractiles, comme par exemple les graphes de Meyniel et les graphes faiblement triangulés, d'après des résultats de Hertz [45] et Hayward, Hoàng, Maffray [41]. Pour ces deux classes, une profonde différence apparaît dans la manière dont la contraction peut être effectuée. La contraction des graphes de Meyniel peut se faire à partir de n'importe quel sommet en le contractant jusqu'à ce qu'il devienne adjacent à tous les sommets du graphe. Cela est impossible pour les graphes faiblement triangulés. Après avoir contracté une paire d'amis dans un graphe triangulé, il n'est pas possible de garantir que la contraction puisse se poursuivre à partir du sommet contracté. Pour s'en persuader, le lecteur pourra tenter de contracter les graphes de la figure 2.2.

La suite de cette partie est consacrée à la définition formelle de cette propriété qui est vrai pour les graphes de Meyniel mais pas pour les graphes faiblement triangulés. Ces définitions sont nouvelles et auront plusieurs conséquences algorithmiques qui seront expliquées ensuite.

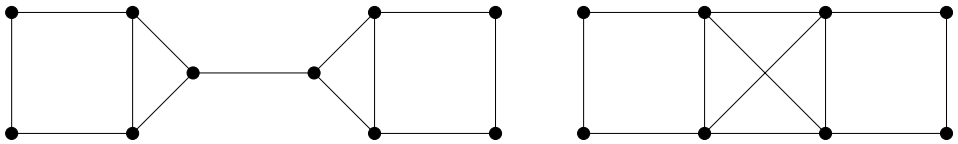


FIG. 2.2 – Graphes faiblement triangulés

Pour un graphe G, un *ordre de contraction* de G est une notation de ses sommets sous la forme $V(G) = \{x_1^1, x_1^2, x_1^3, \ldots, x_1^{k_1}, x_2^1, \ldots, x_2^{k_2}, \ldots, x_\ell^1, \ldots, x_\ell^{k_\ell}\}$ telle que, pour $1 \leq c \leq l$, les ensembles $S_c = \{x_c^1, \ldots, x_c^{k_c}\}$ sont des stables. La coloration qui, à chaque sommet x_c^i associe la couleur c, est appelée *coloration associée* à cet ordre, c'est aussi la coloration obtenue par l'algorithme COLOR sur cet ordre.

La *suite de contractions associée* à cet ordre est la suite de graphes G_c^i et les sommets w_c^i ($1 \leq c \leq \ell$, $1 \leq i \leq k_c$) définis de la manière suivante. Soit $G_1^1 = G$. Pour $2 \leq c \leq \ell$, soit $G_c^1 = G_{c-1}^{k_{c-1}}$. Pour $1 \leq c \leq \ell$, soit $w_c^1 = x_c^1$. Pour $1 \leq c \leq \ell$ et $2 \leq i \leq k_c$, soit G_c^i le graphe obtenu à partir de G_c^{i-1} en contractant les sommets w_c^{i-1} et x_c^i en un nouveau sommet w_c^i. Pour $1 \leq c \leq \ell$, le sommet $w_c^{k_c}$ est noté plus simplement w_c.

Un ordre de contraction est dit *amical* (resp. *P_4-libre*) si la suite de contraction associée vérifie, pour tout $1 \leq c \leq \ell$ et $2 \leq i \leq k_c$:

(i) les sommets w_c^{i-1} et x_c^i sont une paire d'amis (resp. une paire P_4-libre) de G_c^{i-1}

(ii) le sommet w_c est adjacent à tous les sommets de $G_c^{k_c}$

Une coloration est dite *amicale* (resp. *P_4-libre*) si elle est associée à un ordre de contraction amical (resp. P_4-libre). Remarquons qu'un ordre de contraction amical est P_4-libre et qu'une coloration amicale est P_4-libre.

Le théorème suivant montre qu'il est facile d'obtenir une coloration optimale à partir d'un ordre de contraction P_4-libre.

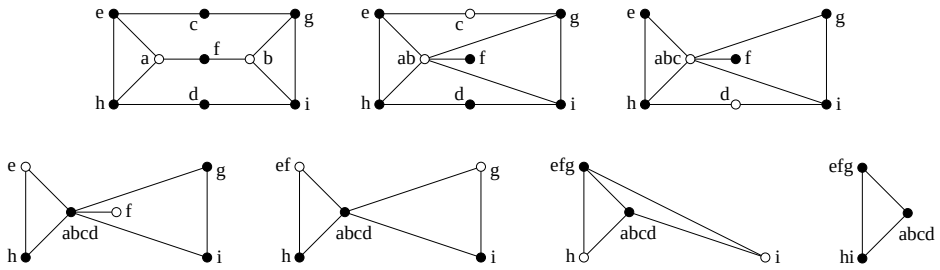
Théorème 2.23 *Soit G un graphe possédant un ordre de contraction P_4-libre $\{x_1^1, \ldots, x_1^{k_1}, \ldots, x_\ell^1, \ldots, x_\ell^{k_\ell}\}$. Alors $\chi(G) = \omega(G) = \ell$ et la coloration P_4-libre associée à cet ordre est une coloration optimale de G.*

Preuve. D'après (i), dans la séquence $G = G_1^1, \ldots, G_\ell^{k_\ell}$, chaque graphe est obtenu à partir de son prédécesseur par une contraction de paire P_4-libre. Le lemme 2.3 appliqué successivement à cette séquence de graphes implique que $\omega(G) = \omega(G_\ell^{k_\ell})$. D'autre part, (ii) implique que $G_\ell^{k_\ell}$ est une clique de taille ℓ, et le lemme 2.2 implique que $\chi(G) \leq |G_\ell^{k_\ell}| = \omega(G_\ell^{k_\ell})$. Donc $\omega(G) \leq \chi(G) \leq \omega(G_\ell^{k_\ell}) = \omega(G)$ et par conséquent $\chi(G) = \omega(G) = \ell$. □

Décider si un ordre de contraction n'est pas amical est un problème NP-complet pour un graphe quelconque et de complexité $\mathcal{O}(n^{10})$ pour un graphe parfait. Décider si un ordre de contraction est P_4-libre est de complexité $\mathcal{O}(n^3)$ pour un graphe quelconque.

Un graphe est dit *ordre-contractile* (resp. *P_4-libre-ordre-contractile*) s'il possède un ordre de contraction amical (resp. P_4-libre). Remarquons qu'un graphe ordre-contractile est contractile et P_4-libre-ordre-contractile et qu'un graphe P_4-libre-ordre-contractile est P_4-libre-contractile.

Les graphes de la figure 2.2 sont des exemples de graphes faiblement triangulés qui ne sont pas P_4-libre-ordre-contractiles. Inversement, les prismes pairs sont des graphes ordre-contractiles qui ne sont pas faiblement triangulés. La figure 2.3 donne un exemple d'ordre de contraction P_4-libre du prisme pair à neuf sommets.



Ordre de contraction amical : (a,b,c,d,e,f,g,h,i)

FIG. 2.3 – Exemple d'ordre de contraction du prisme pair à neuf sommets

Un graphe G est dit *parfaitement ordre-contractile* (resp. *parfaitement P_4-libre-ordre-contractile*) si tout sous-graphe induit de G est ordre-contractile (resp. P_4-libre-ordre-contractile). Remarquons qu'un graphe parfaitement ordre-contractile est parfaitement contractile et parfaitement P_4-libre-ordre-contractile et qu'un graphe parfaitement P_4-libre-ordre-contractile est parfaitement P_4-libre-contractile.

Dans [47], Hertz et de Werra démontre (sans le savoir) que les graphes parfaitement ordonnables sont parfaitement ordre-contractiles. Cette preuve donne un algorithme qui produit un ordre de contraction amical à partir d'un ordre parfait.

ALGORITHME ORDRECONTRACTION

ENTRÉE : Un graphe G et un ordre σ sur ses sommets.

SORTIE : Un ordre de contraction $\{x_1^1, \ldots, x_1^{k_1}, \ldots, x_\ell^1, \ldots, x_\ell^{k_\ell}\}$ et la coloration associée.

CALCUL :

- Soient $c = 1$ et $i = 1$;
- tant qu'il existe des sommets non coloriés :
  - s'il existe un sommet non colorié qui n'a pas de voisin colorié c :
    - choisir un tel sommet v minimum pour σ ;
    - colorier v avec la couleur c, poser $x_c^i = v$ et $i = i + 1$;
  - sinon, poser $c = c + 1$ et $i = 1$.

COMPLEXITÉ : $\mathcal{O}(m + n)$.

Les résultats de [48] peuvent être reformulés ainsi.

Théorème 2.24 ([48]) *Soient G un graphe parfaitement ordonnable et σ un ordre parfait de G. L'algorithme* OrdreContraction *appliqué à cette entrée produit un ordre de contraction amical de G.*

Nous en déduisons le corollaire suivant.

Corollaire 2.25 *Les graphes parfaitement ordonnables sont parfaitement ordre-contractiles.*

Déterminer si un graphe est parfaitement ordonnable est un problème NP-complet [71] et personne ne sait comment, étant donné un graphe parfaitement ordonnable, trouver un ordre parfait de ce graphe. Il n'est donc sûrement pas possible d'utiliser les idées présentées ici pour trouver, pour tout graphe parfaitement ordonnable, un ordre de contraction amical.

Nous verrons dans le chapitre 3 qu'il existe un algorithme simple et efficace qui produit un ordre de contraction amical pour une sous-classe de la classe des graphes parfaitement ordonnables : les graphes sans trou impair, sans antitrou et sans taureau.

Les graphes de Meyniel sont d'autres exemples de graphes parfaitement ordre-contractiles. Lorsque Hertz [45] prouve que les graphes de Meyniel sont parfaitement contractiles, il propose un algorithme qui permet effectivement de les contracter. Cet algorithme appelé Cosine dans [46] peut être reformulé pour produire un ordre de contraction.

ALGORITHME OrdreCosine
ENTRÉE : Un graphe G.
SORTIE : Un ordre de contraction $\{x_1^1, \ldots, x_1^{k_1}, \ldots, x_\ell^1, \ldots, x_\ell^{k_\ell}\}$ et la coloration associée.
CALCUL :
- Soient $c = 1$ et $i = 1$;
- tant qu'il existe des sommets non coloriés :
  - s'il existe un sommet non colorié qui n'a pas de voisin colorié c :

- soit A l'ensemble des sommets non coloriés qui ont un voisin colorié ;
- choisir un sommet non colorié v qui n'a pas de voisin colorié c et qui a le nombre maximum de voisins dans A ;
- colorier v avec la couleur c, poser $x_c^i = v$ et $i = i + 1$;

– sinon, poser $c = c + 1$ et $i = 1$.

COMPLEXITÉ : $\mathcal{O}(mn)$.

Dans [45], Hertz démontre en fait le résultat suivant.

Théorème 2.26 ([45]) *Soit G un graphe de Meyniel. L'algorithme* ORDRE-COSINE *appliqué à G produit un ordre de contraction amical de G.*

Nous en déduisons le corollaire suivant.

Corollaire 2.27 *Les graphes de Meyniel sont parfaitement ordre-contractiles.*

2.4 Clique maximum

Etant donnés un graphe et une coloration des sommets de ce graphe, voici un algorithme qui choisit un sommet de chaque couleur dans l'espoir de trouver une clique de même taille que la coloration.

ALGORITHME CLIQUE

ENTRÉE : Un graphe G et une coloration de ses sommets utilisant ℓ couleurs.

SORTIE : Un ensemble Q composé de ℓ sommets de G.

CALCUL :

- Soit $Q = \emptyset$;
- pour tout sommet x, soit $q(x) = 0$;
- pour $c = \ell, \ldots, 1$:
  - choisir un sommet x de couleur c qui maximise $q(x)$;
  - poser $Q = Q \cup \{x\}$;
  - pour tout voisin y de x, poser $q(y) = q(y) + 1$.

COMPLEXITÉ : $\mathcal{O}(m + n)$.

L'algorithme CLIQUE peut facilement être implémenté en temps $\mathcal{O}(m + n)$. Pour cela, à l'étape c, les sommets de couleur c sont parcourus, en choisissant un sommet qui maximise la valeur du compteur q, puis le compteur de ses voisins est mis à jour.

Le lemme suivant nous permet de montrer que, lorsque la coloration en entrée de l'algorithme est une coloration P_4-libre, alors la sortie Q de l'algorithme est une clique de taille ℓ.

Lemme 2.28 *Soient G un graphe P_4-libre-ordre-contractile et une coloration P_4-libre de G. Soit Q une clique dont les sommets sont de couleurs strictement supérieures à c, avec $1 \leq c \leq \ell - 1$. Alors il existe un sommet de couleur c qui voit tout Q.*

Preuve. Soit $\{x_1^1, \ldots, x_1^{k_1}, \ldots, x_\ell^1, \ldots, x_\ell^{k_\ell}\}$ un ordre de contraction P_4-libre tel que la coloration P_4-libre de G est associée à cet ordre. Soit G_c^i, w_c^i ($1 \leq c \leq \ell$, $1 \leq i \leq k_c$) la suite de contractions associée à cet ordre. Pour $1 \leq i \leq k_c$, considérons la propriété P_i suivante : "Dans le graphe G_c^i, le sommet w_c^i est adjacent à tout Q." Notons que la propriété P_{k_c} est vraie d'après (ii). Nous pouvons supposer que la propriété P_1 n'est pas vraie, sinon le lemme est vérifié avec le sommet $x_c^1 = w_c^1$. Donc, il existe un entier $2 \leq i \leq k_c$ tel que P_i est vraie et P_{i-1} ne l'est pas. Alors, dans le graphe G_c^{i-1}, le sommet x_c^i doit être adjacent à tout Q, sinon, Q contient deux sommets a, b tel que a est adjacent à w_c^{i-1} et pas à x_c^i et b est adjacent à x_c^i et pas à w_c^{i-1}, alors le chemin w_c^{i-1}-a-b-x_c^i est un chemin sans corde de longueur 3 qui contredit (i). Donc le lemme est vérifié avec le sommet x_c^i. □

Lemme 2.29 *Soient G un graphe P_4-libre-ordre-contractile et une coloration P_4-libre de G. Alors l'algorithme* Clique *appliqué à cette entrée produit une clique de taille $\omega(G)$.*

Preuve. Considérons l'ensemble Q construit au cours de l'algorithme Clique. Nous affirmons que, pour tout $c = \ell, \ell - 1, \ldots, 1$, à la fin de l'étape c, l'ensemble Q est une clique de taille $\ell - c + 1$ qui contient un sommet de chaque couleur $c, \ldots, \ell$. Le cas $c = \ell$ est évident. Lors de l'étape générale, le lemme 2.28 assure qu'il existe un sommet de couleur $c - 1$ qui est adjacent à tout Q. L'algorithme choisit un tel sommet et l'ajoute à Q. L'affirmation est donc vraie à la fin de chaque étape. L'algorithme termine avec une clique Q de taille ℓ. Puisque G admet une coloration de taille ℓ, nous avons $\chi(G) = \omega(G) = \ell$. □

En fait, l'hypothèse du lemme 2.29 donne des propriétés légèrement plus fortes. Une coloration est *fortement canonique* [48] si, pour toute clique Q dont les sommets sont de couleurs strictement supérieures à c, avec $1 \leq c \leq \ell - 1$, il existe une clique Q' contenant Q ainsi qu'un sommet de chaque couleur $1 \leq i \leq c$. Un *stable fort* [49] est un stable qui intersecte toutes les cliques maximales.

Lemme 2.30 *Les propriétés suivantes sont vérifiées :*
(i) une coloration P_4-libre est fortement canonique
(ii) l'ensemble des sommets de couleur 1 d'une coloration P_4-libre est un stable fort.

Preuve. Conséquence du lemme 2.28 qui peut être déduite exactement comme pour le lemme 2.29. □

Une clique est dite *maximale* si elle n'est pas strictement incluse dans une autre clique. Un graphe G est *fortement parfait* si et seulement si tout sous-graphe H de G contient un stable qui rencontre toutes les cliques maximales de H. En comparaison, un graphe G est parfait si et seulement si tout sous-graphe H de G contient un stable qui rencontre toutes les cliques de taille maximum de H. Nous en déduisons le corollaire suivant :

Corollaire 2.31 *Les graphes parfaitement P_4-libre-ordre-contractiles sont fortement parfaits.*

2.5 Optimisation pondérée

Soit G un graphe. Une *fonction de poids* sur ses sommets est une fonction w allant de $V(G)$ dans $\mathbb{N}$. Une *coloration pondérée* est un ensemble de stables $S_1, \ldots, S_k$ et d'entiers $I(S_1), \ldots, I(S_k)$ tels que pour tout sommet v, nous avons $w(v) \leq \sum_{v \in S_i} I(S_i)$. Une coloration pondérée est *minimum* si elle minimise la quantité $\sum_{1 \leq i \leq k} I(S_i)$. Cette quantité est alors notée $\chi_w(G)$. Une clique Q est de *taille pondérée maximum* si elle maximise la quantité $\sum_{v \in Q} w(v)$. Cette quantité est alors notée $\omega_w(G)$. Lorsque w associe à tout sommet l'entier 1, nous retrouvons les versions non-pondérées de ces problèmes. Pour un graphe parfait, il est bien connu que $\chi_w(G) = \omega_w(G)$ (voir [30, 35]).

Hoàng [50] a montré le théorème suivant.

Théorème 2.32 ([50]) *S'il existe un algorithme polynomial A qui trouve, pour tout graphe fortement parfait, un stable fort, alors il existe un algorithme polynomial B qui trouve, pour tout graphe fortement parfait et toute fonction de poids, une coloration pondérée minimum et une clique pondérée maximum. Si l'algorithme A fonctionne en temps T, alors l'algorithme B fonctionne en temps $\mathcal{O}(nT)$.*

L'algorithme suivant permet de trouver cette coloration et cette clique :

ALGORITHME PONDÉRÉ

ENTRÉE : Un graphe G, une fonction de poids w sur ses sommets et un algorithme A de complexité T qui trouve, pour tout sous-graphe de G, un stable fort.

SORTIE : Une coloration pondérée minimum $\mathcal{S}, I$ et une clique pondérée maximum Q.

ALGORITHME :

Etape 1 : construction de la coloration

- Soient $i = 1$, $S = \emptyset$, $Q = \emptyset$, $V = V(G)$;
- tant que $V \neq \emptyset$:
  - trouver un stable fort S_i ;
  - poser $\mathcal{S} = \mathcal{S} \cup \{S_i\}$ et $I(S_i) = min_{v \in S_i} w(v)$;
  - pour tout $v \in S_i$, poser $w(v) = w(v) - I(S_i)$ et si $w(v) = 0$, poser $V = V \backslash \{v\}$;
  - $i = i + 1$.

Etape 2 : construction de la clique

- Pour j allant de i à 1 en ordre décroissant :
  - si $Q \cup S_i = \emptyset$, soit $v \in S_i$ qui voit tout Q ;
  - poser $Q = Q \cup \{v\}$.

COMPLEXITÉ : $\mathcal{O}(nT)$.

Le lemme 2.30 montre que l'ensemble des sommets de couleur 1 d'une coloration P_4-libre est un stable fort. Nous pouvons donc en déduire le résultat suivant :

Corollaire 2.33 *S'il existe un algorithme polynomial A qui trouve, pour tout sous-graphe d'une classe de graphe $\mathcal{C}$, une coloration P_4-libre, alors il existe un algorithme polynomial B qui trouve, pour tout graphe dans $\mathcal{C}$ et toute fonction de poids, une coloration pondérée minimum et une clique pondérée maximum. Si l'algorithme A fonctionne en temps T, alors l'algorithme B fonctionne en temps $\mathcal{O}(nT)$.*

2.6 Inclusions

Il est facile de voir que les trous impairs, les antitrous et les prismes impairs ne sont pas des graphes fortement parfaits. Nous en déduisons le lemme suivant.

Lemme 2.34 *Un graphe fortement parfait est un graphe de Grenoble.*

La conjecture 2.14 implique que les graphes fortement parfaits sont parfaitement contractiles mais ceci reste une conjecture.

Conjecture 2.35 ([28]) *Les graphes fortement parfaits sont parfaitement contractiles.*

En fait, nous ne connaissons pas de graphe fortement parfait qui ne soit pas fortement parfaitement contractile. Nous posons donc la question suivante.

Question 2.36 *Est-ce que les graphes fortement parfaits sont parfaitement ordre-contractiles ?*

Ainsi qu'une version moins forte.

Question 2.37 *Est-ce que les graphes fortement parfait sont parfaitement P_4-libre-ordre-contractile ?*

Puisque les graphes parfaitement P_4-libre-ordre-contractiles sont fortement parfaits, la conjecture 2.35 implique une réponse par l'affirmative à la question suivante.

Question 2.38 *Est-ce que les graphes parfaitement P_4-libre-ordre-contractiles sont parfaitement contractiles ?*

Le graphe orienté de la figure 2.4 représente les relations d'inclusion entre certaines des classes de graphes considérées dans cette thèse. Une flèche de la classe A vers la classe B signifie que A contient B. Toutes les classes dont le nom contient "parfait" comme sous-mot sont définies pour avoir certaines propriétés intéressantes vis-à-vis du problème de la coloration. Toutes les autres classes sont définies par sous-graphes exclus. Parfois elles portent un nom mentionné entre parenthèses. Les lettres T, A, P, M signifient respectivement Trou, Antitrou, Prisme, Maison, l'ajout d'un "i" veut dire impair.

Les affirmations suivantes sont suffisantes pour expliquer les relations d'inclusion entre les classes définies par sous-graphes exclus. Si un graphe ne contient pas de "bidule", il ne contient pas non plus de "bidule" impair. Les antitrous de taille ≥ 6 contiennent une maisonnette. Les prismes différents de $\overline{C_6}$ contiennent un trou et une maison.

Une seule classe est définie à la fois par sous-graphes exclus et par ses propriétés intéressantes, il s'agit de la classe des graphes parfaits. Comme remarqué précédemment, il est facile de voir que "Parfait" $\subseteq$ "{Ti,Ai} - libre". Le théorème fort des graphes parfaits de Chudnovsky, Robertson, Seymour et Thomas [14] montre la réciproque symbolisée par le cadre contenant ces deux classes.

Le corollaire 2.16 implique l'arc "{Ti,A} - libre" → "parfaitement P_4-libre-contractile".

Le corollaire 2.13 [28] implique l'arc "{Ti,A,Pi} - libre" → "parfaitement contractile".

Le lemme 2.34 implique l'arc "{Ti,A,Pi} - libre" → "fortement parfait".

Le théorème 2.19 [66] implique l'arc "parfaitement contractile" → "{Ti,A,P} - libre".

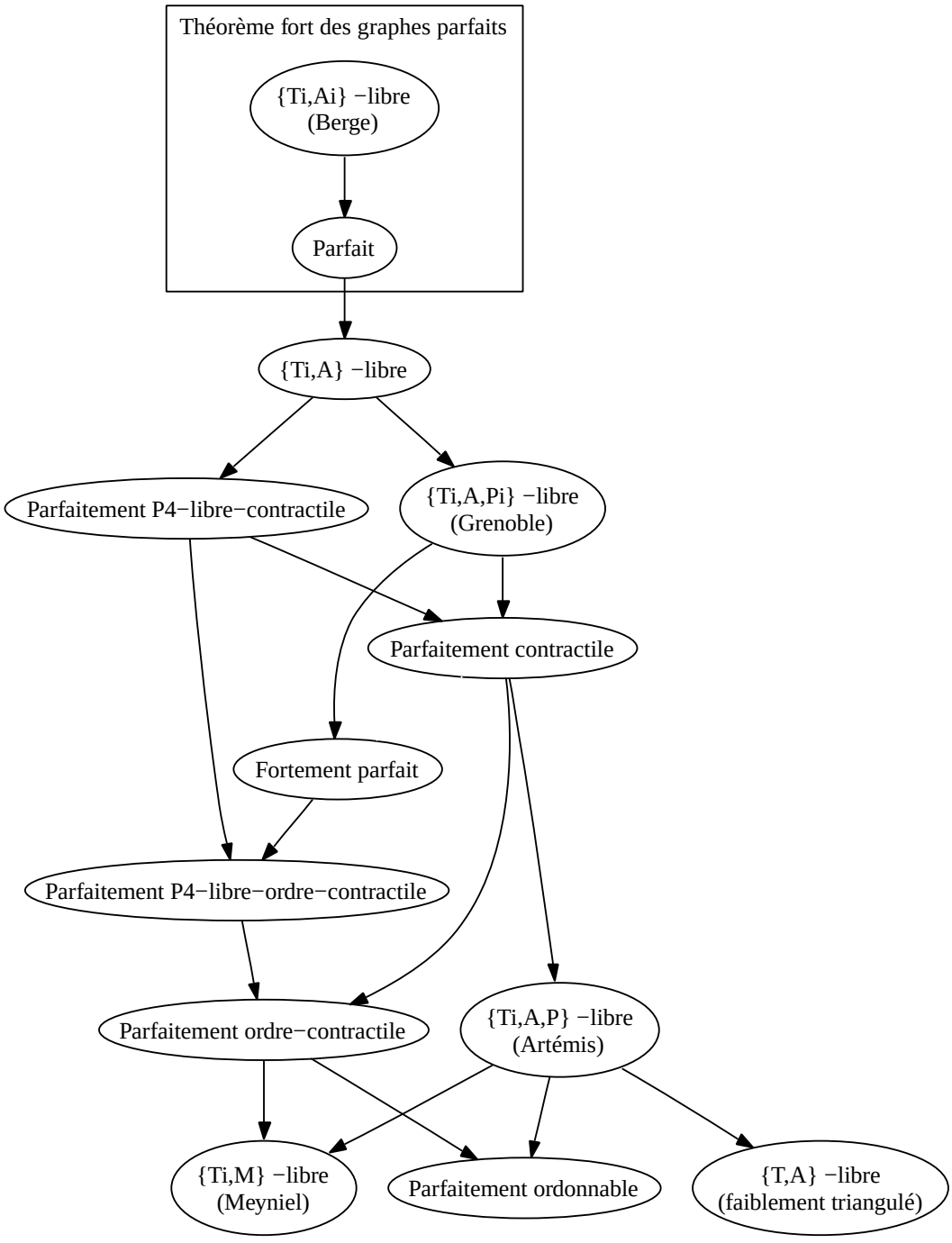
Le corollaire 2.31 implique l'arc "fortement parfait" → "parfaitement P_4-libre-ordre-contractile".

Le corollaire 2.27 [45] implique l'arc "parfaitement ordre-contractile" → "{Ti,M} - libre".

Le corollaire 2.25 [48] implique l'arc "parfaitement ordre-contractile" → "parfaitement ordonnable".

Pour l'arc "{Ti,A,P} - libre" → "parfaitement ordonnable", il suffit de se persuader que les trou impairs, les antitrous et les prismes ne sont pas parfaitement ordonnables [66].

Les autres arcs sont des conséquences directes des définitions.



T = Trou A = Antitrou P = Prisme M = Maison i = impair

FIG. 2.4 – Relations d'inclusion entre certaines classes de graphes

Chapitre 3

Graphes sans taureau

Dans ce chapitre, nous considérons la classe des graphes qui ne contiennent ni trou impair, ni antitrou, ni taureau. Nous présentons un nouvel algorithme qui trouve, pour tout graphe de cette classe, une coloration optimale. L'algorithme est basé sur l'existence dans ces graphes, d'un ordre d'élimination de ses sommets ayant une certaine propriété de simplicialité. Plus généralement, nous démontrons, en utilisant une variante de l'algorithme LexBFS, que dans tous les graphes qui ne contiennent ni trou, ni taureau, il existe un sommet qui n'est pas le milieu d'un chemin sans corde de longueur cinq. Ce dernier fait généralise des résultats connus concernant les graphes bipartis cordaux, les matrices totalement équilibrées et les graphes fortement triangulés.

3.1 Méthode

Chvátal et Sbihi [19] ont prouvé que la Conjecture Forte des Graphes Parfait était vraie pour les graphes contenant ni trou impair, ni antitrou impair, ni taureau. La structure de ces graphes a aussi été étudiée par Reed et Sbihi [77], De Figueiredo, Maffray et Porto [22, 23], Hayward [40]. De Figueiredo et Maffray [21] ont proposé un algorithme combinatoire, basé sur des résultats de [19, 22], qui trouve une coloration optimale pour tout graphe ne contenant ni trou impair, ni antitrou impair, ni taureau. La complexité de cet algorithme est $\mathcal{O}(n^5m^3)$.

Voici quelques définitions utilisées dans cette partie. Un graphe G est *transitivement orientable* [31] s'il est possible de donner une orientation à chacune de ses arêtes telle que, pour tout chemin orienté $u \to v \to w$, l'arc $u \to w$ est présent dans l'orientation. Un *ensemble homogène* dans un graphe G est un ensemble $S \subsetneq V(G)$, contenant au moins deux sommets, tel que

tout sommet de $V(G) \setminus S$ est adjacent, soit à tout, soit à aucun sommet de S.

Soit $\mathcal{B}$ la classe des graphes contenant ni trou impair, ni antitrou, ni taureau. Un prisme différent de $\overline{C_6}$ contient un taureau, donc "Artémis sans taureau", "Grenoble sans taureau" et "parfaitement contractile sans taureau" désignent tous les trois la classe $\mathcal{B}$. Il existe trois méthodes purement combinatoires permettant de colorier les graphes dans la classe $\mathcal{B}$:

Méthode 1

Des résultats de [22, 23] montrent que pour tout graphe G dans $\mathcal{B}$, soit G est faiblement triangulé, soit G est transitivement orientable, soit G contient un ensemble homogène. Les ensembles homogènes peuvent être éliminés par la méthode de *décomposition modulaire*, qui décompose un graphe en $\mathcal{O}(n)$ sous-graphes sans ensemble homogène. La décomposition modulaire peut être effectuée en temps $\mathcal{O}(n+m)$ (voir par exemple [36]). D'après [22, 23], pour un graphe dans la classe $\mathcal{B}$, ses sous-graphes non décomposables sont soit faiblement triangulés, soit transitivement orientables. Il est donc possible de trouver une coloration optimale pour ces sous-graphes, en temps $\mathcal{O}(nm)$ pour les faiblement triangulés [42, 43] et en temps $\mathcal{O}(m)$ pour les transitivement orientables [64]. Il est alors possible de recombiner ces colorations optimales le long de la décomposition modulaire pour obtenir une coloration optimale du graphe d'origine. Cette méthode est de complexité $\mathcal{O}(n^2m)$.

Méthode 2

Chvátal [17] a conjecturé que tout graphe dans $\mathcal{B}$ est parfaitement ordonnable. Hayward [40] a prouvé cette conjecture, en utilisant des résultats de [22, 23]. Nous estimons que les techniques de [40] ont pour complexité $\mathcal{O}(n^5)$ (l'exposant 5 est dû à la recherche d'un P_5 induit effectué dans [23]). Donc, en combinant les techniques de [22, 23, 40] et en utilisant à nouveau une décomposition modulaire de complexité linéaire comme dans [36], il est possible de trouver un ordre parfait, pour tout graphe de la classe $\mathcal{B}$, en temps $\mathcal{O}(n^5(n+m))$. Alors, l'algorithme COLOR, appliqué sur cet ordre, produit une coloration optimale en temps $\mathcal{O}(m)$. Cette méthode est de complexité $\mathcal{O}(n^5(n+m))$.

Méthode 3

Puisque tout graphe de la classe $\mathcal{B}$ est un graphe d'Artémis, il est possible d'utiliser l'algorithme de coloration des graphes d'Artémis présenté au chapitre 5. Cet algorithme est de complexité $\mathcal{O}(n^2m)$.

Nous proposons une nouvelle méthode, plus simple et plus rapide que les précédentes. Nous définissons un algorithme qui permet de trouver un ordre de contraction amical dans un graphe de la classe $\mathcal{B}$. Les résultats du chapitre 2 seront ensuite utilisés pour trouver, de manière robuste, une coloration canonique, un stable fort, une clique de taille maximum, une coloration pondérée minimum, une clique pondérée maximum. D'autre part, cela donne une nouvelle démonstration que ces graphes sont parfaitement ordre-contractiles. Ce que nous savions déjà en combinant les résultats de Hayward [40] et de Herz, de Werra [48].

La classe des graphes dans $\mathcal{B}$ n'est pas très éloignée de la classe des graphes de Meyniel. Ces deux classes ne contiennent ni trou impair, ni antitrou, ni maison de taille ≥ 7. Pour la classe des graphes de Meyniel, il faut exclure aussi la maisonnette et pour la classe $\mathcal{B}$, il faut exclure le taureau. La maisonnette et le taureau ne diffèrent que d'une seule arête (voir figure 3.1).

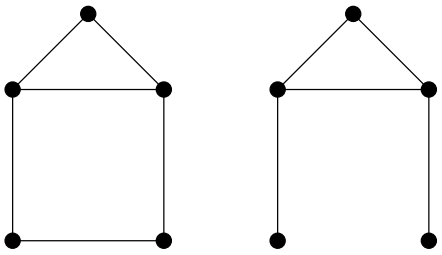


FIG. 3.1 – Similarité entre la maisonnette et le taureau

Les algorithmes COSINE de Hertz [45] et LEXCOLOR de Roussel et Rusu [81] permettent de colorier optimalement les graphes de Meyniel. Ce sont donc de bons candidats pour colorier les graphes dans $\mathcal{B}$. Mais nous avons vu au chapitre 1 que, sur le graphe $\overline{P_6}$, ces algorithmes peuvent ne pas donner une coloration optimale. Le graphe $\overline{P_6}$ ne contient ni trou impair, ni antitrou, ni taureau.

Les algorithmes COSINE et LEXCOLOR commencent tous les deux par colorier n'importe quel sommet du graphe en entrée. Cela fonctionne pour les graphes de Meyniel, car tous les sommets du graphe sont dans une paire d'amis. Ceci n'est malheureusement pas le cas pour les graphes dans $\mathcal{B}$. Une maisonnette est un exemple de graphe dans $\mathcal{B}$ qui ne vérifie pas cette propriété : le toit d'une maisonnette n'est dans aucune paire d'amis. Il est cependant possible de montrer que tout graphe dans $\mathcal{B}$ contient un sommet qui n'est pas le toit d'une maisonnette. En fait, nous démontrons un résultat

plus général concernant la classe $\mathcal{C}$ des graphes contenant ni trou, ni taureau. Comme le taureau est un graphe auto-complémentaire (voir figure 1.2), nous avons clairement $\mathcal{B} \subsetneq \overline{\mathcal{C}}$.

Nous proposons d'appeler sommet *P_k-simplicial*, un sommet v tel que tout chemin sans corde sur k sommets dont v est un milieu est un chemin simplicial. Clairement, un sommet P_k-simplicial est P_{k+1}-simplicial. Nous retrouvons les notions classiques de simplicialité. Un sommet P_1-simplicial est un sommet qui n'est pas le milieu d'un P_3, appelé aussi sommet simplicial [78]. Un sommet P_2-simplicial est un sommet qui n'est interne d'aucun P_4, appelé aussi sommet *semi-simplicial* [53]. La notion qui nous intéresse ici est celle de P_3-simplicialité. Un sommet P_3-simplicial est un sommet qui n'est pas le milieu d'un P_5. Dans le graphe complémentaire, ce sommet n'est pas le toit d'une maisonnette (voir figure 3.2).

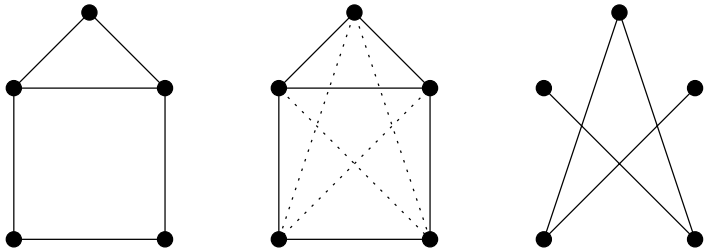


FIG. 3.2 – La maisonnette et son complémentaire.

Un *ordre d'élimination P_k-simplicial* est un ordre $(v_1, \ldots, v_n)$ des sommets de G tel que, pour $1 \leq i \leq n$, le sommet v_i est P_k-simplicial dans le graphe $G[v_1, \ldots, v_i]$. Rose, Tarjan et Lueuker [78] ont montré que, sur un graphe triangulé, l'algorithme LEXBFS produit un ordre d'élimination P_1-simplicial. Jamison et Olariu [53] ont montré que, sur un graphe ne contenant ni trou, ni maisonnette, ni domino, l'algorithme LEXBFS produit un ordre d'élimination P_2-simplicial. Est-ce que l'algorithme LEXBFS produit un ordre d'élimination P_3-simplicial sur un graphe dans $\mathcal{C}$? Malheureusement non, le graphe de la figure 3.3 est un exemple de graphe contenant ni trou, ni taureau et sur lequel l'ordre obtenu par l'algorithme LEXBFS n'est pas P_3-simplicial. Le dernier sommet numéroté par l'algorithme est le milieu d'un P_5.

Il est cependant possible de départager les cas d'égalité de l'algorithme LEXBFS pour obtenir la propriété souhaitée. Nous avons ainsi défini un nouvel algorithme de complexité $\mathcal{O}(nm)$ appelé LEXBFS* qui trouve un ordre d'élimination P_3-simplicial pour tout graphe dans la classe $\mathcal{C}$.

Une conséquence de cet algorithme est la suivante. Un graphe est *biparti cordal* s'il est biparti et s'il ne contient pas de trou de longueur au moins

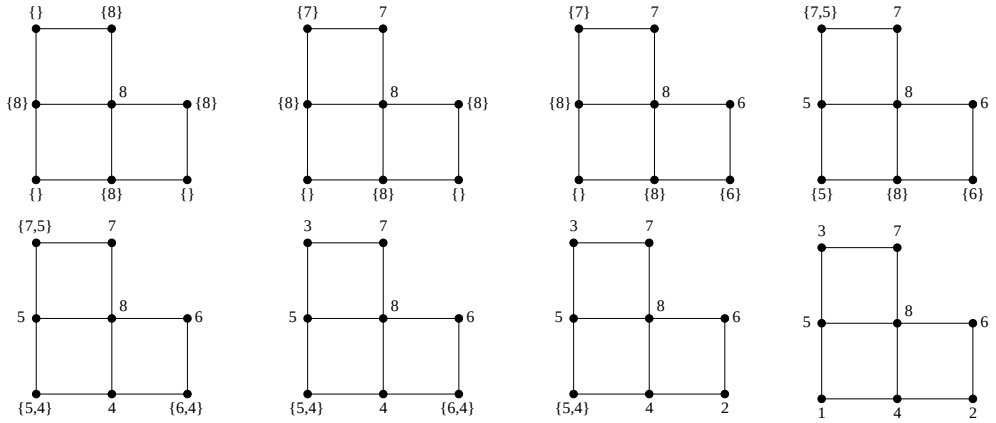


FIG. 3.3 – L'algorithme LEXBFS termine sur le milieu d'un P_5.

six. Un résultat classique est l'existence dans tout graphe biparti cordal d'un sommet qui n'est pas le milieu d'un P_5. Ce résultat est connu sous différentes variantes, telles que l'existence d'un sommet *simple* dans un graphe *fortement cordal*, ou l'existence d'un ordre *sans*-Γ dans une matrice *totalement balancée* [63]. Puisque tout graphe biparti cordal est dans la classe $\mathcal{C}$, nous généralisons ce résultat.

Un sommet est dit *$\overline{P_k}$-simplicial*, s'il est P_k-simplicial dans le graphe complémentaire. Un *ordre d'élimination $\overline{P_k}$-simplicial* est un ordre $(v_1, \ldots, v_n)$ des sommets de G tel que, pour $1 \leq i \leq n$, le sommet v_i est $\overline{P_k}$-simplicial dans le graphe $G[v_1, \ldots, v_i]$. L'algorithme LEXBFS*, appliqué au graphe complémentaire d'un graphe de la classe $\mathcal{B}$, permet d'obtenir un ordre d'élimination $\overline{P_3}$-simplicial. Ensuite, nous transformons cet ordre en un ordre de contraction amical grâce à un algorithme de complexité $\mathcal{O}(nm)$ appelé ORDRECOSINE*. Ce nouvel algorithme combine les idées des algorithmes ORDRECOSINE et ORDRECONTRACTION présentés dans le chapitre 2.

3.2 L'algorithme LEXBFS*

L'algorithme LEXBFS* est un cas particulier de l'algorithme LEXBFS. Lors de l'étape générale de LEXBFS le sommet numéroté est un sommet d'étiquette lexicographiquement maximale, les cas d'égalité étant départagés arbitrairement.

Dans LEXBFS*, nous avons besoin de départager les cas d'égalité en ajoutant la règle suivante. Supposons qu'à une étape donnée de l'algorithme, l'ensemble A des sommets non numérotés ayant une étiquette maximale satisfait $|A| \geq 2$. Soit $L(A)$ l'étiquette commune des sommets de A. Soit U

l'ensemble des sommets non numérotés qui ne sont pas dans A. Pour chaque $u \in U$, soit $L'(u) = L(u) \setminus L(A)$. Les sommets de U sont triés lexicographiquement selon L'. Alors, le premier sommet de U (maximal selon l'ordre sur L') élimine de A tous ses non-voisins (sauf si cela amène A à devenir vide ; dans ce cas il est sans effet) ; puis le deuxième sommet de U élimine ses non-voisins, ... Cette procédure s'arrête lorsque tous les sommets de U ont été considérés (les cas d'égalités sont alors départagés arbitrairement). Voici une description formelle de l'algorithme :

ALGORITHME LEXBFS*
ENTRÉE : Un graphe G.
SORTIE : Un ordre σ sur les sommets de G.
CALCUL :
1. Pour tout sommet a, soit $L(a) = \emptyset$;
2. pour $i = n, \ldots, 1$:
   2.1. soit A l'ensemble des sommets non numérotés ayant une étiquette maximum, et soit U les autres sommets non numérotés ;
   2.2. tant que $|U| > 0$:
      2.2.1 choisir un sommet $u \in U$ pour lequel $L(u) \setminus L(A)$ est maximum ;
      2.2.2 poser $U = U \setminus \{u\}$. Si $A \cap N(u) \neq \emptyset$, alors poser $A = A \cap N(u)$;
   2.3. choisir un sommet $a \in A$ et poser $\sigma(a) = i$;
   2.4. pour chaque voisin non numéroté v de a, ajouter i à $L(v)$.
COMPLEXITÉ : $\mathcal{O}(n \times min(m, \overline{m}))$.

Dans tout ce chapitre, lorsque nous ferons référence aux différentes étapes des algorithmes LEXBFS et LEXBFS*, nous ferons référence à la numérotation des lignes précédentes. L'algorithme LEXBFS est exactement l'algorithme LEXBFS* sans la boucle 2.2.

Analyse de complexité de LEXBFS

Rose, Tarjan et Lueker [78] ont montré que l'algorithme LEXBFS pouvait être implémenté en temps $\mathcal{O}(n + m)$ de la manière suivante. Les sommets peuvent être triés selon la valeur de $L(v)$ en utilisant des techniques usuelles [1]. Pour chaque étiquette ℓ, nous maintenons un ensemble S_ℓ contenant les sommets non numérotés v tel que $L(v) = \ell$. Cet ensemble est implémenté par une liste doublement chaînée, où chaque élément pointe aussi vers la tête de la liste, qui est un cellule spéciale contenant leur étiquette. Les têtes des S_ℓ non-vides sont elles-même placées en ordre lexicographique dé-

croissant dans une liste doublement chaînée M. Pendant l'initialisation, tous les sommets sont mis dans $S_\emptyset$ et $S_\emptyset$ est le seul élément de M. L'initialisation prend donc un temps $\mathcal{O}(n)$. L'ensemble A de l'étape 2.1 de l'algorithme est le premier ensemble de M. Lorsque un sommet a de A est choisi lors de l'étape 2.3, il est retiré de la structure de données, et chaque voisin u de a est retiré de l'ensemble S_ℓ qui contient u et est ajouté dans l'ensemble $S_{\ell \cup \{\sigma(a)\}} = S_\ell \cap N(A)$, placé juste avant S_ℓ dans M (les ensembles vides sont retirés de M). Cette opération de découpage des S_ℓ prend un temps $\mathcal{O}(d(a))$. Le coût total des étapes 2.3 et 2.4 est donc $\mathcal{O}(n+m)$. C'est ainsi que LEXBFS est implémenté dans [78]. La figure 3.4 représente l'état de la structure de données lors de la première étape de l'exécution de l'algorithme sur le graphe de la figure 1.7.

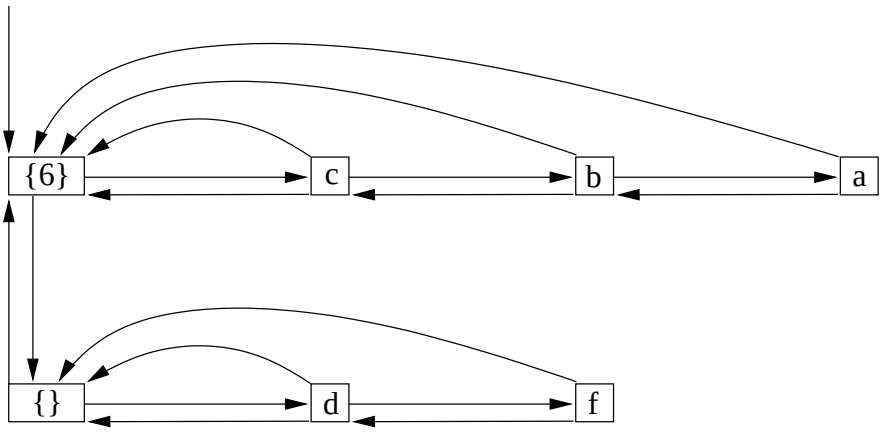


FIG. 3.4 – Structure de données utilisée dans l'algorithme LEXBFS

Analyse de complexité de LEXBFS*

Le fait de départager les cas d'égalité dans LEXBFS* augmente la complexité. L'algorithme LEXBFS* peut être implémenté en temps $\mathcal{O}(nm)$ de la manière suivante. L'ensemble U défini lors de l'étape 2.1 est trié avant l'étape 2.2 selon l'ordre $L'(u)$ en utilisant le même type de structure de données que précédemment. Cela nécessite un temps $\mathcal{O}(n+m)$. Lors de l'étape 2.2.1, nous sélectionnons un sommet maximum u en temps constant, l'étape 2.2.2 est réalisée en temps $\mathcal{O}(d(u))$. Le coût total de la boucle 2.2 est donc $\mathcal{O}(n+m)$ à chaque passage dans la boucle. En conclusion, puisqu'il y a n itérations de la boucle principale, le temps d'exécution total de l'algorithme LEXBFS* est $\mathcal{O}(nm)$.

En fait, nous allons avoir besoin d'appliquer l'algorithme LEXBFS* sur le complément $\overline{G}$ d'un graphe G. Soit $\overline{m}$ le nombre d'arêtes de $\overline{G}$. Puisque

$\overline{m} = \mathcal{O}(n^2)$, la complexité est à priori en $\mathcal{O}(n^3)$. Il est possible de réduire cette complexité de la manière suivante. Lors de l'application de l'algorithme sur $\overline{G}$, le découpage des ensembles S_ℓ nécessite un temps $\mathcal{O}(\overline{d}(a))$, où $\overline{d}$ est la fonction des degrés de $\overline{G}$. Nous pouvons le réaliser en temps $\mathcal{O}(d(a))$ si, au lieu de retirer chaque voisin u de a (dans $\overline{G}$) de l'ensemble S_ℓ qui contient u et de l'ajouter dans le nouvel ensemble $S_\ell \cap N_{\overline{G}}(A)$, nous retirons chaque voisin u de a (dans G) de l'ensemble S_ℓ qui contient u et que nous l'ajoutons à l'ensemble $S_\ell \setminus N_G(A)$, placé juste après S_ℓ dans M. La même idée peut être utilisée pour trier l'ensemble U et pour mettre à jour A en temps $\mathcal{O}(n+m)$. En conclusion, le temps d'exécution total de l'algorithme LexBFS* appliqué sur le complémentaire $\overline{G}$ d'un graphe G est encore $\mathcal{O}(nm)$.

Propriétés de l'algorithme LexBFS

Voici quelques notations et propriétés de l'algorithme LexBFS. Lorsque l'algorithme sélectionne un sommet $a \in A$ lors de l'étape 2.3, la valeur courante de l'étiquette d'un sommet u juste avant que a soit numéroté est notée $L_a(u)$. Pour simplifier les notations, $\sigma(a) < \sigma(b)$ est noté $a < b$ (le sommet a a été numéroté après b).

Lemme 3.1 *Soient a, b, u tels que $a < u$, $b \leq u$ et $L_u(a) < L_u(b)$. Alors $a < b$, et, pour tout v tel que $v \leq u$, $L_v(a) < L_v(b)$.*

Preuve. Supposons $a < u$, $b \leq u$ et $L_u(a) < L_u(b)$. Lors de l'étape de l'algorithme où u est numéroté, il existe $i > \sigma(u)$, $i \in L_u(b) \setminus L_u(a)$, tel que $\forall j > i$, $j \in L_u(a) \cap L_u(b)$ ou $j \notin L_u(a) \cup L_u(b)$. Après que u soit numéroté, les entiers qui peuvent être ajoutés à $L(a)$ et $L(b)$ sont plus petits que $\sigma(u)$ et donc strictement plus petits que i, donc l'inégalité $L(a) < L(b)$ est vérifiée dans toute la suite de l'algorithme. □

Lemme 3.2 *Soient a, b tels que $a < b$ et $L_b(a) \neq L_b(b)$. Alors il existe un sommet $> b$ qui voit b et manque a. Soit $f(b,a)$ un tel sommet maximum. Alors tout sommet u tel que $f(b,a) < u$ vérifie : u voit a, b ou u manque a, b. Et donc tout sommet u tel que u voit a et u manque b vérifie : $u < f(b,a)$.*

Preuve. Le sommet b est choisi avant a et $L_b(a) \neq L_b(b)$ donc $L_b(a) < L_b(b)$. Donc il existe $i \in L_b(b) \setminus L_b(a)$, tel que $\forall j > i$, $j \in L_b(a) \cap L_b(b)$ ou $j \notin L_b(a) \cup L_b(b)$. Soit $f(b,a)$ le sommet tel que $\sigma(f(b,a)) = i$.

Soit u un sommet tel que $f(b,a) < u$. Soit $j = \sigma(u)$. Puisque $i = \sigma(f(b,a)) < \sigma(u) = j$, nous avons $j \in L_b(a) \cap L_b(b)$ ou $j \notin L_b(a) \cup L_b(b)$, et donc u voit a, b, ou u manque a, b. □

Lemme 3.3 *Soient a, b, u tels que $a < b < u$, u voit a et u manque b. Soient $a_0 = a$, $b_0 = b$, $a_1 = u$, $b_1 = f(b, a)$ et pour $i \geq 2$, a_i et b_i définis de la manière suivante, tant que c'est possible :*
- *si b_i manque a_i, soit $a_{i+1} = f(a_i, b_{i-1})$.*
- *si a_{i+1} manque b_i, soit $b_{i+1} = f(b_i, a_i)$.*

Soit k l'entier maximum tel que a_k est défini. Soit ℓ l'entier maximum tel que b_ℓ est défini, alors $\ell \in \{k, k+1\}$. Soit $\mathcal{P}(u, b, a)$ le chemin a_0-...-a_k-b_ℓ-...-b_0. Si a manque b, alors $\mathcal{P}(u, b, a)$ induit un chemin sans corde. Si a voit b, alors $\mathcal{P}(u, b, a)$ induit un trou ou un carré.

Preuve. Supposons $\ell = k$ (le cas $\ell = k + 1$ est similaire). Nous prouvons par induction sur $j \leq k$ les propriétés suivantes :
- les suites $(a_i)_{i \leq j}$ et $(b_i)_{i \leq j}$ sont bien définies ;
- $a_0 < b_0 < a_1 < b_1 < \cdots < a_j < b_j$;
- a_0-...-a_j et b_0-...-b_j sont des chemins sans corde ;
- il n'y a pas d'arête entre un sommet a_i et un sommet b_i, excepté $a_k b_k$ et éventuellement $a_0 b_0$.

Si $j = 1$, alors a_1 voit a_0, manque b_0, et $a_0 < b_0 < a_1$, donc $L_{b_0}(a_0) \neq L_{b_0}(b_0)$. Donc le sommet $b_1 = f(b_0, a_0)$ est bien défini par le lemme 3.2. Le sommet b_1 voit b_0, manque a_0 et $a_1 < b_1$. Donc la propriété est vraie pour $j = 1$.

Supposons que $1 \leq j < k$ et que la propriété est vraie pour j. Puisque b_j voit b_{j-1}, manque a_j, et $b_{j-1} < a_j < b_j$, nous avons $L_{a_j}(b_{j-1}) \neq L_{a_j}(a_j)$. Appliquons le lemme 3.2 pour définir $a_{j+1} = f(a_j, b_{j-1})$. Le sommet a_{j+1} voit a_j, manque b_{j-1} et $b_j < a_{j+1}$. Puisque a_{j+1} manque b_{j-1}, et $a_0 < b_0 < a_1 < b_1 = f(b_0, a_0) < \cdots < a_j = f(a_{j-1}, b_{j-2}) < b_j = f(b_{j-1}, a_{j-1})$, alors a_{j+1} manque $a_0, \ldots, a_{j-1}, b_0, \ldots, b_{j-1}$. Il en va de même pour b_{j+1}. Donc la propriété est vraie pour $j + 1$. □

Lemme 3.4 *Si G est sans trou, et soit a, b, u tels que $a < b < u$, u voit a, u manque b et a voit b. Alors $f(b, a)$ voit u.*

Preuve. Considérons le chemin $\mathcal{P}(u, b, a)$ du lemme 3.3. Puisque a voit b, le chemin induit un trou ou un carré, et G est sans trou donc $f(b, a)$ voit u. □

Propriétés de LexBFS*

Voici quelques notations et propriétés de l'algorithme LexBFS*. Lorsque l'algorithme choisit un sommet $a \in A$ lors de l'étape 2.3 de l'algorithme LexBFS*, pour tout sommet (non numéroté) u, soit $L'_a(u) = L_a(u) \setminus L_a(a)$.

Lemme 3.5 *Soient a, b tels que $a < b$, $L_b(a) = L_b(b)$ et $N(a) \neq N(b)$. Alors, pendant la boucle de l'étape 2.2 de l'algorithme* LexBFS*, *le sommet a a été retiré de A par un sommet $u = g(b, a)$ vérifiant : u voit b, u manque a, $u < a$ et $L_b(u) < L_b(b)$.*

S'il existe un sommet v tel que $v < a$, v voit a, v manque b et $L_b(v) \neq L_b(b)$, alors $L'_b(v) \leq L'_b(u)$. Si $L'_b(v) \neq L'_b(u)$, alors il existe un sommet $> b$ qui voit u et manque a, b, v. Soit $x = h(u, v)$ un tel sommet maximum. Alors, tout sommet y, tel que $x < y$ et y manque a, b, vérifie : y voit u, v ou y manque u, v. Et donc tout sommet y tel que y voit v et y manque a, b, u, vérifie : $y < x$.

Preuve. La définition de u et ses propriétés découlent de la définition de l'algorithme. Supposons qu'il existe un sommet v tel que $v < a$, v voit a, v manque b et $L_b(v) \neq L_b(b)$.

Supposons $L'_b(v) > L'_b(u)$, alors v devrait avoir été sélectionné lors de l'étape 2.2.1 avant u. Alors, à l'étape 2.2.2, $A \cap N(v)$ devrait être vide, sinon b est retiré de A et b n'est pas le sommet choisi lors de l'étape 2.3. Puisque a est dans $N(v)$, il a été retiré de A précédemment, par un sommet w vérifiant $L'_b(w) \geq L'_b(v) > L'_b(u)$, et donc $w \neq u$. Cela contredit la définition de $u = g(b, a)$, donc $L'_b(v) \leq L'_b(u)$.

Si $L'_b(v) \neq L'_b(u)$, alors $x = h(u, v)$ est bien défini. Soit y un sommet tel que $x < y$ et y manque a, b. Si y voit v et y manque u, alors $L'_b(v) < L'_b(u)$ implique qu'il existe un sommet $> y$ qui voit u et manque a, b, v, une contradiction à la définition de x. Si y voit u et manque v, cela contredit aussi la définition de x. Donc y voit u, v ou y manque u, v. □

3.3 P_3-simplicialité

Rappelons que $\mathcal{C}$ désigne la classe des graphes ne contenant ni trou, ni taureau. Dans cette partie, nous montrons que lorsque le graphe en entrée est dans $\mathcal{C}$, l'ordre produit par l'algorithme LexBFS* est un ordre d'élimination P_3-simplicial. Ce résultat n'est pas vrai pour l'algorithme LexBFS comme nous l'avons montré sur l'exemple de la figure 3.3.

Avant de prouver le théorème principal, nous avons besoin du lemme suivant :

Lemme 3.6 *Dans un graphe $G \in \mathcal{C}$, soit $P = a_0$-a_1-$\cdots$-a_r un chemin sans corde avec $r \geq 4$. Soit u un sommet qui voit a_0 et a_r. Alors l'une des propriétés suivantes est vraie :*

(i) u voit tous les sommets de P ;

(ii) r *est pair,* u *voit* $a_0, a_2, \ldots, a_r$ *et manque* $a_1, a_3, \ldots, a_{r-1}$ *;*
(iii) $r = 4$*,* u *voit* a_2 *et exactement l'un de* a_1, a_3*.*
Dans chacun des cas, u *voit* a_2 *et* a_{r-2}*.*

Preuve. Appelons *segment* n'importe quel sous-chemin de P, de longueur au moins 1, dont les extrémités voient u et dont les sommets intérieurs manquent u. Le chemin P est partitionné (au sens des arêtes) en ses segments. Puisque G ne contient pas de trou, chaque segment est de longueur 1 ou 2. Pour $\ell = 1, 2$, soit s_ℓ le nombre de segments de P de longueur ℓ. Alors $r = s_1 + 2s_2$. Si $s_1 = 0$, chaque segment est de longueur 2 et (ii) est vraie. Maintenant, supposons $s_1 > 0$. Alors u voit deux sommets consécutifs de P. Supposons que (i) n'est pas vraie, alors u a un non voisin dans P. Par symétrie, il y a un entier i tel que u voit a_i et a_{i+1} et pas a_{i+2}. Alors $i \leq 1$ (sinon a_0-ua_ia_{i+1}-a_{i+2} est un taureau) et $r \leq i + 3$ (sinon a_r-ua_ia_{i+1}-a_{i+2} est un taureau). Donc $r = 4$ et $i = 1$ et (iii) est vraie. □

Maintenant nous prouvons le théorème suivant.

Théorème 3.7 *Soit* G *un graphe dans* $\mathcal{C}$*. L'algorithme* LexBFS* *appliqué à* G *produit un ordre d'élimination* P_3*-simplicial.*

Preuve du théorème. Un P_5 a-b-c-d-e dans G est *mauvais* si $c < min\{a, b, d, e\}$. Notre but est de prouver qu'il n'y a pas de mauvais P_5. Supposons, au contraire, qu'il en existe un et montrons que nous aboutissons à une contradiction. Un mauvais P_5 a-b-c-d-e est *plus mauvais* qu'un autre mauvais P_5 a'-b'-c'-d'-e' si $a \geq a', b \geq b', c \geq c', d \geq d', e \geq e'$ et qu'au moins une de ces inégalités est stricte. Soit a-b-c-d-e le plus mauvais P_5. Par symétrie, nous pouvons supposer que $e < a$.

Affirmation 3.8 $e < b$.

Preuve. Supposons que l'affirmation est fausse, donc $c < b < e < a$.

Puisque a voit b, manque e et $b < e < a$, nous pouvons considérer le chemin sans corde $R = \mathcal{P}(a, e, b)$ du lemme 3.3. Si aucun sommet de c, d n'a de voisin dans R^*, alors $R \cup \{c, d\}$ est un cycle de longueur au moins 6, donc l'un de c, d a un voisin dans R^*. Soit q le sommet de R^* le plus près de a qui voit l'un de c, d. Si q manque c, alors $R[b, q] \cup \{d, c\}$ est un trou. Donc q voit c. Le cycle sans corde $R[b, q] \cup \{c\}$ doit être de longueur < 5, donc q voit a et donc $q \neq d$.

Puisque q voit c, manque b et $c < b < q$, nous avons $L_b(c) \neq L_b(b)$. Appliquons le lemme 3.2 pour définir $r = f(b, c)$. Le sommet r voit b, manque c et $q < r$. Puisque b voit c, le sommet r voit q d'après le lemme 3.4. Puisque

r voit b, nous avons $r \neq d$. Puisque $f(e,b)$ est le voisin de e sur $\mathcal{R}$, alors $f(e,b) \leq q < r$, et r voit b. Donc r voit e par le lemme 3.2. Si r voit a, alors il y a un taureau e-rab-c, une contradiction, donc r manque a. Si r manque d, alors r, b, c, d, e est un trou, donc r voit d. Si $\mathcal{R}$ est de longueur > 3, alors $f(e,b) < q = f(a,e) < r$, r voit e et r manque a, une contradiction. Donc $\mathcal{R}$ est de longueur 3, q voit e et $q = f(e,b)$.

Puisque r voit e, manque a et $e < a < r$, nous avons $L_a(e) \neq L_a(a)$. Appliquons le lemme 3.2 pour définir $s = f(a,e)$. Le sommet s voit a, manque e et $r < s$. Puisque s voit a, nous avons $s \neq d$. Puisque s manque e et $q = f(e,b) < r = f(b,c) < s$, alors s manque b, c d'après le lemme 3.2. Si s voit d, alors s, a, b, c, d est un trou, donc s manque d. Si s voit q, alors b-asq-e est un taureau, donc s manque q. Si s voit r, alors c-der-s est un taureau, donc s manque r.

Puisque s voit a, manque q et $a < q < s$, nous avons $L_q(a) \neq L_q(q)$. Appliquons le lemme 3.2 pour définir $t = f(q,a)$. Le sommet t voit q, manque a et $s < t$. Puisque q voit a, alors t voit s d'après le lemme 3.4. Puisque t manque a et $q = f(e,b) < r = f(b,c) < s = f(a,e) < t$, alors t manque b, c, e d'après le lemme 3.2. Puisque t manque c, nous avons $t \neq d$. Si t voit r, alors t, r, b, a, s est un trou, donc t manque r, mais alors b-req-t est un taureau, une contradiction. □

Maintenant nous poursuivons la preuve du théorème. Puisque b voit c, manque e et $c < e < b$, nous avons $L_e(c) \neq L_e(e)$. Appliquons le lemme 3.2 pour définir $p = f(e,c)$. Le sommet p voit e, manque c et $b < p$. Puisque p voit e et manque c, nous avons $p \neq a$ et $p \neq d$. Si p voit a, alors p voit les extrémités du P_5 a, b, c, d, e sans voir c, une contradiction au lemme 3.6, donc p manque a. Si p voit b, alors p voit d, sinon p, b, c, d, e est un trou. Si p manque b, alors p manque d, sinon le mauvais P_5 a-b-c-d-p est plus mauvais que a-b-c-d-e. Donc p voit b, d ou p manque b, d.

Affirmation 3.9 *$a < b$.*

Preuve. Supposons que l'affirmation est fausse, donc $c < e < b < a$ d'après l'affirmation 3.8.

Cas 1 : $p < a$ et p voit b, d. Puisque a voit b, manque p et $b < p < a$, nous avons $L_p(b) \neq L_p(p)$. Appliquons le lemme 3.2 pour définir $q = f(p,b)$. Le sommet q voit p, manque b et $a < q$. Puisque p voit b, alors q voit a d'après le lemme 3.4. Puisque q voit a, nous avons $q \neq d$. Puisque $p = f(e,c) < q$, q voit e, c ou q manque e, c. Supposons que q manque e, c. Si q voit d, alors q, a, b, c, d est un trou, donc q manque d. Alors c-dep-q est un taureau, une contradiction. Donc q voit e, c. Puisque q voit c, manque b et $c < b < q$, nous

avons $L_b(c) \neq L_b(b)$. Appliquons le lemme 3.2 pour définir $r = f(b,c)$. Le sommet r voit b, manque c et $q < r$. Puisque b voit c, alors r voit q d'après le lemme 3.4. Puisque r voit b, manque c et $p = f(e,c) < q = f(p,b) < r$, alors r voit p et r manque e. Mais alors *c-brp-e* est un taureau, une contradiction.

Cas 2 : $p < a$ et p manque b, d. Puisque a voit b, manque p et $b < p < a$, nous pouvons considérer le chemin sans corde $R = \mathcal{P}(a,p,b)$ du lemme 3.3. Si aucun de c, d, e n'a de voisin dans R^*, alors $R \cup \{c,d,e\}$ est un trou de longueur au moins 7, donc l'un de c, d, e a un voisin dans R^*. Soit q le sommet de R^* le plus près de a qui voit l'un de c, d, e. Si q manque c, alors l'un de $R[b,q] \cup \{c,d\}$, $R[b,q] \cup \{c,d,e\}$ est un trou. Donc q voit c. Puisque q voit c et $p = f(e,c) < q$, alors q voit e. Le cycle sans corde $R[b,q] \cup \{c\}$ doit être de longueur < 5, donc q voit a et donc $q \neq d$. Puisque q voit c, manque b et $c < b < q$, nous avons $L_b(c) \neq L_b(b)$. Appliquons le lemme 3.2 pour définir $r = f(b,c)$. Le sommet r voit b, manque c et $q < r$. Puisque b voit c, alors r voit q d'après le lemme 3.4. Puisque r voit b, manque c et $p = f(e,c) < f(p,b) \leq q < r$, alors r voit p et r manque e. Supposons que $\mathcal{R}$ est de longueur > 3, alors $f(p,b) < q = f(a,p) < r$ et r voit p, donc r voit a et alors *c-bar-p* est un taureau, une contradiction. Donc $\mathcal{R}$ est de longueur 3 et q voit p. Mais alors q voit les extrémités du P_6 *a-b-c-d-e-p* sans voir b, une contradiction au lemme 3.6.

Cas 3 : $a < p$ et p voit b, d. Puisque p voit b, manque a et $b < a < p$, nous avons $L_a(b) \neq L_a(a)$. Appliquons le lemme 3.2 pour définir $q = f(a,b)$. Le sommet q voit a, manque b et $p < q$. Puisque a voit b, alors q voit p d'après le lemme 3.4. Puisque q voit a, nous avons $q \neq d$. Puisque $p = f(e,c) < q$, soit q voit e, c, soit q manque e, c. Supposons que q manque e, c. Si q voit d, alors q, a, b, c, d est un trou, donc q manque d. Alors *c-dep-q* est un taureau, une contradiction, donc q voit c, e. Puisque q voit c, manque b et $c < b < q$, nous avons $L_b(c) \neq L_b(b)$. Appliquons le lemme 3.2 pour définir $r = f(b,c)$. Le sommet r voit b, manque c et $q < r$. Puisque b voit c, alors r voit q d'après le lemme 3.4. Puisque r voit b, nous avons $r \neq d$. Puisque r voit b, manque c et $p = f(e,c) < q = f(a,b) < r$, alors r voit a et r manque e. Si r voit p, alors *c-brp-e* est un taureau, donc r manque p. Puisque r voit a, manque p et $a < p < r$, nous avons $L_p(a) \neq L_p(p)$. Appliquons le lemme 3.2 pour définir $s = f(p,a)$. Le sommet s voit p, manque a et $r < s$. Puisque s manque a et $p = f(e,c) < q = f(a,b) < r = f(b,c) < s$, alors s manque a, b, c, e. Puisque s manque c, nous avons $s \neq d$. Si s manque d, alors *c-dep-s* est un taureau, donc s voit d. Mais alors *a-b-c-d-s* est un mauvais P_5 qui est plus mauvais que *a-b-c-d-e*, une contradiction.

Cas 4 : $a < p$ et p manque b, d. Puisque p voit e, manque b et $e < b < p$, nous avons $L_b(e) \neq L_b(b)$. Appliquons le lemme 3.2 pour définir $q = f(b,e)$. Le sommet q voit b, manque e et $p < q$. Puisque q voit b, nous avons $q \neq d$.

Puisque q manque e et $p = f(e,c) < q$, alors q manque c. Si q manque d, alors q-b-c-d-e est plus mauvais que a-b-c-d-e, donc q voit d.

Cas 4.1 : q manque a. Puisque q voit b, manque a et $b < a < q$, nous avons $L_a(b) \neq L_a(a)$. Appliquons le lemme 3.2 pour définir $r = f(a,b)$. Le sommet r voit a, manque b et $q < r$. Puisque a voit b, alors r voit q d'après le lemme 3.4. Puisque r voit a, nous avons $r \neq d$. Puisque r manque b et $p = f(e,c) < q = f(b,e) < r$, alors r manque c, e. Si r voit d, alors a, b, c, d, r est un trou, donc r manque d. Si r voit p, alors a, b, c, d, e, p, r est un trou, donc r manque p. Puisque r voit a, manque p et $a < p < r$, nous pouvons considérer le chemin sans corde $R = \mathcal{P}(r, p, a)$ du lemme 3.3. Les sommets de R^* manquent a et $p = f(e,c) < q = f(b,e) < r = f(a,b)$, donc ils manquent b, c, e. Si d n'a pas de voisin dans R^*, alors $R \cup \{b, c, d, e\}$ est un trou de longueur au moins 8, donc d a un voisin dans R^*. Soit s le sommet de R^* le plus près de a qui voit d. Alors $R[a, s] \cup \{b, c, d\}$ est un trou, une contradiction.

Cas 4.2 : q voit a. Si q voit p, alors c-baq-p est un taureau, donc q manque p. Puisque q voit a, manque p et $a < p < q$, nous pouvons considérer le chemin sans corde $R = \mathcal{P}(q, p, a)$ du lemme 3.3. Puisque $p = f(e,c) < q = f(b,e)$, un sommet de R^* voit b, c, e ou manque b, c, e. Soit r le voisin de q dans R^*. Le sommet r manque a, et $f(p,a) \leq r$. Si r manque b, c, e, alors c-baq-r est un taureau, donc r voit b, c, e. Alors a-bcr-e est un taureau, une contradiction. □

Les affirmations 3.8 et 3.9 impliquent que $c < e < a < b$.

Puisque p voit e, manque a et $e < a < p$, nous avons $L_a(e) \neq L_a(a)$. Appliquons le lemme 3.2 pour définir $q = f(a,e)$. Le sommet q voit a, manque e et $p < q$. Puisque q voit a, nous avons $q \neq d$. Puisque d voit e, manque a, alors $d < q = f(a,e)$. Puisque q manque e et $p = f(e,c) < q$, alors q manque c. Si q voit d, alors q voit b, sinon q, a, b, c, d est un trou. Si q manque d, alors q manque b, sinon le mauvais P_5 q-b-c-d-e est plus mauvais que a-b-c-d-e. Donc q voit b, d ou q manque b, d.

Affirmation 3.10 *Le chemin q-a-b-c-d-e-p est sans corde.*

Preuve. Supposons que q voit p. Alors q voit les extrémités du chemin a-b-c-d-e-p sans voir c, donc, d'après le lemme 3.6, le chemin n'est pas sans corde, donc p voit b, d. Si q manque b, d, alors p voit les extrémités du chemin q-a-b-c-d-e sans voir c, une contradiction au lemme 3.6, donc q voit b, d. Mais alors c-bqp-e est un taureau, une contradiction. Donc q manque p.

Puisque q voit a, manque p et $a < p < q$, nous avons $L_p(a) \neq L_p(p)$. Appliquons le lemme 3.2 pour définir $r = f(p,a)$. Le sommet r voit p, manque

a et $q < r$. Puisque r manque a et $p = f(e,c) < q = f(a,e) < r$, alors r manque c, e.

Supposons que p voit b, d. Si r manque d, alors c-dep-r est un taureau, donc r voit d. Si r manque b, alors le mauvais P_5 a-b-c-d-r est plus mauvais que a-b-c-d-e, donc r voit b. Alors a-brp-e est un taureau, une contradiction. Donc p manque b, d.

Supposons que q voit b, d et r voit q. Si r manque b, alors c-baq-r est un taureau, donc r voit b. Si r manque d, alors le mauvais P_5 r-b-c-d-e est plus mauvais que a-b-c-d-e, donc r voit d. Alors e-drq-a est un taureau, une contradiction.

Supposons que q voit b, d et r manque q. Puisque r voit p, manque q et $p < q < r$, nous avons $L_q(p) \neq L_q(q)$. Appliquons le lemme 3.2 pour définir $s = f(q,p)$. Le sommet s voit q, manque p et $r < s$. Puisque s manque p et $p = f(e,c) < q = f(a,e) < r = f(p,a)$, alors s manque a, c, e. Si s manque b, alors c-baq-s est un taureau, donc s voit b. Si s manque d, alors le mauvais P_5 s-b-c-d-e est plus mauvais que a-b-c-d-e, donc s voit d. Alors e-dsq-a est un taureau, une contradiction. donc q manque b, d. □

Affirmation 3.11 $d < b$.

Preuve. Supposons que l'affirmation est fausse, alors $c < e < a < b < d$ d'après les affirmations 3.8 et 3.9.

Cas 1 : $L_d(b) \neq L_d(d)$. Appliquons le lemme 3.2 pour définir $s = f(d,b)$. Le sommet s voit d, manque b et $d < s$. Puisque s voit d, nous avons $s \neq p$. Supposons que s voit c. Si s manque e, alors b-csd-e est un taureau, donc s voit e. Si s manque a, alors le mauvais P_5 a-b-c-s-e est plus mauvais que a-b-c-d-e, donc s voit a. Si s manque p, alors a-sde-p est un taureau, donc s voit p. Alors b-cds-p est un taureau, une contradiction, donc s manque c. Si s voit a, alors a, b, c, d, s est un trou, donc s manque a. Alors le mauvais P_5 a-b-c-d-s est plus mauvais que a-b-c-d-e, une contradiction.

Cas 2 : $L_d(b) = L_d(d)$. Puisque a voit b et a manque d, nous avons $N(b) \neq N(d)$. Appliquons le lemme 3.5 pour définir $s = g(d,b)$. Le sommet s voit d, manque b, $s < b$, et $L_d(s) < L_d(d)$. Puisque s manque b, nous avons $s \neq a, c$. Puisque q voit a, manque d, nous avons $L_d(a) \neq L_d(d)$, et puisque a voit b et a manque d, nous avons $L'_d(a) \leq L'_d(s)$. Si s voit q, alors s voit les extrémités du P_5 q, a, b, c, d sans voir b, une contradiction au lemme 3.6, donc s manque q. Donc $L'_d(a) \neq L'_d(s)$. Appliquons le lemme 3.5 pour définir $t = h(s,a)$. Le sommet t voit s, manque a, b, d et $q < t$. Puisque t manque a et $p = f(e,c) < q = f(a,e) < t$, alors t manque c, e. Puisque t manque e, nous avons $s \neq e$. Si s voit c, alors b-cds-t est un taureau, donc s manque

c. Si s voit a, alors a, b, c, d, s est un trou, donc s manque a. Supposons que $s < e$. Puisque t voit s, manque e et $s < e < t$, nous avons $L_e(s) \neq L_e(e)$. Appliquons le lemme 3.2 pour définir $u = f(e, s)$. Le sommet u voit e, manque s et $t < u$. Puisque u voit e et $p = f(e, c) < q = f(a, e) < t < u$, alors u voit a, c. Puisque u voit a, manque s, $t = h(s, a) < u$ et $s = g(b, d)$, alors u voit b, d. Alors a-ucd-s est un taureau, une contradiction. Donc $e < s$. Alors le mauvais P_5 a-b-c-d-s est plus mauvais que a-b-c-d-e, une contradiction. □

Affirmation 3.12 $L_b(d) = L_b(b)$.

Preuve. Supposons que l'affirmation est fausse, donc $L_b(d) \neq L_b(b)$. Appliquons le lemme 3.2 pour définir $s = f(b, d)$. Le sommet s voit b, manque d et $b < s$. Puisque s voit b, nous avons $s \neq q$. Supposons que s voit c. Si s manque a, alors d-csb-a est un taureau, donc s voit a. Si s manque e, alors le mauvais P_5 a-s-c-d-e est plus mauvais que a-b-c-d-e, donc s voit e. Si s manque q, alors e-sba-q est un taureau, donc s voit q. Alors d-cbs-q est un taureau, une contradiction, donc s manque c. Si s voit e, alors b, c, d, e, s est un trou, donc s manque e. Alors s-b-c-d-e est un mauvais P_5 qui est plus mauvais que a-b-c-d-e, une contradiction. □

Affirmation 3.13 $a < d$.

Preuve. Supposons que l'affirmation est fausse, alors $d < a < b$. D'après le lemme 3.1, $L_b(d) \leq L_b(a) \leq L_b(b)$, et, d'après l'affirmation 3.12, $L_b(d) = L_b(b)$, donc $L_b(a) = L_b(b)$. Le sommet q voit a, manque b, et $a < b < q$, une contradiction. □

D'après les affirmations précédentes, nous avons $c < e < a < d < b < p = f(e, c) < q = f(a, e)$, $L_b(d) = L_b(b)$, et q-a-b-c-d-e-p est un chemin sans corde. Définissons les suites $(a_i), (b_i), (d_i), (e_i)$ de la manière suivante :

- $a_0 = a$, $b_0 = b$, $d_0 = d$, $e_0 = e$, $b_1 = q = f(a, e)$, $d_1 = p = f(e, c)$;
- Pour $i \geq 1$, $a_i = g(b_i, d_i)$, $e_i = g(d_i, b_{i-1})$;
- Pour $i \geq 2$, $b_i = h(a_{i-1}, e_{i-1})$, $d_i = h(e_{i-1}, a_{i-2})$;

Pour tout $k \geq 1$, nous disons que a-b-c-d-e admet une extension d'ordre k, notée $\mathcal{W}_k$, si les suites $(a_i)_{i<k}$, $(b_i)_{i\leq k}$, $(d_i)_{i\leq k}$, $(e_i)_{i<k}$ sont bien définies et vérifient les propriétés suivantes :

- $c < e_0 < a_0 < \cdots < e_{k-1} < a_{k-1} < d_0 < b_0 < \cdots < d_k < b_k$.
- $L_{b_{k-1}}(b_0) = \cdots = L_{b_{k-1}}(b_{k-1}) = L_{b_{k-1}}(d_0) = \cdots = L_{b_{k-1}}(d_{k-1})$.
- b_k-a_{k-1}-b_{k-1}-$\cdots$-b_1-a_0-b_0-c-d_0-e_0-d_1-$\cdots$-d_{k-1}-e_{k-1}-d_k est un chemin sans corde.

Les affirmations 3.8–3.13 et les définitions de p, q montrent que a-b-c-d-e admet une extension d'ordre 1. Soit k le plus grand entier tel que a-b-c-d-e admet une extension $\mathcal{W}_k$ d'ordre k. Nous allons prouver que a-b-c-d-e admet une extension d'ordre $k+1$, ce qui contredira la définition de k et terminera la preuve qu'il n'y a pas de mauvais P_5.

Affirmation 3.14 $L_{d_k}(b_{k-1}) = L_{d_k}(d_k)$

Preuve. Supposons au contraire que $L_{d_k}(b_{k-1}) \neq L_{d_k}(d_k)$. Puisque $b_{k-1} < d_k$ nous pouvons appliquer le lemme 3.2 pour définir $r = f(d_k, b_{k-1})$. Le sommet r voit d_k, manque b_{k-1} et $d_k < r$. Puisque r voit d_k, nous avons $r \neq b_k$. Puisque r manque b_{k-1} et $L_{b_{k-1}}(b_0) = \cdots = L_{b_{k-1}}(b_{k-1}) = L_{b_{k-1}}(d_0) = \cdots = L_{b_{k-1}}(d_{k-1})$, alors r manque $b_0, \ldots, b_{k-1}, d_0, \ldots, d_{k-1}$. Puisque r manque b_0, ..., b_{k-1}, d_0, ..., d_{k-1}, et $e_1 = g(d_1, b_0) < a_1 = g(b_1, d_1) < \ldots < a_{k-2} = g(b_{k-2}, d_{k-2}) < e_{k-1} = g(d_{k-1}, b_{k-2}) < d_1 = f(e_0, c) < b_1 = f(a_0, e_0) < d_2 = h(e_1, a_0) < b_2 = h(a_1, e_1) < \cdots < b_{k-1} = h(a_{k-2}, e_{k-2}) < d_k = h(e_{k-1}, a_{k-2}) < r$, alors soit r voit tous les sommets $c, a_0, \ldots, a_{k-2}, e_0, \ldots, e_{k-1}$, soit r les manque tous. Si r les voit tous, alors d_{k-1}-$e_{k-1}d_k r$-a_{k-2} est un taureau, donc r les manque tous. Si r voit l'un de a_{k-1}, b_k alors $\mathcal{W}_k \cup \{r\}$ contient un trou de longueur au moins six, une contradiction, donc r manque a_{k-1}, b_k.

Cas 1 : $r < b_k$. Puisque b_k voit a_{k-1}, manque b_{k-1} et $a_{k-1} < b_{k-1} < b_k$, nous avons $L_{b_{k-1}}(a_{k-1}) \neq L_{b_{k-1}}(b_{k-1})$. Appliquons le lemme 3.2 pour définir $s = f(b_{k-1}, a_{k-1})$. Le sommet s voit b_{k-1}, manque a_{k-1} et $b_k < s$. Puisque b_{k-1} voit a_{k-1}, alors s voit b_k d'après le lemme 3.4. Puisque s voit b_{k-1} et $r = f(d_k, b_{k-1}) < b_k < s$, alors s voit d_k. Puisque s voit b_k, d_k et s manque a_{k-1}, alors, d'après le lemme 3.6, r voit b_0, ..., b_k, d_0, ..., d_k et r manque c, a_0, ..., a_{k-1}, e_0, ..., e_{k-1}. Si s voit r, alors e_{k-1}-$d_k r s$-b_k est un taureau, donc s manque r.

Puisque s voit d_k, manque r et $d_k < r < s$, nous avons $L_r(d_k) \neq L_r(r)$. Appliquons le lemme 3.2 pour définir $t = f(r, d_k)$. Le sommet t voit r, manque d_k et $s < t$. Puisque r voit d_k, alors t voit s d'après le lemme 3.4. Puisque t manque d_k et $r = f(d_k, b_{k-1}) < s = f(b_{k-1}, a_{k-1}) < t$, alors t manque a_{k-1}, b_{k-1}. Puisque t manque b_{k-1} et $L_{b_{k-1}}(b_0) = \cdots = L_{b_{k-1}}(b_{k-1}) = L_{b_{k-1}}(d_0) = \cdots = L_{b_{k-1}}(d_{k-1})$, alors t manque tout $b_0, \ldots, b_{k-1}, d_0, \ldots, d_{k-1}$. Puisque t manque a_{k-1}, b_0, ..., b_{k-1}, d_0, ..., d_k, et $e_1 = g(d_1, b_0) < a_1 = g(b_1, d_1) < \ldots < e_{k-1} = g(d_{k-1}, b_{k-2}) < a_{k-1} = g(b_{k-1}, d_{k-1}) < d_1 = f(e_0, c) < b_1 = f(a_0, e_0) < d_2 = h(e_1, a_0) < b_2 = h(a_1, e_1) < \cdots < d_k = h(e_{k-1}, a_{k-2}) < b_k = h(a_{k-1}, e_{k-1}) < t$, alors t manque tout $c, a_0, \ldots, a_{k-1}, e_0, \ldots, e_{k-1}$.

Puisque t voit r, manque b_k et $r < b_k < t$, nous pouvons considérer le chemin sans corde $R = \mathcal{P}(t, b_k, r)$ du lemme 3.3. Tout sommet u de R^*

manque r et satisfait $t = f(r, d_k) < u$, donc u manque d_k. Le cycle $R \cup \mathcal{W}_k$ est de longueur au moins 10, donc l'un de $\mathcal{W}_k \setminus \{b_k\}$ a un voisin dans R^*. Soit u le sommet de R^* le plus proche de t qui voit l'un de $\mathcal{W}_k \setminus \{b_k\}$, alors $R[u, r] \cup \mathcal{W}_k$ contient un trou, une contradiction.

Cas 2 : $b_k < r$. Puisque r voit d_k, manque b_k et $d_k < b_k < r$, nous pouvons considérer le chemin sans corde $R = \mathcal{P}(r, b_k, d_k)$ du lemme 3.3. Tout sommet u de R^* manque d_k et satisfait $r = f(d_k, b_{k-1}) < u$, donc u manque b_{k-1}. Alors, puisque $L_{b_{k-1}}(b_0) = \cdots = L_{b_{k-1}}(b_{k-1}) = L_{b_{k-1}}(d_0) = \cdots = L_{b_{k-1}}(d_{k-1})$, le sommet u manque tout $b_0, \ldots, b_{k-1}, d_0, \ldots, d_{k-1}$. Puisque u manque b_0, ..., b_{k-1}, d_0, ..., d_k, et $e_1 = g(d_1, b_0) < a_1 = g(b_1, d_1) < \ldots < e_{k-1} = g(d_{k-1}, b_{k-2}) < a_{k-1} = g(b_{k-1}, d_{k-1}) < d_1 = f(e_0, c) < b_1 = f(a_0, e_0) < d_2 = h(e_1, a_0) < b_2 = h(a_1, e_1) < \cdots d_k = h(e_{k-1}, a_{k-2}) < b_k = h(a_{k-1}, e_{k-1}) < u$, soit le sommet u voit tous les sommets $c, a_0, \ldots, a_{k-1}, e_0, \ldots, e_{k-1}$, soit il les manque tous.

Soit t le voisin de b_k dans R^*, donc $t = f(b_k, d_k)$. Si t voit $c, a_0, \ldots, a_{k-1}$, $e_0, \ldots, e_{k-1}$, alors b_{k-1}-$a_{k-1}b_k t$-e_{k-1} est un taureau. donc t manque $c, a_0, \ldots, a_{k-1}$, $e_0, \ldots, e_{k-1}$. Si t voit r, alors $\mathcal{W}_k \cup \{r, t\}$ est un trou, donc t manque r.

Soit u le voisin de r dans R^*, donc $u = f(r, b_k)$. Le sommet u manque b_0, ..., b_k, d_0, ..., d_k. Si u manque $c, a_0, \ldots, a_{k-1}, e_0, \ldots, e_{k-1}$, alors $R \cup \mathcal{W}_k$ contient un trou, donc u voit $c, a_0, \ldots, a_{k-1}, e_0, \ldots, e_{k-1}$.

Puisque u voit c, manque b_0 et $c < b_0 < u$, nous avons $L_{b_0}(c) \neq L_{b_0}(b_0)$. Appliquons le lemme 3.2 pour définir $v = f(b_0, c)$. Le sommet v voit b_0, manque c et $u < v$. Puisque b_0 voit c, alors v voit u d'après le lemme 3.4. Puisque v voit b_0 et $L_{b_{k-1}}(b_0) = \cdots = L_{b_{k-1}}(b_{k-1}) = L_{b_{k-1}}(d_0) = \cdots = L_{b_{k-1}}(d_{k-1})$, alors v voit $b_0, \ldots, b_{k-1}, d_0, \ldots, d_{k-1}$. Puisque v voit b_{k-1}, manque c et $d_1 = f(e_0, c) < r = f(d_k, b_{k-1}) < t = f(b_k, d_k) < u = f(r, b_k) < v$, alors v voit d_k, b_k, r et v manque e_0. Mais alors b_0-vru-e_0 est un taureau, une contradiction. □

Affirmation 3.15 $L_{d_k}(b_0) = \cdots = L_{d_k}(b_{k-1}) = L_{d_k}(d_0) = \cdots = L_{d_k}(d_k)$

Preuve. D'après l'affirmation 3.14, $L_{d_k}(b_{k-1}) = L_{d_k}(d_k)$, et $L_{b_{k-1}}(b_0) = \cdots = L_{b_{k-1}}(b_{k-1}) = L_{b_{k-1}}(d_0) = \cdots = L_{b_{k-1}}(d_{k-1})$, et $b_{k-1} < d_k$, donc $L_{d_k}(b_0) = \cdots = L_{d_k}(b_{k-1}) = L_{d_k}(d_0) = \cdots = L_{d_k}(d_k)$. □

Puisque a_{k-1} voit b_{k-1} et manque d_k, nous avons $N(b_{k-1}) \neq N(d_k)$. Appliquons le lemme 3.5 pour définir $e_k = g(d_k, b_{k-1})$. Le sommet e_k voit d_k, manque b_{k-1}, $e_k < b_{k-1}$, et $L_{d_k}(e_k) < L_{d_k}(d_k) = L_{d_k}(d_0)$. Puisque $L_{d_k}(e_k) < L_{d_k}(d_0)$, donc $e_k < d_0$ d'après le lemme 3.1. Puisque e_k voit d_k, alors $e_k \notin \mathcal{W}_k \setminus \{e_{k-1}\}$. Puisque a_{k-1} voit b_{k-1}, manque d_k, alors

$L'_{d_k}(a_{k-1}) \leq L'_{d_k}(e_k)$. Si e_k voit b_k, alors e_k voit les extrémités du chemin sans corde $\mathcal{W}_k$ sans voir b_{k-1}, une contradiction au lemme 3.6, donc e_k manque b_k. Donc $L'_{d_k}(a_{k-1}) < L'_{d_k}(e_k)$. Appliquons le lemme 3.5 pour définir $d_{k+1} = h(e_k, a_{k-1})$. Le sommet d_{k+1} voit e_k, manque a_{k-1}, b_{k-1}, d_k et $b_k < d_{k+1}$.

Affirmation 3.16 *$\mathcal{W}_k$-e_k-d_{k+1} est un chemin sans corde.*

Preuve. Puisque d_{k+1} manque d_k, et $L_{d_k}(b_0) = \cdots = L_{d_k}(b_{k-1}) = L_{d_k}(d_0) = \cdots = L_{d_k}(d_k)$, alors d_{k+1} manque $b_0, \ldots, b_{k-1}, d_0, \ldots, d_k$. Puisque d_{k+1} manque a_{k-1}, b_0, ..., b_{k-1}, d_0, ..., d_k, et $e_1 = g(d_1, b_0) < a_1 = g(b_1, d_1) < \ldots < e_{k-1} = g(d_{k-1}, b_{k-2}) < a_{k-1} = g(b_{k-1}, d_{k-1}) < d_1 = f(e_0, c) < b_1 = f(a_0, e_0) < d_2 = h(e_1, a_0) < b_2 = h(a_1, e_1) < \cdots d_k = h(e_{k-1}, a_{k-2}) < b_k = h(a_{k-1}, e_{k-1}) < t$, alors d_{k+1} manque $c, a_0, \ldots, a_{k-1}, e_0, \ldots, e_{k-1}$. Puisque d_{k+1} manque e_{k-1}, nous avons $e_k \neq e_{k-1}$. Si d_{k+1} voit b_k, alors e_k voit les extrémités du chemin sans corde $\mathcal{W}_k \cup \{d_{k+1}\}$ sans voir b_{k-1}, une contradiction au lemme 3.6, donc d_{k+1} manque b_k.

Supposons que e_k voit d_{k-1}. Considérons l'étape générale de l'algorithme où b_{k-1} est choisi. Puisque $L_{d_k}(e_k) < L_{d_k}(d_k) = L_{d_k}(b_{k-1})$, alors $L_{b_{k-1}}(e_k) < L_{b_{k-1}}(b_{k-1})$, d'après le lemme 3.1. Puisque $L'_{d_k}(a_{k-1}) < L'_{d_k}(e_k)$, et $L_{d_k}(b_{k-1}) = L_{d_k}(d_k)$, alors $L'_{b_{k-1}}(a_{k-1}) < L'_{b_{k-1}}(e_k)$. L'ensemble U de l'étape 1 de l'algorithme contient e_k puisque $L_{b_{k-1}}(e_k) < L_{b_{k-1}}(b_{k-1})$. Puisque $L'_{b_{k-1}}(a_{k-1}) < L'_{b_{k-1}}(e_k)$, le sommet e_k est choisi dans U à l'étape 2.1 avant a_{k-1}. Alors à l'étape 2.2, $A \cap N(e_k)$ doit être vide, car sinon b_{k-1} est retiré de A et b_{k-1} n'est pas le sommet sélectionné à l'étape 3. Puisque d_{k-1} est dans $N(e_k)$, il a déjà été retiré de A par un sommet u avec $L'_{b_{k-1}}(e_k) \leq L'_{b_{k-1}}(u)$. Puisque $L'_{b_{k-1}}(a_{k-1}) < L'_{b_{k-1}}(e_k) \leq L'_{b_{k-1}}(u)$, nous avons $u \neq a_{k-1}$. Ce qui contredit la définition de a_{k-1}, donc e_k manque d_{k-1}.

Si e_k voit e_{k-1}, alors d_{k-1}-$e_{k-1}d_ke_k$-d_{k+1} est un taureau, donc e_k manque e_{k-1}. Si e_k voit l'un de b_0, ..., b_{k-2}, d_0, ..., d_{k-2}, $c, a_0, \ldots, a_{k-1}, e_0, \ldots, e_{k-2}$, alors $\mathcal{W}_k \cup \{s\}$ contient un trou, donc e_k manque b_0, ..., b_{k-2}, d_0, ..., d_{k-2}, $c, a_0, \ldots, a_{k-2}, e_0, \ldots, e_{k-2}$. □

Affirmation 3.17 $a_{k-1} < e_k$.

Preuve. Supposons que l'affirmation est fausse, donc $e_k < a_{k-1}$. Puisque d_{k+1} voit e_k, manque a_{k-1} et $e_k < a_{k-1} < d_{k+1}$, nous avons $L_{a_{k-1}}(e_k) \neq L_{a_{k-1}}(a_{k-1})$. Appliquons le lemme 3.2 pour définir $u = f(a_{k-1}, e_k)$. Le sommet u voit a_{k-1}, manque e_k et $d_{k+1} < u$. Puisque u voit a_{k-1}, manque e_k, $d_{k+1} = h(e_k, a_{k-1}) < u$, et $e_k = g(d_k, b_{k-1})$, alors u voit d_k, b_{k-1}. Puisque u voit les extrémités du chemin sans corde $\mathcal{W}_k \setminus \{b_k\}$, d'après le lemme 3.6, il doit voir

tous les sommets de $\mathcal{W}_k \setminus \{b_k\}$. Mais alors a_{k-1}-$ue_{k-1}d_k$-e_k est un taureau, une contradiction. □

Les affirmations 3.15, 3.16, et 3.17 et les définitions de e_k, d_{k+1}, montrent que les suites $(a_i)_{i<k}$, $(b_i)_{i\leq k}$, $(d_i)_{i\leq k+1}$, $(e_i)_{i<k+1}$ sont bien définies et satisfont les propriétés suivantes :

- $c < e_0 < a_0 < \cdots < a_{k-1} < e_k < d_0 < b_0 < \cdots << b_k < d_{k+1}$.
- $L_{d_k}(b_0) = \cdots = L_{d_k}(b_{k-1}) = L_{d_k}(d_0) = \cdots = L_{d_k}(d_k)$.
- $\mathcal{W}_k$-e_k-d_{k+1} est un chemin sans corde.

Une preuve similaire permet de définir les sommets $a_k = g(b_k, d_k)$ et $b_{k+1} = h(a_k, e_k)$ et de montrer qu'ils vérifient les propriétés suivantes :

- $c < e_0 < a_0 < \cdots < e_k < a_k < d_0 < b_0 < \cdots < d_{k+1} < b_{k+1}$.
- $L_{b_k}(b_0) = \cdots = L_{b_k}(b_k) = L_{b_k}(d_0) = \cdots = L_{b_k}(d_k)$.
- b_{k+1}-a_k-$\mathcal{W}_k$-e_k-d_{k+1} est un chemin sans corde.

Cela signifie que a-b-c-d-e admet une extension d'ordre $k+1$. Ce qui contredit la définition de k et termine la preuve du théorème. □

3.4 L'algorithme OrdreCosine*

L'algorithme Cosine* est un cas particulier de l'algorithme Cosine de Hertz [45]. La différence entre Cosine et Cosine* est que les sommets du graphe en entrée de Cosine* sont ordonnés selon un ordre σ. Les cas d'égalité de Cosine sont départagés en utilisant cet ordre.

A chaque étape, le sommet choisi et colorié c est celui qui n'a pas de voisin déjà colorié c et qui a le nombre maximum de voisins non coloriés avec les sommets déjà coloriés c, les cas d'égalité sont départagés en prenant le sommet qui est minimum pour σ.

L'algorithme Cosine* est défini de manière formelle dans [58]. Dans cette thèse, nous préférons formuler l'algorithme de telle sorte qu'il produise un ordre de contraction. Nous définissons ainsi l'algorithme OrdreCosine* qui combine les idées des algorithmes OrdreCosine et OrdreContraction présenté dans le chapitre 2. Voici une description formelle de l'algorithme :

Algorithme OrdreCosine*

Entrée : Un graphe G et un ordre σ sur ses sommets.

Sortie : Un ordre de contraction $\{x_1^1, \ldots, x_1^{k_1}, \ldots, x_\ell^1, \ldots, x_\ell^{k_\ell}\}$ et la coloration associée.

Calcul :

1. Soient $c = 1$ et $i = 1$;
2. tant qu'il existe des sommets non coloriés :

2.1. s'il existe un sommet non colorié qui n'a pas de voisin colorié c :
2.1.1 soit A l'ensemble des sommets non coloriés qui ont un voisin colorié c ;
2.1.2. choisir un sommet non colorié v qui n'a pas de voisin colorié c et qui a le nombre maximum de voisins dans A. Les cas d'égalité sont départagés en prenant le sommet minimum pour σ ;
2.1.3 colorier v avec la couleur c, poser $x_c^i = v$ et $i = i + 1$;
2.2 sinon, poser $c = c + 1$ et $i = 1$.
COMPLEXITÉ : $\mathcal{O}(mn)$.

Analyse de complexité

Pour analyser la complexité de l'algorithme ORDRECOSINE*, le graphe en entrée est considéré connexe, alors $m \geq n - 1$. Si le graphe n'est pas connexe il suffit d'appliquer l'algorithme sur chacune de ses composantes connexes. Le fait de départager les cas d'égalité dans ORDRECOSINE* n'augmente pas la complexité de l'algorithme ORDRECOSINE. L'algorithme ORDRECOSINE* peut donc être implémenté en temps $\mathcal{O}(nm)$ de la manière suivante. La mise à jour de l'ensemble A à l'étape 2.1.1 peut être faite en temps $\mathcal{O}(d(u))$: lorsqu'un nouveau sommet u est colorié à l'étape 2.1.3, les voisins non coloriés de u sont ajoutés à A. Pour une couleur donnée c, cette procédure prend un temps $\mathcal{O}(n + m)$, donc le coût total est $\mathcal{O}(nm)$ sur toutes les couleurs. Pour exécuter l'étape 2.1.2 efficacement, un compteur est utilisé pour chaque sommet, il représente le nombre de ses voisins dans A. A chaque fois qu'un sommet est ajouté à A, le compteur des autres sommets est mis à jour ; cela peut être fait en temps $\mathcal{O}(n+m)$ pour une couleur donnée et donc en temps $\mathcal{O}(nm)$ pour toutes les couleurs. Ensuite, tous les sommets sont parcourus en temps $\mathcal{O}(n)$ pour trouver le sommet non colorié qui a un compteur maximum et qui est minimum pour σ. Après chacun de ces parcours, un sommet est marqué, donc le coût total de tous les parcours est $\mathcal{O}(n^2)$. Par conséquence, le temps total d'exécution de l'algorithme ORDRECOSINE* est $\mathcal{O}(nm)$.

3.5 Amicalité

Prouvons que, pour tout graphe G dans $\mathcal{B}$, l'algorithme ORDRECOSINE* transforme un ordre d'élimination $\overline{P_3}$-simplicial en un ordre de contraction amical.

Nous devons définir une superclasse de $\mathcal{B}$. Un graphe G est *quasi-$\mathcal{B}$* si G ne contient ni trou impair, ni antitrou et G possède un sommet, appelé *pivot*, qui est l'oreille de tous les taureaux de G. (Cette définition peut être

comparée à la définition des graphes quasi-Meyniel dans [45].) Remarquons que tout graphe dans la classe $\mathcal{B}$ est un graphe quasi-$\mathcal{B}$ (tout sommet est un pivot). Si G est un graphe quasi-$\mathcal{B}$ et z est un pivot, alors $G \setminus z$ est dans la classe $\mathcal{B}$.

Le lemme suivant généralise le lemme 3.6 aux graphes quasi-$\mathcal{B}$.

Lemme 3.18 *Dans un graphe G* quasi-$\mathcal{B}$*, soit $P = a_0$-a_1-...-a_r un chemin sans corde impair avec $r \geq 5$, où a_0 est un pivot de G et u un sommet qui voit les deux extrémités a_0, a_r de P. Alors u voit a_2.*

Preuve. Supposons que le lemme est faux et u manque a_2. Si u voit a_1, alors a_r-ua_0a_1-a_2 est un taureau pour lequel a_0 n'est pas une oreille, une contradiction. Donc u manque a_1. Appelons *segment* tout sous-chemin de P, de longueur au moins 1, dont les extrémités voient u et dont les sommets intérieurs manquent u. Le chemin P est partitionné (au sens des arêtes) en ses segments. Puisque G ne contient pas de trou impair, chaque segment est de longueur 1 ou paire. Puisque P est impair, il y a au moins un segment de longueur 1. Soit i le plus petit entier tel que u voit a_i et a_{i+1}. Puisque u manque a_1, a_2, nous avons $i \geq 3$. Alors a_{i-1}-$a_ia_{i+1}u$-a_0 est un taureau pour lequel a_0 n'est pas une oreille, une contradiction. □

Maintenant nous prouvons le théorème suivant.

Théorème 3.19 *Soient G un graphe dans $\mathcal{B}$ et σ un ordre d'élimination $\overline{P_3}$-simplicial de G. Alors, l'algorithme* OrdreCosine* *appliqué sur cette entrée produit un ordre de contraction amical de G.*

Preuve du théorème. Soit $\{x_1^1, \ldots, x_1^{k_1}, \ldots, x_\ell^1, \ldots, x_\ell^{k_\ell}\}$ l'ordre de contraction produit par l'algorithme. Considérons les graphes G_c^i et les sommets w_c^i ($1 \leq c \leq \ell$, $1 \leq i \leq k_c$) qui forment la suite de contractions associée à cet ordre. Pour tout $1 \leq c \leq \ell$, dans le graphe $G_c^{k_c}$, remarquons que le sommet w_c est adjacent à tous les autres sommets de G_c ; sinon, il existe un sommet y qui n'est pas adjacent à $w_c^{k_c}$, ce qui signifie que y n'a aucun voisin colorié c et donc l'algorithme aurait dû colorier plus de sommets avec la couleur c, une contradiction.

Affirmation 3.20 *Pour toute couleur $1 \leq c \leq \ell$ et tout entier $1 \leq i \leq k_c - 1$, si G_c^i est un graphe quasi-$\mathcal{B}$, w_c^i est un pivot et n'est pas le toit d'une maisonnette de G_c^i, alors il n'y a pas de chemin sans corde impair entre w_c^i et x_c^{i+1} dans G_c^i.*

Preuve. Supposons au contraire qu'il existe un chemin sans corde impair $P = a_0\text{-}a_1\text{-}\cdots\text{-}a_{r-1}\text{-}a_r$ entre $a_0 = w_c^i$ et $a_r = x_c^{i+1}$ dans G_c^i. Nous avons $r \geq 3$ puisque w_c^i, x_c^{i+1} ne sont pas adjacents. Remarquons que tout sommet de P a un non-voisin dans G_c^i. Soit $W_c = \{w_1, \ldots, w_{c-1}\}$. Un sommet $w \in W_c$ est un sommet de G_c^i adjacent à tous les sommets de $G_c^i \setminus w$, donc P ne contient pas de sommet de W_c. A la fin de l'algorithme, tout sommet de $G_c^i \setminus W_c$ aura reçu une couleur de l'ensemble $\{c, c+1, \ldots, \ell\}$.

Considérons le moment où l'algorithme OrdreCosine* choisit x_c^{i+1}. Soit A l'ensemble défini lors de l'étape 2.1.1 de l'algorithme. Le sommet a_1 est dans A et a_2 n'est pas dans A. Soit $T = N(x_c^{i+1}) \cap A$. Chaque sommet de T est adjacent à au moins un sommet colorié c dans G et est donc adjacent à w_c^i dans G_c^i.

Supposons qu'il existe un sommet $t \in T$ qui manque a_2. Si $r = 3$, alors soit t manque a_1 et alors t, a_0, a_1, a_2, a_3 induit un trou impair, soit t voit a_1 et alors a_0 est le toit d'une maisonnette , dans les deux cas une contradiction. Donc $r \geq 5$. Le sommet t voit les extremités du chemin sans corde impair P sans voir a_2, une contradiction au lemme 3.18. Donc tout sommet de T voit a_2. Alors $T \cup \{a_1\} \subset N(a_2) \cap A$, donc a_2 a strictement plus de voisins dans A que x_c^{i+1}, ce qui contredit le fait que x_c^{i+1} est choisi lors de l'étape 2.1.2. □

Affirmation 3.21 *Pour toute couleur $1 \leq c \leq \ell$ et tout entier $0 \leq i \leq k_c - 1$, les deux propriétés suivantes sont vraies :*

(A_i) Si $i \geq 1$, alors w_c^i et x_c^{i+1} forment une paire d'amis de G_c^i.

(B_i)

(1) G_c^{i+1} est un graphe quasi-$\mathcal{B}$

(2) w_c^{i+1} est un pivot de G_c^{i+1}

(3) w_c^{i+1} n'est pas le toit d'une maisonnette dans G_c^{i+1}.

Preuve. Pour $1 \leq c \leq \ell$, prouvons par induction sur i que (A_i) et (B_i) sont vraies.

La propriété (A_0) est vraie par vacuité pour $1 \leq c \leq \ell$.

Montrons que (B_0) est vraie pour $c = 1$. Considérons la propriété (B_0). Le graphe $G_1^1 = G$ est un graphe dans $\mathcal{B}$, dont tout sommet est un pivot, donc (1) et (2) sont vérifiées. Pour montrer (3), considérons le début de l'algorithme OrdreCosine* : l'ensemble A de l'étape 2.1.1 est vide, donc w_1^1 est le sommet minimum pour σ. Puisque l'ordre σ est $\overline{P_3}$-simplicial, le sommet w_1^1 n'est pas le toit d'une maisonnette dans G_1^1.

Montrons que (B_0) est vraie pour $2 \leq c \leq \ell$. Dans le graphe G_c^1, tout sommet w_h, avec $1 \leq h \leq c-1$ est adjacent à tous les autres sommets du graphe ; de plus $G_c^1 \setminus \{w_1, \ldots, w_{c-1}\}$ est dans $\mathcal{B}$, puisque c'est un sous-graphe de G. Donc G_c^1 est dans $\mathcal{B}$, et w_c^1 est un pivot de ce graphe. A cette

étape de l'algorithme OrdreCosine*, l'ensemble A de l'étape 2.1.1 est vide, donc à l'étape 2.1.2, les sommets de $G_c^1 \setminus \{w_1, \ldots, w_{c-1}\}$ n'ont pas de voisin colorié c et ont le nombre maximum de voisins dans A, donc le sommet $w_c^1 = x_c^1$ choisi est celui qui est minimum pour σ dans $G_c^1 \setminus \{w_1, \ldots, w_{c-1}\}$. Comme σ est un ordre d'élimination $\overline{P_3}$-simplicial, ce sommet n'est pas le toit d'une maisonnette dans $G_c^1 \setminus \{w_1, \ldots, w_{c-1}\}$. Puisque tout sommet w_h, avec $1 \leq h \leq c-1$ est adjacent à tous les autres sommets du graphe, le sommet w_c^1 n'est pas le toit d'une maisonnette dans G_c^1.

Supposons maintenant que $1 \leq c \leq l$, $i \geq 1$ et (A_{i-1}) et (B_{i-1}) sont vérifiées. L'affirmation 3.20 implique immédiatement que (A_i) est vérifiée. Il reste à prouver (B_i). D'après (A_i), (B_{i-1}) et les lemmes 2.7 et 2.9 du chapitre 2, le graphe G_c^{i+1} ne contient pas de trou impair et pas d'antitrou autre que $\overline{C_6}$.

Supposons que G_c^{i+1} contient un $\overline{C_6}$, ayant pour sommets $a_1, a_2, a_3, a_4, a_5, a_6$ et pour non-arêtes a_1a_2, a_2a_3, a_3a_4, a_4a_5, a_5a_6, a_6a_1. Si w_c^{i+1} n'est pas l'un des a_i, alors ce $\overline{C_6}$ est aussi contenu dans G_c^i, une contradiction. Donc, par symétrie, nous pouvons supposer que $w_c^{i+1} = a_1$. D'après la définition de la contraction, w_c^i et x_c^{i+1} manquent tous les deux a_6 et a_2, et chaque sommet a_3, a_4, a_5 voit au moins un sommet de w_c^i, x_c^{i+1}. L'un de w_c^i, x_c^{i+1} voit à la fois a_3, a_5, car sinon $\{w_c^i, x_c^{i+1}, a_3, a_5\}$ induit un chemin sans corde impair entre w_c^i et x_c^{i+1}, une contradiction à (A_i). Soit u un sommet de w_c^i, x_c^{i+1} qui voit a_3, a_5, et soit v l'autre. Aucun de u, v ne voit à la fois a_3, a_4, a_5, car sinon un $\overline{C_6}$ est contenu dans G_c^i. Donc u manque a_4, et donc v voit a_4 et manque au moins l'un de a_3, a_5. Par symétrie, nous pouvons supposer que v manque a_3. Mais alors v-$a_4a_2a_6$-a_3 est un taureau de G_c^i pour lequel w_c^i n'est pas une oreille, une contradiction. Donc G_c^{i+1} ne contient pas de $\overline{C_6}$.

Supposons que G_c^{i+1} contient un taureau a_1-$a_2a_3a_4$-a_5 tel que w_c^{i+1} n'est pas une oreille de ce taureau. Si w_c^{i+1} n'est pas dans le taureau, alors le taureau est aussi contenu dans G_c^i et w_c^i n'est pas dedans, ce qui contredit le fait que w_c^i est un pivot de G_c^i. Donc, par symétrie, nous pouvons supposer que $w_c^{i+1} = a_1$ ou $w_c^{i+1} = a_3$. Si $w_c^{i+1} = a_1$, alors w_c^i, x_c^{i+1} manquent a_3, a_4, a_5, et au moins l'un de w_c^i, x_c^{i+1} voit a_2 ; mais alors G_c^i contient un taureau pour lequel w_c^i n'est pas une oreille, une contradiction. Si $w_c^{i+1} = a_3$, alors w_c^i, x_c^{i+1} manque a_1, a_5, et au moins l'un de w_c^i, x_c^{i+1} voit a_2, a_4, car sinon $\{w_c^i, x_c^{i+1}, a_2, a_4\}$ induit un chemin sans corde impair entre w_c^i et x_c^{i+1}, une contradiction à (A_i). Mais alors G_c^i contient un taureau pour lequel w_c^i n'est pas une oreille, une contradiction.

D'après les deux paragraphes précédents, G_c^{i+1} est un graphe quasi-$\mathcal{B}$ et w_c^{i+1} est un pivot de G_c^{i+1}.

Supposons maintenant que w_c^{i+1} est le toit d'une maisonnette dans G_c^{i+1}

ayant pour sommets a_1, a_2, a_3, a_4, a_5 et non-arêtes a_1a_2, a_2a_3, a_3a_4, a_4a_5. Donc $w_c^{i+1} = a_3$. Dans G_c^i, les sommets w_c^i, x_c^{i+1} manquent a_2, a_4. Le sommet w_c^i manque au moins l'un de a_1, a_5, car sinon c'est le toit d'une maisonnette dans G_c^i, une contradiction à (B_{i-1}). Par symétrie, nous pouvons supposer que w_c^i manque a_5, et donc x_c^{i+1} voit a_5. Alors x_c^{i+1} voit aussi a_1, car sinon w_c^i-a_1-a_5-x_c^{i+1} est un chemin qui contredit (A_i). Alors w_c^i manque a_1, car sinon w_c^i-$a_1x_c^{i+1}a_5$-a_2 est un taureau dans G_c^i pour lequel w_c^i n'est pas une oreille. Remarquons que, dans G_c^i, les sommets $a_1, a_2, x_c^{i+1}, a_4, a_5$ induisent une maisonnette, dont x_c^{i+1} est le toit, et w_c^i manque tous ces sommets. Considérons le moment où l'algorithme OrdreCosine* choisit x_c^{i+1}. Soit A l'ensemble défini à l'étape 2.1.1 de l'algorithme. Puisque w_c^i manque tous les a_i, aucun d'entre eux n'est dans A. Soit $T = N(x_c^{i+1}) \cap A$. Considérons n'importe quel sommet t de T. D'après la définition de T, le sommet t voit x_c^{i+1} et w_c^i dans G_c^i. Si t manque à la fois a_1, a_5, alors t voit a_4, car sinon t-$x_c^{i+1}a_5a_1$-a_4 est un taureau dans G_c^i pour lequel w_c^i n'est pas une oreille, et similairement t voit a_2, mais alors w_c^i-ta_4a_2-a_5 est un taureau de G_c^i pour lequel w_c^i n'est pas une oreille. Donc t voit au moins l'un de a_1, a_5, par exemple a_1. Alors t voit a_4, car sinon w_c^i-$tx_c^{i+1}a_1$-a_4 est un taureau de G_c^i pour lequel w_c^i n'est pas une oreille. Alors t voit a_2, car sinon w_c^i-ta_1a_4-a_2 est un taureau de G_c^i pour lequel w_c^i n'est pas une oreille. Alors t voit a_5, car sinon w_c^i-ta_4a_2-a_5 est un taureau de G_c^i pour lequel w_c^i n'est pas une oreille. Donc tout sommet de T voit a_1, a_2, a_4, a_5. A ce moment a_1, a_2, a_4, a_5 sont tous des sommets non coloriés qui n'ont pas de voisin colorié c et qui ont au moins autant de voisins dans A que x_c^{i+1}, donc ils ont le nombre maximum de voisins dans A, et selon l'ordre σ nous avons $x_c^{i+1} < min\{a_1, a_2, a_4, a_5\}$ et x_c^{i+1} n'est pas le toit de la maisonnette, une contradiction. □

L'affirmation 3.21 montre que l'ordre de contraction produit par l'algorithme est un ordre de contraction amical. □

3.6 Commentaires

D'après les théorèmes 3.7 et 3.19, nous pouvons colorier optimalement un graphe G dans $\mathcal{B}$, en appliquant l'algorithme LexBFS* sur $\overline{G}$, pour obtenir un ordre $\overline{P_3}$-simplicial, suivi de l'algorithme OrdreCosine* sur G, pour obtenir un ordre de contraction amical. La complexité de cet algorithme de coloration que nous pouvons noté $\overline{\text{LexBFS*}}$+OrdreCosine* est $\mathcal{O}(nm)$.

Etant donné que, sur un graphe dans $\mathcal{B}$, l'algorithme $\overline{\text{LexBFS*}}$ produit un ordre, nous pouvons nous demander si cet ordre n'est pas déjà un ordre

de contraction amical. Est-il vraiment nécessaire d'appliquer ensuite l'algorithme OrdreCosine* pour colorier le graphe ? La figure 3.5 donne un exemple d'exécution de l'algorithme LexBFS* sur le graphe $\overline{P_6}$ où l'ordre ainsi calculé sur le graphe P_6 n'est pas un ordre de contraction amical : la coloration obtenue par l'algorithme Color appliqué à cet ordre n'est pas optimale (voir figure 3.6).

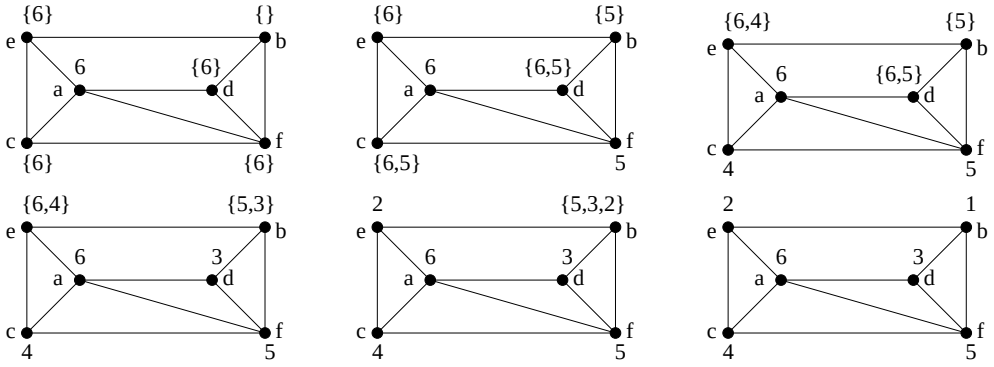
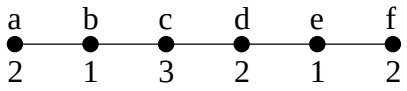


FIG. 3.5 – Exemple d'exécution de l'algorithme LexBFS*



Ordre obtenu par $\overline{\text{LexBFS*}}$ (voir figure 3.5) : b $<$ e $<$ d $<$ c $<$ f $<$ a

FIG. 3.6 – Exemple d'exécution de l'algorithme $\overline{\text{LexBFS*}}$+Color

Appliquer seulement l'algorithme OrdreCosine pour colorier les graphes dans $\mathcal{B}$ ne fonctionne pas non plus puisque la coloration obtenue par l'algorithme Cosine n'est pas optimale sur le graphe $\overline{P_6}$ (voir figure 1.9). Le premier sommet colorié dans ce contre-exemple est justement le toit d'une maison.

L'algorithme LexColor est un algorithme plus rapide que Cosine permettant de colorier les graphes de Meyniel. De la même manière que nous avons modifié Cosine, il est possible de modifier LexBFS en départageant les cas d'égalité grâce à un ordre σ donné en entrée. L'exemple de la figure 1.10 montre que, même lorsque σ est un ordre $\overline{P_3}$-simplicial, cette méthode ne donne pas toujours une coloration optimale de $\overline{P_6}$. L'ordre dans lequel sont coloriés les sommets de la figure 1.10 est un ordre $\overline{P_3}$-simplicial. Le fait de donner cet ordre à l'algorithme pour départager les cas d'égalités ne modifie pas son exécution.

Hayward a prouvé que tout graphe dans $\mathcal{B}$ possède un ordre parfait [40]. Nous avons montré que l'algorithme $\overline{\text{LexBFS*}}$+Color permettait de trouver un ordre de contraction amical pour un graphe dans $\mathcal{B}$. Donc l'algorithme Color appliqué à cet ordre produit une coloration optimale. Est-ce que cette propriété est vraie pour tout sous-graphe induit ? Le graphe de la figure 3.8 donne un exemple de graphe tel que l'algorithme $\overline{\text{LexBFS*}}$+Color ne construit pas un ordre parfait. Les sommets b, c, d, e forment un P_4 que l'algorithme Color colorie avec 3 couleurs au lieu de 2.

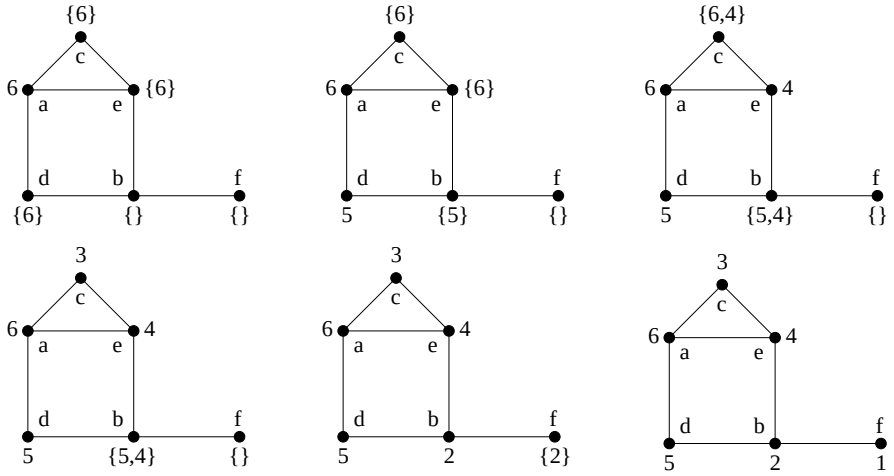
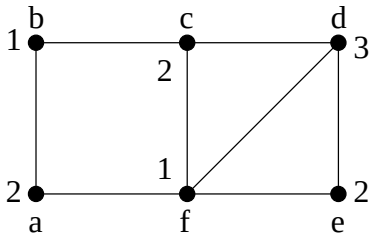


FIG. 3.7 – Exemple d'exécution de l'algorithme LexBFS*



Ordre obtenu par $\overline{\text{LexBFS*}}$ (voir figure 3.7) : f < b < c < e < d < a

FIG. 3.8 – Exemple d'exécution de l'algorithme $\overline{\text{LexBFS*}}$+Color

Les preuves des théorèmes 3.7 et 3.19 redémontrent que les graphes dans $\mathcal{B}$ sont parfaitement ordre-contractiles. Il est donc possible d'appliquer les résultats du chapitre 2. L'algorithme $\overline{\text{LexBFS*}}$+OrdreCosine* peut être étendu par l'algorithme Clique, de complexité $\mathcal{O}(n+m)$, pour trouver une clique de taille maximum dans un graphe dans $\mathcal{B}$. Pour

un graphe dans $\mathcal{B}$, l'ensemble des sommets coloriés 1 par l'algorithme LexBFS*+OrdreCosine* est un stable fort. L'algorithme Pondéré permet alors de résoudre les versions pondérées du problème de la coloration et de la clique maximum, en temps $\mathcal{O}(n^2m)$, pour un graphe dans $\mathcal{B}$.

Tous ces algorithmes sont *robustes* [73] dans le sens où le graphe en entrée peut être un graphe G quelconque, et si G n'est pas dans $\mathcal{B}$ et que les objets obtenus par ces algorithmes ne sont pas optimaux, il est possible de détecter la non-optimalité. Pour cela, il suffit de vérifier en temps $\mathcal{O}(n^3)$ que l'ordre de contraction obtenu par l'algorithme LexBFS*+OrdreCosine* est bien un ordre P_4-libre. Pour obtenir un algorithme de coloration robuste légèrement plus rapide, il suffit d'exécuter LexBFS*+OrdreCosine*+Clique et de vérifier, en temps linéaire, que l'ensemble Q produit par l'algorithme Clique est bien une clique. Si Q est une clique, la coloration est optimale puisqu'elle utilise ℓ couleurs et que Q est de taille ℓ. Si Q n'est pas une clique, le graphe en entrée n'est pas dans $\mathcal{B}$. Cela donne un algorithme de complexité $\mathcal{O}(nm)$ qui permet de colorier optimalement les graphes de la classe $\mathcal{B}$ de manière robuste.

Dans le cas où les objets trouvés par ces algorithmes ne sont pas optimaux et que ces algorithmes retournent "le graphe n'est pas dans $\mathcal{B}$", il serait appréciable qu'ils fournissent un certificat vérifiable en temps polynomial montrant que le graphe n'est pas dans $\mathcal{B}$. La preuve du fait que ces algorithmes trouvent un objet optimal sur la classe de graphes considérée constitue un tel certificat. Ces preuves pourraient être transformées en algorithmes polynomiaux permettant d'exhiber effectivement un trou impair, un antitrou ou un taureau lorsque l'objet trouvé n'est pas optimal. Mais la technicité actuelle des preuves laissent penser que ce travail est un peu trop fastidieux et que la complexité des algorithmes augmenterait significativement.

Nous répondons à ce type de problématique dans le chapitre suivant à propos des graphes de Meyniel. Nous montrons que, pour ces graphes, le fait d'exiger un sous-graphe exclu, lorsque la coloration obtenue par l'algorithme n'est pas optimale, n'augmente pas la complexité.

Chapitre 4

Graphes de Meyniel

Une *obstruction de Meyniel* est un trou impair ou une maison. Dans ce chapitre nous donnons un algorithme de complexité $\mathcal{O}(n^2)$ qui trouve, pour tout graphe, soit une clique et une coloration de même taille soit une obstruction de Meyniel. Nous donnons aussi un algorithme de complexité $\mathcal{O}(n^3)$ qui trouve, pour tout graphe, soit un ordre de contraction P_4-libre soit une obstruction de Meyniel.

4.1 Robustesse

Le travail présenté dans ce chapitre est motivé par le problème de trouver un algorithme de coloration robuste des graphes parfaits. Dans [10], Cameron et Edmonds proposent de chercher un algorithme combinatoire qui, pour tout graphe G, trouve soit une clique et une coloration de même taille soit une obstruction, combinatoire et reconnaissable en temps polynomial, au fait que G soit parfait. Un exemple simple d'obstruction à la perfection est l'existence d'un trou impair ou d'un antitrou impair.

Ici, nous présentons un algorithme polynomial qui répond à cette problématique sur la classe des graphes de Meyniel. Rappelons qu'un graphe est de Meyniel si tout cycle impair de longueur supérieure ou égale à cinq contient au moins deux cordes. Meyniel [68, 69] a prouvé que de tels graphes sont parfaits. Ces graphes ont aussi été étudiés par Markosyan et Karapetyan [67] qui ont indépendamment prouvé leur perfection. Ravindra [74] a montré qu'ils sont fortement parfaits. Les graphes cordaux, les graphes bipartis, ainsi que, plus généralement, les graphes i-triangulés et les graphes de parité (voir [9]) sont des exemples de graphes de Meyniel. Burlet et Fonlupt [9] ont proposé un algorithme polynomial de reconnaissance des graphes de Meyniel. Plus tard Roussel et Rusu [79] ont trouvé un autre algorithme plus rapide, de

complexité $\mathcal{O}(m(m+n))$.

En ce qui concerne le problème de la coloration, Hoàng [49] a donné un algorithme de complexité $\mathcal{O}(n^8)$ permettant de colorier optimalement les graphes de Meyniel. Cet algorithme utilise la décomposition appelée *amalgame* de Burlet et Fonlupt [9]. Puis, la notion de *paire d'amis* [28] a permis à Hertz [45] de prouver que l'algorithme COSINE de complexité $\mathcal{O}(nm)$ donne une coloration optimale. Ensuite, Roussel et Rusu [81] ont défini l'algorithme de coloration LEXCOLOR de complexité $\mathcal{O}(n^2)$. Cet algorithme est actuellement l'algorithme le plus rapide permettant de colorier optimalement les graphes de Meyniel.

Les preuves de Hertz [45] et Roussel et Rusu [81] montrent en fait que l'ordre dans lequel sont coloriés les sommets permet d'obtenir un ordre de contraction amical. L'ordre de coloration des sommets est exactement un ordre de contraction amical pour l'algorithme COSINE. Pour LEXBFS, il suffit de rassembler les sommets par classe de couleur. Comme dans le chapitre précédent, il est alors possible d'utiliser les algorithmes CLIQUE et PONDÉRÉ pour obtenir, de manière robuste [73], pour tout graphe de Meyniel, une coloration et une clique de même taille (éventuellement pondérées). Dans le cas où l'algorithme retourne "le graphe n'est pas de Meyniel", l'idéal serait qu'il retourne aussi un sous-graphe interdit dans la classe des graphes de Meyniel.

4.2 Obstruction

Il est facile de voir qu'un graphe est de Meyniel si et seulement s'il ne contient pas d'obstruction de Meyniel. Le théorème de Meyniel [68], Markosyan et Karapetyan [67] peut être reformulé de la manière suivante :

Théorème 4.1 ([68],[67]) *Pour tout graphe G, soit G contient une clique et une coloration de même taille soit G contient une obstruction de Meyniel.*

Nous donnons un algorithme polynomial qui trouve, pour tout graphe, une instance dont le théorème de Meyniel-Markosyan-Karapetyan affirme l'existence. Cet algorithme est basé sur l'algorithme LEXCOLOR de Roussel et Rusu [81] et fonctionne en temps $\mathcal{O}(n^2)$.

Le théorème de Ravindra [74] peut être reformulé de la manière suivante :

Théorème 4.2 ([74]) *Pour tout graphe G, soit G contient un stable fort, soit G contient une obstruction de Meyniel.*

La preuve de Ravindra est une description informelle d'un algorithme qui trouve, pour tout graphe, une instance dont le théorème affirme l'existence. Hoàng [49] a renforcé ce théorème de la manière suivante :

Théorème 4.3 ([49]) *Pour tout graphe G et tout sommet v de G, soit G contient un stable fort contenant v, soit G contient une obstruction de Meyniel.*

Hoang [49] propose un algorithme de complexité $\mathcal{O}(n^7)$ qui trouve, pour tout sommet d'un graphe de Meyniel, un stable fort contenant ce sommet. Un inconvénient des théorèmes de Ravindra et Hoàng est que l'on ne sait pas vérifier en temps polynomial qu'un ensemble de sommets forme un stable fort.

Cameron et Edmonds [11] ont défini la notion de *bon stable* comme étant un stable maximal dont les sommets peuvent être ordonnés de telle sorte qu'il n'y ait pas de P_4 induit entre n'importe quel sommet x de S et le sommet obtenu en contractant tous les sommets de S qui précèdent x. Un bon stable est un cas particulier de stable fort, vérifiable en temps polynomial. C'est la même notion qui apparaît dans la définition d'un ordre de contraction P_4-libre. Nous avons montré au chapitre 2 que l'ensemble des sommets de couleur 1 d'une coloration P_4-libre est un stable fort, c'est aussi un bon stable. Cameron et Edmonds [11] ont reformulé le théorème de Hoàng de la manière suivante :

Théorème 4.4 ([11]) *Pour tout graphe G et tout sommet v de G, soit G contient un bon stable contenant v, soit G contient une obstruction de Meyniel.*

Hertz [45], puis Roussel et Rusu [81] ont montré le théorème suivant qui renforce le théorème de Cameron et Edmonds :

Théorème 4.5 ([45],[81]) *Pour tout graphe G et tout sommet v de G, soit G contient un ordre de contraction amical dont v est le premier sommet, soit G contient une obstruction de Meyniel.*

Pour un graphe quelconque, nous ne savons pas vérifier qu'un ordre de contraction est amical. Par contre, nous savons vérifier qu'un ordre de contraction est P_4-libre. Une version plus faible du théorème 4.5, mais plus forte que le théorème 4.4 est la suivante :

Théorème 4.6 *Pour tout graphe G et tout sommet v de G, soit G contient un ordre de contraction P_4-libre dont v est le premier sommet, soit G contient une obstruction de Meyniel.*

Nous donnons un algorithme de complexité $\mathcal{O}(n^3)$ qui trouve, pour tout graphe G et tout sommet v de G, soit un ordre de contraction P_4-libre dont

v est le premier sommet, soit une obstruction de Meyniel. Il est ensuite possible d'utiliser l'algorithme PONDÉRÉ, pour en déduire un algorithme de complexité $\mathcal{O}(n^4)$ qui trouve, pour tout graphe et toute fonction de poids, soit une coloration pondérée et une clique pondérée de même taille, soit une obstruction de Meyniel.

4.3 Méthode

Soit G un graphe quelconque (pas nécessairement de Meyniel) sur lequel l'algorithme LEXCOLOR est appliqué. Soit ℓ le nombre total de couleurs utilisées par l'algorithme. L'algorithme CLIQUE est ensuite appliqué à G. A chaque étape de cet algorithme, nous vérifions que le sommet x de couleur c qui est sélectionné satisfait $q(x) = \ell - c$. Si cela est vérifié au cours de toute l'exécution de l'algorithme, alors l'ensemble Q final est une clique de taille l et la coloration obtenue est optimale. Sinon, l'algorithme CLIQUE s'arrête la première fois que l'égalité n'est pas vérifiée et enregistre la couleur courante c et la clique courante Q. Il n'y a alors pas de sommet colorié c qui est adjacent à tout Q. Montrons maintenant comment trouver une obstruction de Meyniel dans G.

Soit k_c le nombre de sommets coloriés c, et pour $i = 1, \ldots, k_c$ soit x_i le i-ème sommet colorié c. Soit G^* le sous-graphe de G obtenu en retirant les sommets de couleur $< c$. Soit G_i^* le graphe obtenu à partir de G^* en contractant $x_c^1, \ldots, x_c^i$ en w_c^i.

L'algorithme fonctionne de la manière suivante. Aucun sommet colorié c n'étant adjacent à tout Q, l'algorithme trouve un chemin impair R dans G_i^* avec une certaine propriété (pour une valeur de i que nous préciserons plus tard), ainsi qu'un sommet z adjacent aux deux extrémités de R. Si R n'est composé que de sommets de G^* et a au plus une corde, et si z n'est adjacent à aucun sommet intérieur de R, alors $R \cup \{z\}$ contient une obstruction. Cependant, ces conditions ne sont pas toujours vérifiées, et nous définissons deux procédures pour remédier à ces difficultés. La première partie de l'algorithme consiste à utiliser une procédure BOUCLE, appelée itérativement. Elle trouve un chemin R et un sommet z adjacent aux deux extrémités de R, où z et les sommets de R sont des sommets de G^* (et donc de G). Le sommet z peut être adjacent aux sommets intérieurs de R. Le chemin R a au plus une corde et, si elle existe, elle est courte. Dans la seconde partie de l'algorithme, BOUCLE appelle la procédure TROUVEROBSTRUCTION sur R et z pour trouver une obstruction de Meyniel. Les variantes $1, 2, 3, 4$ de TROUVEROBSTRUCTION correspondent aux différentes positions possibles de la corde de R. Dans la partie suivante, ces procédures sont décrites plus formellement.

4.4 Algorithme

INITIALISATION

Pour tout sommet v de couleur $> c$ de G^*, soit $n(v)$ le plus petit entier i tel que x_c^i est adjacent à v. L'entier $n(v)$ existe, car v a reçu une couleur strictement plus grande que c. Clairement, la fonction n peut être calculée en temps linéaire. Soit $i = \max\{n(v) \mid v \in Q\}$. Nous avons $i > 1$ car x_c^1 n'est pas adjacent à tout Q. Soit v_1 un sommet de Q non-adjacent à x_c^i. Le sommet v_1 existe, car x_c^i n'est pas adjacent à tout Q. Notons que v_1 est adjacent à w_c^{i-1} dans G_{i-1}^*. Soit v_2 un sommet de Q tel que $n(v_2) = i$. Le sommet v_2 est adjacent à x_c^i, mais pas à w_c^{i-1} dans G_{i-1}^*. Alors $R = w_c^{i-1}$-v_1-v_2-x_c^i est un chemin impair sans corde de G_{i-1}^*. Soit $P = v_1$-v_2-x_c^i. Appeler $Boucle(P, i, \emptyset)$.

BOUCLE(P, i, e)

[où $i > 1$, $P = v_1$-$\cdots$-v_p avec $v_p = x_c^i$ et w_c^{i-1}-P est un chemin impair de G_{i-1}^* avec au plus une corde, et si cette corde existe, elle est du type $v_{t-1}v_{t+1}$ avec $1 < t < p-1$, et où e est un ensemble contenant l'éventuelle corde.]

Affirmation : Il existe un sommet z, colorié avant x_c^i avec une couleur $> c$, adjacent à x_c^i et vérifiant la propriété suivante. Si v_1v_3 est la corde de P, z n'est pas adjacent à au moins un sommet de v_1, v_3. Si v_1v_3 n'est pas la corde de P, alors z n'est pas adjacent à au moins un sommet de v_1, v_2.

Preuve. Soient $R = w_c^{i-1}$-P et $v_0 = w_c^{i-1}$. Considérons l'étape où l'algorithme choisit x_c^i en vue de le colorier. Soit U l'ensemble des sommets de G_{i-1}^* qui sont déjà coloriés à ce moment. Tout sommet de G_{i-1}^* aura une couleur de $\{c, c+1, \ldots, \ell\}$ à la fin de l'algorithme. Donc, si $c \geq 2$, tout sommet v de U vérifie $\forall\ c' < c$, $label_v(c') \neq 0$. Pour tout $X \subseteq U$, soit $couleur(X)$ l'ensemble des couleurs des sommets de X.

Soit $T = N(x_c^i) \cap U$. Tout sommet de T a une couleur $\geq c+1$, donc est adjacent à au moins un sommet colorié c dans G et donc à w_c^{i-1} dans G_{i-1}^*. Définissons le sommet v_r de R de la manière suivante : si v_1v_3 est la corde de R, alors $r = 3$, sinon $r = 2$. Remarquons que v_r n'est pas adjacent à v_0 et $v_r \neq x_c^i$ à cause des positions possibles de la corde et de la parité de R.

Supposons que l'affirmation est fausse : alors tout sommet de T est adjacent à v_1 et v_r.

Supposons que v_1 n'est pas encore colorié (c'est à dire $v_1 \notin U$). Puisque tout sommet de T est adjacent à v_1, nous avons $\forall\ c' > c$, $label_{v_1}(c') \geq label_{x_c^i}(c')$. Puisque v_1 est adjacent à w_c^{i-1}, nous avons $label_{v_1}(c) > 0$. Puisque x_c^i est colorié c, nous avons $label_{x_c^i}(c) = 0$. Donc $label_{v_1} >_{Lex} label_{x_c^i}$, ce qui contredit le fait que l'algorithme s'apprête à colorier x_c^i. Donc v_1 est déjà

colorié et la couleur de v_1 n'est pas dans $\{1, \ldots, c\} \cup couleur(T)$.

Supposons que v_r n'est pas encore colorié. Puisque tout sommet de T est adjacent à v_r, nous avons $\forall\ c' > c$, $label_{v_r}(c') \geq label_{x_c^i}(c')$. Puisque v_r est adjacent à v_1, nous avons $label_{v_r}(couleur(v_1)) > 0$. Puisque $couleur(v_1) \notin \{1, \ldots, c\} \cup couleur(T)$, nous avons $label_{x_c^i}(couleur(v_1)) = 0$. Donc $label_{v_r} >_{Lex} label_{x_c^i}$, encore une contradiction. Donc v_r est déjà colorié. Cependant, v_r n'est pas adjacent à w_c^{i-1}, donc, lorsque v_r a été colorié, c était sa plus petite couleur disponible. Cela contredit la définition de w_c^{i-1} et x_c^i.

□

Soit z un sommet qui satisfait l'affirmation ci-dessus. Le sommet z est adjacent à w_c^{i-1} dans G_{i-1}^*, car z a été colorié avant x_c^i et a reçu une couleur strictement plus grande que c. (Cela prend un temps $deg(x_c^i)$ de trouver un tel sommet z.)

Soit $j = max(n(z), n(v_1))$. Nous avons $j < i$, car z et v_1 sont adjacents à w_c^{i-1}.

Supposons $n(z) = n(v_1)$. Alors x_c^j est adjacent à v_1 et z. Si v_1v_3 est la corde de P, appeler TROUVEROBSTRUCTION2$(z, x_c^j\text{-}P)$. Si v_1v_3 n'est pas la corde de P et z n'est pas adjacent à v_1, appeler TROUVEROBSTRUCTION3$(z, x_c^j\text{-}P, e)$. Si v_1v_3 n'est pas la corde de P et z est adjacent à v_1 (et alors n'est pas adjacent à v_2), appeler TROUVEROBSTRUCTION4$(z, x_c^j\text{-}P, e)$.

Maintenant, supposons $n(z) \neq n(v_1)$. Alors $j > 1$, l'un de w_c^{j-1}, x_c^j est adjacent à v_1 et n'est pas adjacent à z, et l'autre n'est pas adjacent à v_1 et est adjacent à z. Soit k le plus petit entier tel que z est adjacent à v_k. Un tel k existe puisque z est adjacent à v_p. (Cela prend un temps $deg(z)$ pour calculer k.)

Supposons que k est impair. Si $n(z) = j$, alors soit $P' = v_1\text{-}\cdots\text{-}v_k\text{-}z\text{-}x_c^j$; sinon soit $P' = z\text{-}v_k\text{-}\cdots\text{-}v_1\text{-}x_c^j$. Si e est une corde de P', alors appeler BOUCLE(P', j, e) ; sinon appeler BOUCLE$(P', j, \emptyset)$.

Maintenant, supposons que k est pair. Alors, $k < p$ puisque p est impair. Considérons les cas suivants :

Cas 1 : P a une corde $v_{t-1}v_{t+1}$ avec $t < k$. Si $n(z) = j$, alors soit $P' = v_1\text{-}\cdots\text{-}v_{t-1}\text{-}v_{t+1}\text{-}\cdots\text{-}v_k\text{-}z\text{-}x_c^j$; sinon soit $P' = z\text{-}v_k\text{-}\cdots\text{-}v_{t+1}\text{-}v_{t-1}\text{-}\cdots\text{-}v_1\text{-}x_c^j$. Appeler BOUCLE$(P', j, \emptyset)$.

Cas 2 : P a une corde $v_{k-1}v_{k+1}$. Lorsque z est adjacent à la fois à v_{k+1} et v_{k+2}, si $n(z) = j$, alors soit $P' = v_1\text{-}\cdots\text{-}v_{k-1}\text{-}v_{k+1}\text{-}v_{k+2}\text{-}z\text{-}x_c^j$; sinon soit $P' = z\text{-}v_{k+2}\text{-}v_{k+1}\text{-}v_{k-1}\text{-}\cdots\text{-}v_1\text{-}x_c^j$; appeler BOUCLE(P', j, zv_{k+1}). Lorsque z n'est pas adjacent à v_{k+1}, appeler TROUVEROBSTRUCTION3$(z, v_k\text{-}\cdots\text{-}v_p, \emptyset)$. Lorsque z est adjacent à v_{k+1} et n'est pas adjacent à v_{k+2}, appeler TROUVEROBSTRUCTION4$(z, v_k\text{-}\cdots\text{-}v_p, \emptyset)$.

Cas 3 : P a une corde v_kv_{k+2}. Lorsque z est adjacent à v_{k+1}, si $n(z) = j$,

alors soit $P' = v_1\text{-}\cdots\text{-}v_k\text{-}v_{k+1}\text{-}z\text{-}x_c^j$; sinon soit $P' = z\text{-}v_{k+1}\text{-}v_k\text{-}\cdots\text{-}v_1\text{-}x_c^j$; appeler BOUCLE(P', j, zv_k). Lorsque z n'est pas adjacent à v_{k+1} et est adjacent à v_{k+2}, si $n(z) = j$, alors soit $P' = v_1\text{-}\cdots\text{-}v_k\text{-}v_{k+2}\text{-}z\text{-}x_c^j$; sinon soit $P' = z\text{-}v_{k+2}\text{-}v_k\text{-}\cdots\text{-}v_1\text{-}x_c^j$; appeler BOUCLE(P', j, zv_k). Lorsque z n'est pas adjacent à v_{k+1} et v_{k+2}, appeler TROUVEROBSTRUCTION1$(z, v_k\text{-}\cdots\text{-}v_p)$.

Cas 4 : P n'a pas de corde $v_{t-1}v_{t+1}$ avec $t \leq k+1$. Lorsque z est adjacent à v_{k+1}, si $n(z) = j$, alors soit $P' = v_1\text{-}\cdots\text{-}v_k\text{-}v_{k+1}\text{-}z\text{-}x_c^j$; sinon soit $P' = z\text{-}v_{k+1}\text{-}v_k\text{-}\cdots\text{-}v_1\text{-}x_c^j$; appeler BOUCLE(P', j, zv_k). Lorsque z n'est pas adjacent à v_{k+1}, appeler TROUVEROBSTRUCTION3$(z,\ v_k\text{-}\cdots\text{-}v_p, e)$.

TROUVEROBSTRUCTION1(z, P)

[où $P = v_0\text{-}\cdots\text{-}v_p$ est un chemin impair avec une seule corde v_0v_2, z est adjacent à v_0, v_p et z n'est adjacent ni à v_1 ni à v_2]

Soit r le plus petit entier > 0 tel que z est adjacent à v_r. Cet entier r existe, car z est adjacent à v_p. Nous avons $r \geq 3$, car z n'est adjacent ni à v_1 ni à v_2. Lorsque r est impair, retourner $z, v_0, \ldots, v_r$, qui induit une maison. Lorsque r est pair, nous avons $r < p$ et $r \geq 4$; retourner $z, v_0, v_2, \ldots, v_r$, qui induit un trou impair.

TROUVEROBSTRUCTION2(z, P)

[où $P = v_0\text{-}\cdots\text{-}v_p$ est un chemin impair $p > 3$ avec une seule corde v_1v_3, z est adjacent à v_0, v_p et z n'est pas adjacent à l'un de v_1, v_3]

Cas 1 : z n'est adjacent ni à v_1 ni à v_2. Soit r le plus petit entier > 0 tel que z est adjacent à v_r ; nous avons $r \geq 3$. Lorsque r est impair, retourner $z, v_0, \ldots, v_r$, qui induit une maison. Lorsque r est pair, nous avons $r \geq 4$; retourner $z, v_0, v_1, v_3, \ldots, v_r$, qui induit un trou impair.

Cas 2 : z n'est pas adjacent à v_1, est adjacent à v_2, n'est pas adjacent à v_3. Appeler TROUVEROBSTRUCTION3$(z, v_2\text{-}\cdots\text{-}v_p, \emptyset)$.

Cas 3 : z n'est pas adjacent à v_1, est adjacent à v_2, est adjacent à v_3, n'est pas adjacent à v_4. Appeler TROUVEROBSTRUCTION4$(z, v_2\text{-}\cdots\text{-}v_p, \emptyset)$,

Cas 4 : z n'est pas adjacent à v_1, est adjacent à v_2, est adjacent à v_3, est adjacent à v_4. Retourner z, v_0, v_1, v_3, v_4, qui induit une maison.

Cas 5 : z est adjacent à v_1. Le sommet z n'est pas adjacent à v_3, car z n'est pas adjacent à l'un de v_1 ou v_3 ; appeler TROUVEROBSTRUCTION3$(z, v_1\text{-}v_3\text{-}\cdots\text{-}v_p, \emptyset)$.

TROUVEROBSTRUCTION3(z, P, e)

[où $P = v_0\text{-}\cdots\text{-}v_p$ est un chemin impair $p \geq 3$ avec au plus une corde, et, si

cette corde existe, elle est du type $v_{t-1}v_{t+1}$ avec $1 < t < p-1$, z est adjacent à v_0, v_p et z n'est pas adjacent à v_1, et où e est un ensemble contenant l'éventuelle corde]

Soit r le plus petit entier > 0 tel que z est adjacent à v_r ($r \geq 2$). Lorsque r est impair, retourner $z, v_0, \ldots, v_r$, qui induit un trou impair ou une maison. Lorsque r est pair, considérons les différents cas suivants :

Cas 1 : P a une corde $v_{t-1}v_{t+1}$ avec $t < r$. Retourner $z, v_0, \cdots, v_{t-1}, v_{t+1}, \cdots, v_r$, qui induit un trou impair.

Cas 2 : P a une corde $v_{r-1}v_{r+1}$. Lorsque z est adjacent à la fois à v_{r+1} et v_{r+2}, retourner z,$v_0, \cdots, v_{r-1}$,v_{r+1},v_{r+2}, qui induit une maison. Lorsque z n'est pas adjacent à v_{r+1}, appeler TROUVEROBSTRUCTION3(z,v_r-$\cdots$-v_p,$\emptyset$). Lorsque z est adjacent à v_{r+1} et z n'est pas adjacent à v_{r+2}, appeler TROUVEROBSTRUCTION4(z,v_r-$\cdots$-v_p,$\emptyset$).

Cas 3 : P a une corde v_rv_{r+2}. Lorsque z est adjacent à v_{r+1}, retourner $z, v_0, \cdots, v_{r+1}$, qui induit une maison. Lorsque z n'est pas adjacent à v_{r+1} et z est adjacent à v_{r+2}, retourner $z, v_0, \cdots, v_r, v_{r+2}$, qui induit une maison. Lorsque z n'est adjacent ni à v_{r+1} ni à v_{r+2}, appeler TROUVEROBSTRUCTION1(z, v_r-$\cdots$-v_p).

Cas 4 : P n'a pas de corde $v_{t-1}v_{t+1}$ avec $t \leq r+1$. Lorsque z est adjacent à v_{r+1}, retourner $z, v_0, \cdots, v_{r+1}$, qui induit une maison. Lorsque z n'est pas adjacent à v_{r+1}, appeler TROUVEROBSTRUCTION3(z, v_r-$\cdots$-v_p, e).

TROUVEROBSTRUCTION4(z, P, e)

[où $P = v_0$-$\cdots$-v_p est un chemin impair $p \geq 3$ avec au plus une corde, et si cette corde existe, elle est du type $v_{t-1}v_{t+1}$ avec $2 < t < p-1$, z est adjacent à v_0, v_1, v_p et z n'est pas adjacent à v_2, et où e est un ensemble contenant l'éventuelle corde]

Soit r le plus petit entier > 1 tel que z est adjacent à v_r ; nous avons $r \geq 3$.

Cas 1 : P a une corde $v_{t-1}v_{t+1}$ avec $2 < t < r$. Lorsque r est impair, retourner $z, v_1, \ldots, v_{t-1}, v_{t+1}, \ldots, v_r$, qui induit un trou impair. Lorsque r est pair, retourner $z, v_1, \ldots, v_r$, qui induit une maison.

Cas 2 : P n'a pas de corde $v_{t-1}v_{t+1}$ avec $2 < t < r$. Lorsque r est impair, retourner $z, v_0, \ldots, v_r$, qui induit une maison. Lorsque r est pair, retourner $z, v_1, \ldots, v_r$ qui induit un trou impair.

4.5 Complexité

Cet algorithme peut être implémenté en temps $\mathcal{O}(n+m)$. A chaque passage dans la boucle BOUCLE(P, i), i est décrémenté d'au moins 1, donc il y a au plus k_c passages dans cette boucle. Chaque passage prend un temps $\mathcal{O}(deg(x_c^i) + deg(z))$, et x_c^i est différent à chaque passage puisque i diminue. Le sommet z est aussi différent à chaque passage, car si $n(z) = j$, alors z n'est pas adjacent à w_{j-1}^c et alors z ne sera adjacent à aucun futur sommet x_c^i, alors que si $n(z) \neq j$, le sommet z devient le second sommet sur le chemin P avant que dans une boucle future, nous ayons $n(z) = j$.

Chaque appel à une variante de TROUVEROBSTRUCTION a lieu avec le même sommet z, donc nous pouvons calculer le tableau d'adjacence de z, la première fois que TROUVEROBSTRUCTION est appelé. Alors, calculer la valeur de r prend un temps $\mathcal{O}(r)$ et le reste de chaque TROUVEROBSTRUCTION prend un temps constant. A chaque passage dans TROUVEROBSTRUCTION, la longueur de P décroît d'au moins r.

L'algorithme LEXCOLOR et l'algorithme CLIQUE fonctionnent respectivement en temps $\mathcal{O}(n^2)$ et $\mathcal{O}(n+m)$. Le temps total pour trouver, dans tout graphe, une clique et une coloration de même taille, ou une obstruction de Meyniel, est donc $\mathcal{O}(n^2)$.

Nous pouvons maintenant en déduire un algorithme de complexité $\mathcal{O}(n^3)$ qui trouve, pour tout graphe G et tout sommet v de G, soit un ordre de contraction P_4-libre dont v est le premier sommet, soit une obstruction de Meyniel.

Appliquons l'algorithme LEXCOLOR sur un graphe G, en commençant par colorier v. Supposons que l'algorithme utilise l couleurs. Pour $1 \leq c \leq l$, soit k_c le nombre de sommets coloriés c, et pour $1 \leq i \leq k_c$ soit x_c^i le i-ème sommet colorié c. Nous pouvons vérifier en temps $\mathcal{O}(n^3)$ que l'ordre de contraction $\{x_1^1, \ldots, x_{k_l}^{kl}\}$ est P_4-libre. Si ce n'est pas un ordre de contraction P_4-libre, alors la procédure de vérification retourne c et i tels que, $1 \leq c \leq l$, $1 \leq i \leq k_c$ et il existe un P_4 induit w_c^{i-1}-v_1-v_2-x_c^i (où w_c^{i-1} est le sommet issu de la contraction de $x_c^1 \ldots x_c^{i-1}$). Soit $P = v_1$-v_2-x_c^i. L'appel à la procédure BOUCLE$(P, i, \emptyset)$ retourne une obstruction de Meyniel de G.

Il est ensuite possible d'utiliser l'algorithme PONDÉRÉ, pour en déduire un algorithme de complexité $\mathcal{O}(n^4)$ qui trouve, pour tout graphe et toute fonction de poids, soit une coloration pondérée et une clique pondérée de même taille, soit une obstruction de Meyniel.

4.6 Commentaires

Les algorithmes présentés ici ne sont pas des algorithmes de reconnaissance des graphes de Meyniel. Il peut arriver que le graphe en entrée ne soit pas de Meyniel et que la sortie soit optimale.

La figure 4.1 montre un exemple de graphe qui n'est pas de Meyniel et sur lequel l'algorithme LexColor suivi de l'algorithme Clique peut donner une coloration et une clique de même taille, alors que l'ordre de contraction $a, f, h, b, d, i, g, c, e$ n'est pas P_4-libre. Une fois que a et f sont contractés en af, il y a un P_4 af-b-c-h.

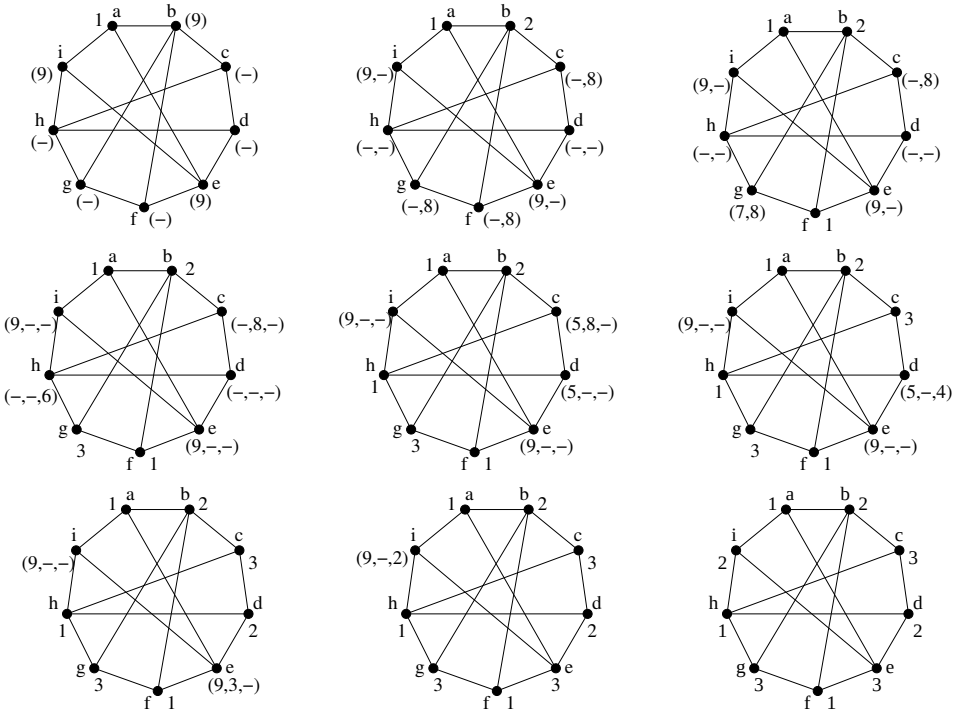


FIG. 4.1 – Exemple d'exécution de l'algorithme LexColor

L'algorithme le plus rapide pour reconnaître les graphes de Meyniel est dû à Roussel et Rusu [79]. Sa complexité est $\mathcal{O}(m(m+n))$. Il apparaît donc qu'il est plus facile de colorier les graphes de Meyniel de manière robuste que de les reconnaître. Peut-être en est-il de même pour les graphes parfaits ? Pour le moment, la reconnaissance des graphes parfaits se fait en temps $\mathcal{O}(n^9)$ par un algorithme dû à Chudnovsky, Cornuéjols, Liu, Seymour et Vušković [13].

Chapitre 5

Graphes d'Artémis

Maffray et Trotignon [66] ont démontré le théorème 2.19 qui était une conjecture d'Everett et Reed [28, 75] affirmant que les graphes d'Artémis sont parfaitement contractiles. La preuve de Maffray et Trotignon contient un algorithme qui trouve une coloration optimale de tout graphe d'Artémis en temps $\mathcal{O}(n^4m)$. Nous montrons que cet algorithme peut être simplifié et accéléré. Nous obtenons alors un algorithme de complexité $\mathcal{O}(n^2m)$ permettant de colorier les graphes d'Artémis.

5.1 Méthode

La preuve de [66] consiste à trouver une paire d'amis spéciale et à la contracter. Le lemme suivant assure que le graphe contracté est toujours d'Artémis.

Lemme 5.1 ([28, 66]) *Si G est un graphe d'Artémis et $\{a, b\}$ est une paire d'amis spéciale de G, alors G/ab est un graphe d'Artémis.*

La contraction de paire d'amis spéciale peut être itérée jusqu'à ce que le graphe soit une union disjointe de cliques. Ces cliques peuvent alors être contractées trivialement en une seule. Lorsque le graphe n'est pas une union disjointe de cliques, la méthode suivante permet de trouver une paire d'amis spéciale.

Un sommet $V \setminus X$ est dit *X-complet* s'il voit tous les sommets de X ; et $C(X)$ dénote l'ensemble des sommets X-complet de $V \setminus X$. Un ensemble non vide $T \subseteq V$ est dit *intéressant* si $G[T]$ est co-connexe et $G[C(T)]$ n'est pas une clique (donc $|C(T)| \geq 2$ puisque l'ensemble vide est considéré comme une clique). Un ensemble intéressant est *maximal* s'il n'est pas strictement inclus dans un autre ensemble intéressant. Un *chemin T-sortant* est un chemin sans

corde dont les extrémités sont dans $C(T)$ et dont les sommets intérieurs sont tous dans $V \setminus (T \cup C(T))$. Un chemin T-sortant P est *minimal* s'il n'y a pas de chemin T-sortant dont les sommets intérieurs sont strictement contenus dans les sommets intérieurs de P. La recherche d'une paire d'amis spéciale considère les trois cas suivants :

- le graphe n'a pas d'ensemble intéressant ;
- un ensemble intéressant T de G n'a pas de chemin T-sortant ;
- un ensemble intéressant T de G a un chemin T-sortant.

Ces trois cas correspondent aux trois lemmes suivants.

Lemme 5.2 ([66]) *Pour tout graphe G, les conditions suivantes sont équivalentes :*

- *G n'a pas d'ensemble intéressant.*
- *Tout sommet de G est simplicial.*
- *G est une union disjointe de cliques.*

De plus, si G n'est pas une union disjointe de cliques, alors tout sommet non-simplicial forme un ensemble intéressant.

Lemme 5.3 ([66]) *Soit G un graphe d'Artémis qui contient un ensemble intéressant et soit T un ensemble intéressant maximal de G. Si T n'a pas de chemin T-sortant, alors toute paire d'amis spéciale du sous-graphe $G[C(T)]$ est une paire d'amis spéciale de G.*

Lorsqu'un ensemble intéressant maximal T a un chemin T-sortant, notons $\alpha z_1 \cdots z_p \beta$ un chemin T-sortant minimal. Les ensembles A et B sont définis de la manière suivante :

$$\begin{aligned} A &= \{v \in C(T) \mid vz_1 \in E, vz_i \notin E \ (i = 2, \ldots, p)\}, \\ B &= \{v \in C(T) \mid vz_p \in E, vz_i \notin E \ (i = 1, \ldots, p-1)\}. \end{aligned}$$

Une relation $<_A$ est définie sur A en posant $u <_A u'$ si et seulement si $u, u' \in A$ et il existe un chemin sans corde impair de u vers un sommet de B tel que u' est le deuxième sommet de ce chemin (u en est le premier sommet). De même, une relation $<_B$ est définie sur B en posant $v <_B v'$ si et seulement si $v, v' \in B$ et il existe un chemin impair sans corde de v vers un sommet de A tel que v' est le deuxième sommet de ce chemin.

Lemme 5.4 ([66]) *Lorsque A, B et $<_A, <_B$ sont définis comme ci-dessus, ils vérifient :*

- *Les ensembles A et B sont des cliques non-vides avec aucune arête entres elles.*

- *Si $P = uu' \cdots v'v$ est un chemin sans corde impair avec $u \in A$ et $v \in B$, alors soit $u' \in A$, soit $v' \in B$.*
- *La relation $<_A$ est un ordre partiel strict sur A. La relation $<_B$ est un ordre partiel strict sur B.*
- *Si a est un sommet maximal pour $<_A$ et b est un sommet maximal pour $<_B$, alors $\{a, b\}$ est une paire d'amis spéciale de G.*

Le lemme suivant est immédiat :

Lemme 5.5 *Soient T un ensemble intéressant maximal dans le graphe G et a, b deux sommets non-adjacents de $C(T)$. Soit $C'(T)$ l'ensemble des sommets T-complets dans G/ab. Si $C'(T)$ n'est pas une clique, alors T est un ensemble intéressant maximal de G/ab.*

Voici comment fonctionne la méthode. Premièrement, un algorithme trouve un ensemble intéressant maximal T dans G. Puis, un deuxième algorithme trouve une paire d'amis spéciale dans $C(T)$, en se basant sur les lemmes 5.3 et 5.4, et la contracte. Ce deuxième algorithme est itéré tant que l'ensemble $C(T)$ n'est pas une clique, ce qui est possible d'après les lemmes 5.1 et 5.5. Lorsque l'ensemble $C(T)$ devient une clique, le premier algorithme est appelé à nouveau pour trouver un nouvel ensemble intéressant maximal. Puisque la contraction d'une paire d'amis réduit le nombre de sommets de 1, il y aura au plus n contractions. Donc la complexité totale est égale à n fois la complexité du problème de trouver une paire d'amis spéciale. Nous verrons qu'une paire d'amis spéciale peut être trouvée en temps $\mathcal{O}(nm)$, donc la complexité totale de l'algorithme de coloration est $\mathcal{O}(n^2m)$.

5.2 Ensemble intéressant maximal

Pour trouver un ensemble intéressant maximal, nous appliquons l'algorithme suivant :

ALGORITHME INTÉRESSANT
ENTRÉE : Un graphe G.
SORTIE : Un ensemble intéressant maximal T de G ou la réponse "G est une union disjointe de cliques".
CALCUL :
Etape 1 : Chercher un sommet non-simplicial t.
- Calculer les composantes connexes de G ;
- si tout sommet est de degré égal à la taille de sa composante moins 1, retourner "G est une union disjointe de cliques" ;

- sinon, soit u un sommet dont le degré est strictement plus petit que la taille de sa composante moins 1. Exécutons un parcours en largeur à partir de u, soit v un sommet à distance 2 de u, et soit t le père de v dans le parcours.

Etape 2 : Construisons T à partir de t.

- Soient $T = \{t\}$, $C = N(t)$, $U = V(G) \setminus (T \cup C)$, $Z = \emptyset$;
- tant qu'il existe $u \in U$:
  - si $N(u) \cap C$ est une clique, déplacer u de U vers Z ;
  - si $N(u) \cap C$ n'est pas une clique, déplacer u de U vers T et déplacer tout sommet de $C \setminus N(u)$ de C vers U ;
- retourner l'ensemble T.

COMPLEXITÉ : $\mathcal{O}(\max(n+m, m(n-k)))$ où k est le nombre de sommets de $C(T)$ de l'ensemble T retourné (si aucun ensemble T n'est retourné, nous considérons que $k = n$ et que la complexité est $\mathcal{O}(n+m)$).

Le lemme suivant est immédiat :

Lemme 5.6 *Soit T un ensemble intéressant dans un graphe G. S'il y a un sommet $u \in V(G) \setminus (T \cup C(T))$ tel que $N(u) \cap C(T)$ n'est pas une clique, alors $T \cup \{u\}$ est un ensemble intéressant. S'il n'y pas de tel sommet, alors T est un ensemble intéressant maximal.*

Lemme 5.7 *L'algorithme* INTÉRESSANT *est correct.*

Preuve. L'étape 1 de l'algorithme est correcte. Au début de l'étape 2, l'ensemble T est intéressant, C est égal à l'ensemble des sommets T-complets et n'est pas une clique. La définition de l'étape 2 implique que ces propriétés restent vraies et le lemme 5.6 assure que, lorsque l'étape 2 se termine, l'ensemble T est un ensemble intéressant maximal. □

Analyse de complexité

L'étape 1 prend un temps $\mathcal{O}(n+m)$. Dans l'étape 2, un sommet peut seulement être déplacé de C vers U ou de U vers Z ou vers T. Donc les ensembles T et Z ne peuvent qu'augmenter et les ensembles $U \cup C$ et C ne peuvent que diminuer. Décider si $N(u) \cap C$ est une clique prend un temps $\mathcal{O}(m)$, et la mise à jour de C prend un temps $\mathcal{O}(\deg(t))$ puisque C peut seulement décroître de sa valeur initiale $N(t)$. Donc, chaque itération de la boucle prend un temps $\mathcal{O}(m)$. Un sommet joue le rôle de u au plus une fois et les k sommets qui sont dans C à la fin de l'algorithme n'ont jamais joué ce rôle. Donc il y a au plus $n-k$ itérations de la boucle.

5.3 Chemin sortant minimal

Etant donnés deux ensembles disjoints $X, Y \subseteq V$ d'un graphe G, nous appelons *parcours en largeur de X vers Y* une variante de l'algorithme BFS telle que les sommets de X sont mis dans la file au début de l'algorithme et les sommets de Y ne sont jamais mis dans la file.

Une fois que nous avons un ensemble intéressant maximal, nous trouvons un chemin sortant minimal, grâce à l'algorithme suivant :

ALGORITHME CHEMINSORTANT
ENTRÉE : Un graphe G et un ensemble intéressant maximal T de G.
SORTIE : Un chemin T-sortant minimal ou la réponse "G n'a pas de chemin T-sortant".
CALCUL :
- Tous les sommets de $V(G) \setminus (T \cup C(T))$ sont non-marqués ;
- tant qu'il existe un sommet non-marqué r dans $V(G) \setminus (T \cup C(T))$:
  - effectuer un parcours en largeur de r vers $C(T)$ dans G, en notant S les sommets rencontrés, M l'ensemble $S \cap C(T)$ et en arrêtant le parcours dès que M n'est plus une clique ; soit F les sommets qui sont encore dans la file lorsque le parcours est arrêté ;
  - si M n'est pas une clique :
    - soit x le dernier sommet ajouté à M et $M_x = M \cap N(x)$;
    - effectuer un parcours en largeur à partir de x dans le sous-graphe $G[S \setminus (F \cup M_x)]$;
    - soit y le premier sommet de $M \setminus M_x$ qui est atteint par ce parcours ;
    - retourner le chemin de ce parcours qui va de x à y ;
  - sinon, marquer les sommets de S ;
- retourner "G n'a pas de chemin T-sortant".

Complexité : $\mathcal{O}(\ell m)$, où ℓ est le nombre de composantes de $G \setminus (T \cup C(T))$.

Le lemme suivant est immédiat :

Lemme 5.8 *Un ensemble intéressant T a un chemin sortant si et seulement s'il existe une composante R de $V(G) \setminus (T \cup C(T))$ telle que $N(R) \cap C(T)$ n'est pas une clique.*

Lemme 5.9 ([66]) *Soit G un graphe d'Artémis qui contient un ensemble intéressant et soit T un ensemble intéressant maximal de G. Alors tout chemin T-sortant est de longueur paire et au moins 4.*

Lemme 5.10 *L'algorithme* CHEMINSORTANT *est correct.*

Preuve. Soit R la composante de $V(G) \setminus (T \cup C(T))$ qui contient r. Le parcours à partir de r peut atteindre potentiellement tous les sommets de R et de $N(R) \cap C(T)$ et mettre ces derniers dans M. Si $N(R) \cap C(T)$ est une clique, le parcours marque tous les sommets de R et recommence avec un nouveau sommet r d'une autre composante connexe de $V(G) \setminus (T \cup C(T))$, si elle existe. Le lemme 5.8 assure que l'algorithme retournera "G n'a pas de chemin T-sortant" si et seulement si G n'a pas de chemin T-sortant. Il reste à prouver que lorsque l'algorithme retourne un chemin, c'est un chemin T-sortant minimal. Considérons cette situation. Pour une composante R de $V(G)\setminus(T\cup C(T))$, l'ensemble $N(R)\cap C(T)$ n'est pas une clique et le parcours depuis un sommet $r \in R$ trouve un sommet x la première fois que M n'est pas une clique. L'ensemble $M \setminus M_x$ n'est pas vide. Les seuls voisins de x dans $S \setminus M$ sont soit son parent dans le parcours, soit des sommets qui sont encore dans la file, sinon x aurait auparavant été ajouté à S. Donc, tout sommet de $S \setminus (F \cup M)$ non-adjacent à x a été parcouru avant le parent de x.

Le parcours à partir de x dans $G[S \setminus (F \cup M_x)]$ atteint tous les sommets de $M \setminus M_x$, donc le sommet y existe bien. Notons $P = x\text{-}z_1\text{-}\cdots\text{-}z_p\text{-}y$ le chemin retourné par l'algorithme, et posons $Z = \{z_1, \ldots, z_p\}$. Pour montrer que P est un chemin T-sortant, supposons au contraire qu'un élément z_i de Z est dans $C(T)$ et soit i le plus petit entier vérifiant cette propriété. $(1 \leq i \leq p)$. Nous avons $i \geq 2$, car $z_1 = v$ est dans R ; mais alors z_i contredit la définition de y. Donc tous les sommets de $z_1, \ldots, z_p$ sont dans R, ce qui signifie que P est un chemin T-sortant. Pour prouver la minimalité de P, supposons au contraire qu'il existe un chemin T-sortant $x'\text{-}z_i\text{-}\cdots\text{-}z_j\text{-}y'$ avec $1 \leq i \leq j \leq p$ et $j - i < p - 1$. D'après le lemme 5.9, $j - i$ est pair et au moins égal à 2, donc $j > 2$. Le sommet y' est dans $M \setminus \{x\}$ puisque z_j a été ajouté dans S avant z_1. Si $i > 1$ alors x' est aussi dans $M \setminus \{x\}$, puisque z_i a été ajouté à S avant z_1 ; mais alors $M \setminus \{x\}$ n'est pas une clique, ce qui contredit la définition de x. Donc $i = 1$. Si x est adjacent à y', alors $x\text{-}z_1\text{-}\cdots\text{-}z_j\text{-}y'\text{-}x$ est un trou impair, une contradiction. Donc x n'est pas adjacent à y', mais alors $x\text{-}z_1\text{-}\cdots\text{-}z_j\text{-}y'$ est un chemin sans corde et y' contredit la définition de y. Donc P est un chemin T-sortant minimal. □

Analyse de complexité

Un parcours à partir d'un sommet r atteint tous les sommets de la composante R de $V(G) \setminus (T \cup C(T))$ qui contient r et tous les sommets de $N(R) \cap C(T)$, et seulement ceux-là. De plus, ces sommets sont parcourus seulement une fois durant le parcours. Le parcours à partir de x atteint les sommets de R une deuxième fois. Donc les sommets de R sont parcourus au

plus deux fois. Pour vérifier si M est une clique, nous utilisons un compteur qui, pour chaque sommet u de $C(T)$, compte le nombre de voisins de u dans M. Lorsqu'un nouveau sommet u est ajouté à M, nous vérifions que le compteur de u est égal à $|M|$ et nous parcourons u pour accroître de 1 le compteur de ses voisins dans $C(T)$. Donc, la complexité pour une composante R est $\mathcal{O}(m)$.

Lorsque l'ensemble T n'a pas de chemin T-sortant, le lemme 5.3 dit que nous devons continuer la recherche récursivement dans le sous-graphe $G[C(T)]$. Nous trouvons donc un ensemble intéressant maximal T_1 de G, puis un intéressant maximal T_2 de $G[C_1]$ (où $C_1 = C(T_1)$), puis un ensemble intéressant maximal T_3 de $G[C_2]$ (où $C_2 = C(T_2) \cap C_1$), ..., jusqu'à trouver un intéressant maximal T_q de $G[C_{q-1}]$ (où $C_{q-1} = C(T_{q-1}) \cap C_{q-2}$) tel que, soit il y a un chemin T_q-sortant dans $G[C_{q-1}]$, soit $G[C(T_q) \cap C_{q-1}]$ est une union disjointe de cliques. Analysons la complexité de cette procédure.

Pour $i = 1, \ldots, q$, soit $n_i = |C_i|$ et $m_i = |E(G[C_i])|$, et posons $n_0 = n$ et $m_0 = m$. La complexité totale pour trouver un intéressant maximal au cours de tous les appels récursifs est donc $\mathcal{O}(\Sigma_{i=1}^{q-1} m_{i-1}(n_{i-1}-n_i)+\max\{n_{q-1}+m_{q-1}, m_{q-1}(n_{q-1}-n_q)\}) = O(\max\{n+m,\, mn)\}$.

Pour $i = 1, \ldots, q$, soit l_i le nombre de composantes connexes de $G[C_{i-1} \setminus (T_i \cup C(T_i))]$. Observons que toutes ces composantes (pour tout $i = 1, \ldots, q$) sont deux à deux disjointes, donc $l_1 + \cdots + l_q \leq n$. La complexité totale pour trouver les chemins sortants au cours de tous les appels récursifs est $\mathcal{O}(\Sigma_{i=1}^{q} l_i m) = O(nm)$.

La complexité totale de cette procédure récursive est donc $\mathcal{O}(nm)$.

5.4 Paire d'amis spéciale

Une fois que nous avons un intéressant maximal et un chemin sortant minimal, nous trouvons une paire d'amis spéciale grâce à l'algorithme suivant :

ALGORITHME PAIREDAMIS
ENTRÉE : Un graphe G, un ensemble intéressant maximal T et un chemin T-sortant minimal x-z_1-$\cdots$-z_p-y.
SORTIE : Une paire d'amis spéciale de G.
CALCUL :
1. Soient $A = (N(z_1) \cap C(T)) \setminus N(y)$ et $B = (N(z_p) \cap C(T)) \setminus N(x)$;
2. effectuer un parcours en largeur de B vers $N(A)$ dans $G \setminus (T \cup A)$ et soit K les sommets de $N(A)$ qui sont atteints par ce parcours ;
3. effectuer un parcours en largeur de A vers $N(B)$ dans $G \setminus (T \cup B)$ et soit L l'ensemble des sommets de $N(B)$ qui sont atteints par ce

parcours ;
4. soit a un sommet de A qui voit tout K ;
5. soit b un sommet de B qui voit tout L ;
6. retourner la paire $\{a, b\}$.

COMPLEXITÉ : $\mathcal{O}(n + m)$.

Lemme 5.11 *L'algorithme* PAIREDAMIS *est correct.*

Preuve. Posons $P = x\text{-}z_1\text{-}\cdots\text{-}z_p\text{-}y$, et $Z = \{z_1, \ldots, z_p\}$. Soient $A' = \{u \in C(T) \mid uz_1 \in E(G)$, et pour $2 \leq i \leq p$, $uz_i \notin E(G)\}$ et $B' = \{u \in C(T) \mid uz_i \in E(G)$, et pour $2 \leq i \leq p$, $uz_i \notin E(G)\}$. Il s'agit des ensembles mentionnés dans le lemme 5.4. Nous affirmons que les ensembles A, B définis dans l'algorithme satisfont $A = A'$ et $B = B'$. Premièrement, observons que $x \in A'$ et $y \in B'$ et qu'il n'y a pas d'arête $a'b'$ avec $a' \in A'$ et $b' \in B'$, car sinon $Z \cup \{a', b'\}$ induirait un trou impair de longueur $p + 2 \geq 5$. Cela implique que $A' \subseteq A$ et $B' \subseteq B$. Maintenant, soit a un sommet quelconque de A. Supposons que a a un voisin z_i dans Z avec $i \geq 2$, et soit i le plus grand entier de ce type. D'après la définition de A, les sommets a et y sont non-voisins. Alors $a\text{-}z_i\text{-}\cdots\text{-}z_p\text{-}y$ est un chemin T-sortant, ce qui contredit la minimalité de P. Donc a n'a pas de voisin dans $Z \setminus \{z_1\}$. Donc $A \subseteq A'$. Similairement $B \subseteq B'$. Donc $A = A'$ et $B = B'$.

Il reste à prouver que les lignes 2–5 de l'algorithme produisent des éléments maximaux de $(A, <_A)$ et $(B, <_B)$. Soit K comme défini dans l'algorithme. Soit a^* un sommet maximal pour la relation $<_A$. Supposons que a^* n'est pas adjacent à un sommet u de K. Soit $a' \in A \setminus \{a^*\}$ un voisin de u (a' existe par définition de K), et soit $Q = q_1\text{-}\cdots\text{-}q_k$ le chemin sans corde de $q_1 \in B$ à $u = q_k$ donné par le parcours de la ligne 2 de l'algorithme. Puisque A est une clique, a^* et a' sont adjacents. Supposons qu'un sommet de A est adjacent à un sommet q_i avec $1 \leq i \leq k-1$. Alors q_i est un sommet de $N(A)$ et ce parcours ne devrait pas avoir continué à partir de q_i ; mais cela contredit l'existence de q_{i+1}. Donc a' et a^* ne sont adjacents à aucun sommet de $q_1, \ldots, q_{k-1}$. Soient $Q' = q_1\text{-}\cdots\text{-}q_k\text{-}a'$ et $Q^* = q_1\text{-}\cdots\text{-}q_k\text{-}a'\text{-}a^*$. Alors Q' et Q^* sont sans corde. Le lemme 5.4 implique que Q' est de longueur paire puisque aucun de ses sommets intérieurs n'est dans $A \cup B$. Nous en déduisons que Q^* est impair, ce qui implique que $a^* <_A a'$, ce qui contredit le choix a^*. Cela prouve que tout sommet maximal pour $<_A$ est adjacent à tout K, et, par conséquent, que le sommet a de l'algorithme existe. Réciproquement, prouvons qu'un tel a est maximal pour la relation d'inclusion $<_A$. Supposons le contraire. Alors, par définition de $<_A$, il existe un chemin impair sans corde $Q'' = a\text{-}a''\text{-}q\text{-}\cdots\text{-}b''$ allant de a à un sommet $b'' \in B$ avec $a'' \in A$. Alors q est

dans $N(A)$ et sur un chemin sans corde partant de B, donc q a été atteint par le parcours de la ligne 2, donc q est dans K. Alors a est adjacent à q, ce qui contredit le fait que Q'' est sans corde. Donc a est maximal pour la relation $<_A$. La preuve est similaire pour B : un sommet de B est maximal pour la relation $<_B$ si et seulement s'il est adjacent à tout sommet de L. Maintenant, le lemme 5.4 implique que la paire $\{a, b\}$ retournée par l'algorithme est une paire d'amis spéciale de G. □

Analyse de complexité

Déterminer les ensembles A et B nécessite respectivement un temps $\mathcal{O}(d(z_1) + d(y))$ et $\mathcal{O}(d(z_p) + d(x))$. Effectuer un parcours en largeur de B vers $N(A)$ et déterminer l'ensemble K prend un temps $\mathcal{O}(m)$. Il en est de même pour l'ensemble L. De plus, à chaque fois qu'un sommet est mis dans K, nous ajoutons $+1$ aux compteurs qui sont associés à chacun de ses voisins dans A. Il en est de même pour L et B. Donc, trouver les sommets a et b prend un temps $\mathcal{O}(|A| + |B|)$.

Une paire d'amis spéciale peut être trouvée en temps $\mathcal{O}(nm)$, donc la complexité totale de l'algorithme de coloration est $\mathcal{O}(n^2m)$.

5.5 Graphes faiblement triangulés

La méthode que nous venons de présenter pour colorier les graphes d'Artémis peut être considérée comme une généralisation de l'algorithme de Hayward, Spinrad et Sritharan [42, 43], de complexité $\mathcal{O}(n^3)$, permettant de colorier les graphes faiblements triangulés. Dans un graphe faiblement triangulé, les ensembles intéressants maximaux n'ont pas de chemin sortant [66], et il n'est donc jamais nécessaire de faire appel aux algorithmes CHEMIN-SORTANT et PAIREDAMIS. La recherche récursive d'ensembles intéressants maximaux amène systématiquement à une union disjointe de cliques.

Nous montrons que les outils utilisés dans [42, 43] sont analogues à ceux utilisés dans [66]. Une *poignée* [38, 39] dans un graphe $G = (V, E)$ est un sous-ensemble $H \subset V$, de taille au moins 2, tel que $G[H]$ est connexe, qu'une composante $J \neq H$ de $G \setminus N(H)$ satisfait $N(J) = N(H)$ et que chaque sommet de $N(H)$ voit au moins un sommet de chaque arête de $G[H]$. Un tel J est appelé *co-poignée* de H. Hayward, Spinrad et Sritharan [42, 43] utilisent les poignées pour obtenir un algorithme de complexité $\mathcal{O}(m^2)$ permettant de reconnaître les graphes faiblement triangulés et un algorithme de complexité $\mathcal{O}(n^3)$ permettant de les colorier.

Les lemmes suivants mettent en évidence l'analogie entre les poignées et les ensembles intéressants.

Lemme 5.12 *Soient H une poignée de G et J une co-poignée de H. Alors J est un ensemble intéressant de $\overline{G}$.*

Preuve. D'après la définition d'une poignée, $G[J]$ est connexe. De plus, dans le graphe $\overline{G}$, nous avons $H \subseteq C(J)$, H est connexe et $|H| \geq 2$. Donc J est un ensemble intéressant dans $\overline{G}$. □

Lemme 5.13 *Soient T un ensemble intéressant maximal dans G et H une composante co-connexe de $G[C(T)]$ de taille au moins 2. Alors H est une poignée de $\overline{G}$ et T est une co-poignée de H.*

Preuve. Puisque $C(T)$ n'est pas une clique dans G, il existe une composante H de $\overline{G}[C(T)]$ de taille au moins 2. Soit $X = N_{\overline{G}}(H)$. Puisque T est connexe dans $\overline{G}$, il y a une composante T' de $\overline{G} \setminus X$ qui contient T. Si $T \neq T'$, il existe un sommet $u \in V \setminus (X \cup T)$ tel que (dans $\overline{G}$) u a un voisin dans T. Alors (dans G) $T \cup \{u\}$ est un ensemble intéressant puisque $T \cup \{u\}$ est co-connexe et $H \subseteq C(T \cup \{u\})$. Cela contredit la maximalité de T. Donc T est une composante de $\overline{G} \setminus X$. Maintenant, soit $Y = N_{\overline{G}}(T)$. Clairement $Y \subseteq X$. Dans G, tout sommet x de X a un non-voisin dans H, donc n'est pas dans $T \cup C(T)$, donc $x \in Y$. Donc $Y = X$. Finalement, supposons que (dans $\overline{G}$) un sommet $x \in X$ manque les deux sommets d'une arête de $\overline{G}(H)$. Alors (dans G) l'ensemble $T \cup \{x\}$ est un ensemble intéressant strictement plus grand que T, une contradiction. Donc H est une poignée et T est une co-poignée de H dans $\overline{G}$. □

Les résultats précédents montrent qu'un ensemble intéressant maximal de G donne une poignée de $\overline{G}$, mais qu'une poignée de $\overline{G}$ donne seulement un ensemble intéressant de G, qui n'est pas forcément maximal. Cela suggère la définition suivante qui étendra l'analogie. Une *poignée généralisée* est un sous-ensemble $H \subset V$ contenant au moins une arête, telle qu'une composante $J \neq H$ de $G \setminus N(H)$ satisfait $N(J) = N(H)$ et que tout sommet de $N(H)$ voit au moins un sommet de chaque arête de $G[H]$. Un tel J est appelé *co-poignée généralisée* de H. Cette nouvelle définition relaxe l'hypothèse de connexité de H qui ne semble pas nécessaire dans [42]. Une poignée est un cas particulier de poignée généralisée, et l'algorithme FIND-HANDLE de [42] peut être remplacé par l'algorithme suivant :

ALGORITHME POIGNÉEGÉNÉRALISÉE
ENTRÉE : Un graphe G.
SORTIE : Un couple poignée et co-poignée généralisée (H, J) de G ou la réponse "G n'a pas de poignée généralisée".
CALCUL :
- Chercher un sommet v et une arête e tels que v manque e ;
- si aucun v, e de ce type n'existe, alors retourner "G n'a pas de poignée" ;
- soit J une composante connexe de $G \setminus N(e)$ contenant v ;
- soit $H = V \setminus (J \cup N(J))$;
- tant qu'il existe v dans $N(H)$ qui manque e dans H :
  - soit J une composante connexe de $G \setminus N(e)$ qui contient v ;
  - soit $H = V \setminus (J \cup N(J))$;
- retourner (H, J)

COMPLEXITÉ : $\mathcal{O}(nm)$

Lemme 5.14 *La co-poignée produite par l'algorithme* POIGNÉEGÉNÉRALISÉE *est un ensemble intéressant maximal de* $\overline{G}$.

Preuve. D'après le lemme 5.12, J est un ensemble intéressant de $\overline{G}$. Supposons que J n'est pas maximal. Alors il existe $j \notin J$ tel que $J' = J \cup \{j\}$ est un ensemble intéressant de $\overline{G}$. Puisque $G[J']$ est connexe, j est dans $N(J)$. Cependant, $N(J) = N(H)$, donc j voit au moins un sommet de chaque arête de H. Alors $V \setminus (J' \cup N(J')) \subset V \setminus (J \cap N(J)) = H$ et donc $V \setminus (J' \cup N(J'))$ est un stable et J' n'est pas un ensemble intéressant, une contradiction. □

5.6 Commentaires

Les algorithmes présentés dans ce chapitre permettent de contracter des paires d'amis dans un graphe d'Artémis jusqu'à l'obtention d'une clique. Est-ce que l'ordre dans lequel les sommets sont contractés est un ordre de contraction amical ? Non, car il existe des graphes faiblement triangulés et donc des graphes d'Artémis qui ne sont pas fortement parfaits (voir figure 2.2). Pour ces graphes, il n'existe pas d'ordre de contraction amical. Nous ne pouvons donc pas utiliser les résultats du chapitre 2 et en particulier l'algorithme CLIQUE.

Pour trouver une clique de taille maximum dans un graphe d'Artémis il faut procéder de la manière suivante : tous les sommets de la clique finale sont marqués, les paires d'amis sont décontractés une par une en ne gardant

marqué qu'un des deux sommets qui induit une clique avec les autres sommets marqués. Cela est possible grâce au lemme 2.3 du chapitre 2. Si nous appliquons l'algorithme de contraction suivi de cette méthode à un graphe quelconque, il se peut que nous n'arrivions pas à construire une clique de taille maximum, nous pouvons alors affirmer que le graphe en entrée n'est pas un graphe d'Artémis. Nous obtenons donc un algorithme de complexité $\mathcal{O}(n^2m)$ qui permet de colorier les graphes d'Artémis de manière robuste.

Chapitre 6

Extension de pré-coloration

Etant donné un graphe G et un sous-ensemble de sommets qui ont reçu une couleur donnée, le problème d'extension de pré-coloration consiste à trouver une coloration de G qui respecte la pré-coloration et qui utilise le nombre minimum de couleurs. D'une part, ce problème généralise le problème classique de la coloration et, d'autre part, il peut être réduit au problème de la coloration sur un certain graphe contracté. Nous prouvons que l'extension de pré-coloration est polynomial pour les complémentaires des graphes de Meyniel. Nous répondons à une question de Hujter et Tuza en montrant que les graphes "PrExt-parfaits" sont exactement les graphes de co-Meyniel, ce qui généralise aussi des résultats de Hujter et Tuza et de Hertz. De plus, nous montrons que, étant donné un graphe de co-Meyniel, le graphe contracté correspondant appartient à une sous-classe de graphes parfaits : la classe des complémentaires des graphes d'Artémis, dont la perfection est plus facile à établir que le théorème fort des graphes parfaits. Cependant, la polynomialité de notre algorithme dépend toujours de la méthode des ellipsoïdes qui permet de colorier les graphes parfaits.

6.1 Contraction des pré-couleurs

Il arrive souvent qu'en optimisation, au cours de la modélisation d'un problème concret, certaines contraintes du système soient fixées a priori pour des raisons historiques, techniques ou sociales. Cela se traduit en terme de coloration par une partie des sommets qui ont déjà reçu une couleur.

Soit G un graphe. Une *pré-coloration* de G est une coloration d'un sous-graphe induit de G. Autrement dit, une pré-coloration est une famille $\mathcal{Q} = \{C_1, \ldots, C_\ell\}$ de stables deux-à-deux disjoints. Une coloration $\{S_1, \ldots, S_k\}$ de G *étend* $\mathcal{Q}$ si $\ell \leq k$ et si, pour chaque $j = 1, \ldots, \ell$, $C_j \subseteq S_j$. Etant donné

un graphe G et une pré-coloration $\mathcal{Q}$, le problème *PrExt* consiste à trouver le plus petit entier k tel que G possède une coloration en k couleurs qui étend $\mathcal{Q}$. Ce problème généralise celui de la coloration classique qui consiste à prendre $\mathcal{Q} = \emptyset$. Il est plus difficile que celui-ci puisque le problème de décision associé est NP-complet, même sur les graphes bipartis [8, 51], les graphes d'intervalles [7] ou les graphes de permutation [54]. Le problème est connu pour être polynomial sur certaines classes de graphes, les principaux résultats à ce sujet sont résumés dans [84].

Soit $\mathcal{Q} = \{C_1, \ldots, C_\ell\}$ une pré-coloration de G. Le *graphe contracté* $G/\mathcal{Q}$ désigne le graphe dont les sommets sont obtenus à partir de V en contractant, pour chaque j, l'ensemble des sommets de C_j en un seul sommet c_j. Les autres sommets restent inchangés. Une arête relie chaque paire de c_j. Les arêtes contenues dans $V \setminus (C_1 \cup \cdots \cup C_\ell)$ sont inchangées. Il y a une arête entre $v \in V \setminus (C_1 \cup \cdots \cup C_\ell)$ et c_j si v est adjacent à au moins un sommet de C_j. Cette notion de contraction est différente de celle utilisée dans le reste de cette thèse puisque les arêtes entre chaque paire de c_j sont ajoutées. La preuve du lemme suivant est évidente.

Lemme 6.1 *Pour tout graphe G et toute pré-coloration $\mathcal{Q}$ de G, l'ensemble des colorations qui étendent $\mathcal{Q}$ est en bijection avec les colorations de $G/\mathcal{Q}$. En particulier, le nombre minimum de couleurs qu'il faut pour étendre $\mathcal{Q}$ est égal à $\chi(G/\mathcal{Q})$.*

Le lemme 6.1 montre que si nous sommes capables de résoudre le problème de la coloration pour $G/\mathcal{Q}$, alors nous sommes aussi capables de résoudre PrExt pour G. Ce lemme permet donc de transposer des résultats de complexité entre le problème de la coloration et celui de l'extension de pré-coloration. En particulier, il semble naturel de se demander dans quels cas le graphe contracté est un graphe parfait, puisque nous pouvons alors résoudre PrExt en temps polynomial en coloriant le graphe contracté avec la méthode de Grötschel, Lovász et Schrijver [35].

Etant donné un graphe G, une *co-coloration* de G est une partition de $V(G)$ en cliques de G. Ici, il sera plus facile de parler de co-coloration plutôt que de coloration. Evidemment, une co-coloration de G est une coloration de $\overline{G}$, et tous les résultats que nous présentons peuvent être transposés d'un problème vers l'autre en passant aux graphes complémentaires. Etant donnée une classe de graphes $\mathcal{G}$, la classe co-$\mathcal{G}$ désigne la classe des graphes dont le complémentaire est dans $\mathcal{G}$.

Soit $\mathcal{Q} = \{C_1, \ldots, C_\ell\}$ une famille de cliques deux-à-deux disjointes. Remarquons que $\mathcal{Q}$ est une pré-coloration de $\overline{G}$; donc $\mathcal{Q}$ sera appelé *pré-co-coloration* de G. Etant donnés un graphe G et une pré-co-coloration $\mathcal{Q}$, le

problème *co-PrExt* consiste à trouver le plus petit entier k tel que G possède une co-coloration en k couleurs qui étend $\mathcal{Q}$. Le *graphe co-contracté* $G^{\mathcal{Q}}$ désigne le graphe dont les sommets sont obtenus à partir de V en contractant, pour chaque j, l'ensemble des sommets de C_j en un seul sommet c_j. Les autres sommets restent inchangés. Il n'y a aucune arête entre les paires de c_j. Les arêtes contenues dans $V \setminus (C_1 \cup \cdots \cup C_\ell)$ sont inchangées. Il y a une arête entre $v \in V \setminus (C_1 \cup \cdots \cup C_\ell)$ et c_j si v est adjacent à tous les sommets de C_j. Clairement, le graphe co-contracté $G^{\mathcal{Q}}$ est le complémentaire du graphe contracté $\overline{G}/\mathcal{Q}$. Nous démontrons le théorème suivant.

Théorème 6.2 *Le graphe co-contracté $G^{\mathcal{Q}}$ est un graphe parfait pour toute pré-co-coloration $\mathcal{Q}$ si et seulement si G est un graphe de Meyniel.*

Le sens direct est facile. Pour la réciproque, nous démontrons en fait un théorême plus fort.

Théorème 6.3 *Si G est un graphe de Meyniel et $\mathcal{Q}$ est une pré-co-coloration de G, alors le graphe co-contracté $G^{\mathcal{Q}}$ est un graphe d'Artémis.*

Le théorème 6.2 a pour conséquence algorithmique le corollaire suivant.

Corollaire 6.4 *Le problème d'extension de pré-coloration est polynomial pour les graphes de co-Meyniel.*

Preuve. PrExt sur les graphes de co-Meyniel est équivalent à co-PrExt sur les graphes de Meyniel. Etant donnés un graphe G et une co-coloration $\mathcal{Q}$ de G, le graphe co-contracté $G^{\mathcal{Q}}$ est parfait d'après le théorème 6.2. Nous pouvons alors utiliser l'algorithme polynomial de coloration des graphes parfaits [35, 82] pour trouver une co-coloration optimale de $G^{\mathcal{Q}}$. A partir de cette co-coloration, en utilisant le lemme 6.1, nous en déduisons une extension de pré-co-coloration optimale de $\mathcal{Q}$. □

Le corollaire 6.4 contient et unifie un certain nombre de résultats. Il était déjà connu [51, 52, 84] que le problème PrExt est polynomial sur les graphes scindés, les cographes, les graphes bipartis sans P_5, les compléments de graphes bipartis, et sur les graphes de co-Meyniel lorsque les classes de pré-couleur sont de taille 1 [44]. La preuve du corollaire 6.4 peut être utilisée pour déduire une conséquence algorithmique plus générale du lemme 6.1. Etant donné une classe de graphes $\mathcal{G}$, la *clôture par co-contraction* est la classe de tous les graphes qu'il est possible d'obtenir en co-contractant une pré-co-coloration quelconque d'un graphe dans $\mathcal{G}$.

Corollaire 6.5 *Soit $\mathcal{G}$ une classe de graphes. Si le problème de la co-coloration est polynomiale sur $\mathcal{G}^+$, alors co-PrExt est polynomial sur $\mathcal{G}$.*

Le corollaire 6.5 permet de réduire le problème co-PrExt sur une classe de graphes particulière $\mathcal{G}$ à la question "quel est $\mathcal{G}^+$?" et au problème de la co-coloration sur $\mathcal{G}^+$. Malheureusement, cette stratégie ne marche pas toujours, car même s'il est possible de décrire $\mathcal{G}^+$, la complexité du problème de co-coloration sur $\mathcal{G}^+$ n'est pas forcément connue. Il est donc peut être plus judicieux de tenter de traduire des résultats de co-coloration vers co-PrExt. Par exemple, ici, nous souhaitons appliquer la méthode des ellipsoïdes qui permet de (co-)colorier les graphes parfaits en temps polynomial [35, 82]. Nous posons donc la question : quelle est la classe $\mathcal{G}$ telle que $\mathcal{G}^+ = \text{Parfait}$? Malheureusement, une telle classe n'existe pas !

En effet, étant donnée une classe $\mathcal{G}$, soit $\mathcal{G}^-$ l'ensemble des graphes G tel que $G^{\mathcal{Q}}$ appartient à $\mathcal{G}$ pour toute pré-co-coloration $\mathcal{Q}$. Il est facile de voir que toute classe de graphes $\mathcal{G}$ vérifie $(\mathcal{G}^-)^+ \subseteq \mathcal{G} \subseteq (\mathcal{G}^+)^-$. Le théorème 6.2 dit que $\text{Parfait}^- = \text{Meyniel}$, et le théorème 6.3 dit que $\text{Meyniel}^+ \subseteq \text{Artemis}$, qui est une sous-classe stricte des graphes parfaits. Nous en déduisons que $(\text{Parfait}^-)^+ \neq \text{Parfait}$, et par conséquent, il n'y a pas de classe de graphes $\mathcal{G}$ telle que $\mathcal{G}^+ = \text{Parfait}$.

Cette discussion suggère une version plus faible mais plus directement utilisable du corollaire 6.5.

Corollaire 6.6 *Soit $\mathcal{G}$ une classe de graphes. Si le problème de la co-coloration est polynomial sur $\mathcal{G}$, alors co-PrExt est polynomial sur $\mathcal{G}^-$.*

Le théorème 6.2 répond aussi à une question de Hujter et Tuza [52] en caractérisant la classe des graphes *PrExt-parfaits* qui correspond précisément à la classe des graphes Parfait^-.

6.2 Graphes de Meyniel

Voici une série de lemmes concernant les graphes de Meyniel. Ces lemmes seront utilisés dans la partie suivante pour prouver les théorèmes 6.2 et 6.3.

Lemme 6.7 *Dans un graphe de Meyniel G, soit $P = v_0\text{-}\cdots\text{-}v_r$ un chemin induit et z un sommet de $V(G) \setminus V(P)$ adjacent à v_0 et v_r. Alors, soit z est adjacent à tous les sommets de P, soit r est pair et $N(z) \cap V(P) \subseteq \{v_{2i} \mid 0 \leq i \leq r/2\}$.*

Preuve. Appelons *segment* n'importe quel sous-chemin de P, de longueur au moins 1, dont les extrémités voient z et dont les sommets intérieurs manquent z. Le chemin P est partitionné (au sens des arêtes) en ses segments. Puisque G ne contient pas de trou impair, chaque segment est de longueur 1 ou paire. Supposons qu'il y ait un segment de longueur 1 et un segment de longueur paire. Alors, il y a deux tels segments consécutifs. Avec z, ces deux segments induisent une maison, ce qui contredit le fait que G est de Meyniel. En conclusion, soit tous les segments sont de longueur 1 et donc z est adjacent à tous les sommets de P, soit tous les segments sont de longueur paire et donc r est pair et $N(z) \cap V(P) \subseteq \{v_{2i} \mid 0 \leq i \leq r/2\}$. □

Lemme 6.8 *Dans un graphe de Meyniel G, soit H un trou pair ou un carré et z un sommet de $V(G) \setminus V(H)$ adjacent à deux sommets consécutifs de H. Alors z est adjacent soit à tous les sommets de H, soit à exactement trois sommets consécutifs de H.*

Preuve. Soient $v_1, \ldots, v_r$ les sommets de H ordonnés de manière circulaire. Supposons que z soit adjacent à v_1 et v_2 mais pas à tous les sommets de H et soit v_i un sommet de H qui n'est pas voisin de z. Supposons que z soit adjacent à un sommet v_j tel que $j \notin \{1, 2, 3, r\}$. Par symétrie, nous pouvons supposer que $i < j$. Alors v_1-$\cdots$-v_j ou v_2-$\cdots$-v_j est un chemin sans corde impair. Dans les deux cas, z est adjacent aux deux extrémités de ce chemin sans être adjacent à tous ses sommets, ce qui contredit le lemme 6.7. Donc $N(z) \cap V(H) \subseteq \{v_1, v_2, v_3, v_r\}$. Si z n'est adjacent ni à v_3 ni à v_r, alors $V(H) \cup \{z\}$ induit une maison, une contradiction. Si z est adjacent à la fois à v_3 et v_r alors $z, v_3, \ldots, v_r$ induisent un trou impair lorsque $r \geq 6$ ou z voit v_i lorsque $r = 4$, une contradiction. Donc z est adjacent à exactement un seul sommet de v_3, v_r et le lemme est vérifié. □

Lemme 6.9 *Dans un graphe de Meyniel G, soient Q une clique, $P = v_0$-$\cdots$-v_r un chemin sans corde de $G \setminus Q$ et z un sommet extérieur à $Q \cup V(P)$. Supposons que z et v_0 soient adjacents à tous les sommets de Q, que z ne soit adjacent ni à v_0, ni à v_1, et qu'il existe $q \in Q$ adjacent à v_r et pas adjacent à v_1. Alors v_r est adjacent à tous les sommets de Q.*

Preuve. Nous prouvons ce lemme par induction sur r. Supposons qu'un sommet $q' \in Q$ ne soit pas adjacent à v_r. D'après le lemme 6.7 appliqué à P et q, puisque q est adjacent à v_0, v_r mais pas à v_1, le chemin P a une longueur paire et $N(q) \cap V(P) \subseteq \{v_{2i} \mid i = 1, \ldots, r/2\}$. Soit $j < r$ le plus grand entier tel que q soit adjacent à v_j. Puisque j est pair, $v_j, \ldots, v_r, q$ induisent un

trou pair C. Supposons que $j = 0$. Alors q' est adjacent à q et v_0 mais pas à v_r. D'après le lemme 6.8 appliqué à C et q', le sommet q' est adjacent à v_1 et à aucun sommet de $v_2, \ldots, v_r$. Appelons C' le trou pair induit par $(V(C)\backslash\{v_0\})\cup\{q'\}$. Le sommet z est adjacent à q et q' dans C' mais pas à v_1. D'après le lemme 6.8, z est adjacent à v_r et à aucun sommet de $v_1, \ldots, v_{r-1}$. Mais alors $V(C) \cup \{z\}$ induit une maison, une contradiction. Donc $j \geq 2$. Par hypothèse d'induction, v_j est adjacent à tous les sommets de Q. Puisque q' est adjacent à q et v_j sur le trou $q, v_j, \ldots, v_r$ mais pas à v_r, le lemme 6.8 implique que q' est adjacent à v_{j+1} et à aucun sommet de $v_{j+2}, \ldots, v_r$. Mais alors $v_0, q, q', v_{j+1}, \ldots, v_r$ induisent une maison, une contradiction. □

Lemme 6.10 *Dans un graphe de Meyniel G, soient Q une clique, X un ensemble connexe de sommets de $G \setminus Q$ et z un sommet extérieur à $Q \cup X$. Supposons que z soit adjacent à tous les sommets de Q et à aucun sommet de X et que chaque sommet de X ait un non-voisin dans Q. Alors un des sommets de Q n'a aucun voisin dans X.*

Preuve. Nous prouvons le lemme par induction sur la taille de X. Si $|X| = 1$, l'énoncé est creux. Supposons donc que $|X| \geq 2$. Soient x, x' deux sommets distincts de X tels que $X\backslash\{x\}$ et $X\backslash\{x'\}$ soient connexes (par exemple, soient x, x' deux feuilles d'un arbre couvrant de X). Par hypothèse d'induction, il y a deux sommets q, q' de Q tels que q n'a pas de voisin dans $X \setminus \{x\}$ et q' n'a pas de voisin dans $X \setminus \{x'\}$. Si q n'est pas voisin de x ou si q' n'est pas voisin de x', alors le lemme est vrai. Supposons donc que q soit voisin de x et que q' soit voisin de x'. Soit P le plus court chemin de x à x' dans X. Alors, soit $V(P)\cup\{q, q'\}$ induit un trou impair, soit $V(P)\cup\{q, q', z\}$ induit une maison, une contradiction. □

6.3 Caractérisation des graphes PrExt-parfaits

Dans toute cette partie, G est un graphe et $\mathcal{Q}$ est une pré-co-coloration de G. Le sens direct dans le théorème 6.2 (c'est à dire, Parfait$^-$ $\subseteq$ Meyniel) est facile :

Lemme 6.11 *Si le graphe co-contracté $G^{\mathcal{Q}}$ est parfait pour toute pré-co-coloration $\mathcal{Q}$ de G, alors G est un graphe de Meyniel.*

Preuve. Si G contient un trou impair, alors le graphe $G^{\emptyset}$ contient aussi ce trou impair. Si G contient une maison ayant pour corde xy, alors le graphe

co-contracté $G^{\{\{x\},\{y\}\}}$ contient un trou impair. Donc G est un graphe de Meyniel. □

Le reste de cette partie est consacrée à l'étude de la classe Meyniel$^+$.

Lemme 6.12 *Soient G un graphe de Meyniel et $\mathcal{Q}$ une famille de cliques deux-à-deux disjointes de G. Alors le graphe co-contracté $G^{\mathcal{Q}}$ ne contient pas d'antitrou de taille ≥ 6.*

Preuve. Supposons que $G^{\mathcal{Q}}$ contienne un antitrou A de taille ≥ 6. Puisque l'ensemble des cliques de $\mathcal{Q}$ est co-contracté en un stable, il y a au plus deux sommets de A qui proviennent de la co-contraction d'une clique et s'il existe deux tels sommets dans A, ils sont consécutifs dans l'ordre cyclique de $\overline{A}$. S'il y a 5 sommets consécutifs de A qui ne proviennent pas de la co-contraction d'une clique, alors ils induisent une maison de G, une contradiction. Donc il n'y a pas cinq tels sommets. Donc A est de taille exactement 6 et a exactement deux sommets provenant de la co-contraction d'une clique. Soit $x_1, \ldots, x_6$ les sommets de A ordonnés de manière cyclique, tels que x_1 et x_2 soient des sommets co-contractés. Soit C_1 la clique dont la co-contraction a donné x_1. Puisque x_1 et x_6 ne sont pas adjacents, x_6 n'est pas adjacent à tous les sommets de C_1, donc il y a un sommet q_1 de C_1 qui n'est pas adjacent à x_6 dans G. Donc q_1, x_3, x_4, x_5, x_6 induisent une maison de G, une contradiction. □

Lemme 6.13 *Soient G un graphe de Meyniel et $\mathcal{Q}$ une famille disjointe de cliques de G. Alors le graphe co-contracté $G^{\mathcal{Q}}$ ne contient pas de trou impair.*

Preuve. Nous prouvons le lemme par induction sur $\ell = |\mathcal{Q}|$. Si $\ell = 0$, alors $G^{\mathcal{Q}} = G$ et le lemme est vérifié. Supposons donc que $\ell > 0$ et soit $\mathcal{Q} = \{C_1, \ldots, C_\ell\}$. Supposons que $G^{\mathcal{Q}}$ contienne un trou impair $\mathcal{H}$. Soit $x_1, \ldots, x_r$ l'ensemble des sommets de $\mathcal{H}$ ordonnés de manière cyclique. Pour chaque $j = 1, \ldots, \ell$, nous pouvons supposer que le sommet qui provient de la co-contraction de C_j appartient à $\mathcal{H}$ (sinon $\mathcal{H}$ est un trou impair de $G^{\mathcal{Q}\setminus\{C_j\}}$, ce qui contredit l'hypothèse d'induction). Soit x_{i_j} le sommet de $\mathcal{H}$ créé par la co-contraction de C_j, et sans perte de généralité, supposons que $1 < i_1 < i_2 < \cdots < i_\ell \leq r$.

Supposons que $\ell = 1$. Nous pouvons supposer que $i_1 = 1$. Puisque x_3 et x_1 ne sont pas adjacents dans $G^{\mathcal{Q}}$, x_3 n'est pas voisin de tous les sommets de C_1 dans G. Il y a donc un sommet q_1 de C_1 qui n'est pas voisin de x_3 dans G. Le chemin $P = x_2\text{-}\cdots\text{-}x_r$ est sans corde et impair, et q_1 est adjacent

aux deux extrémités de P mais pas au sommet x_3 de P, ce qui contredit le lemme 6.7. Donc $\ell \geq 2$.

Les sommets co-contractés $x_{i_1}, \ldots, x_{i_\ell}$ induisent un stable dans $\mathcal{H}$. Donc pour tout j, nous avons $i_{j+1} - i_j \geq 2$. Puisque r est impair et que $\ell \geq 2$, il existe j tel que $i_{j+1} - i_j$ est impair (et donc $i_{j+1} - i_j \geq 3$). Sans perte de généralité, nous pouvons supposer que $i_2 - i_1$ est impair et que $i_1 = 1$ (donc $i_2 \geq 4$). Soit R le chemin impair x_2-$\cdots$-x_{i_2-1}.

Puisque x_3 et x_1 ne sont pas voisins dans $G^{\mathcal{Q}}$, il y a un sommet q_1 de C_1 qui n'est pas adjacent à x_3 dans G. De même, il y a un sommet q_2 de C_2 qui n'est pas adjacent à x_{i_2-2} dans G. De plus, si $\ell \geq 3$, nous pouvons appliquer le lemme 6.10 à la clique C_j, l'ensemble connexe R et x_{i_j-1}, ce qui implique que :

Affirmation 6.14 *Pour $j = 3, \ldots, \ell$, il y a un sommet $q_j \in C_j$ qui manque tous les sommets de R.*

Maintenant, choisissons des sommets $y_1, \ldots, y_r$ de G de la manière suivante. Pour $k = 1, \ldots, r$, s'il existe j tel que $i_j = k$, poser $y_k = q_j$, sinon poser $y_k = x_k$. Les sommets $y_1, \ldots, y_r$ induisent un cycle impair H de G. Notons que, dans H, le sommet y_2 est adjacent seulement à y_1, y_3 et éventuellement à $y_{i_2} = q_2$ d'après l'affirmation 6.14. Nous montrons maintenant que :

Affirmation 6.15 *Tout voisin de q_1 dans $V(H) \setminus \{y_2, y_r\}$ est dans $\{q_2, \ldots, q_\ell\}$.*

Preuve. Soit u un voisin de q_1 dans $V(H) \setminus \{y_2, y_r\}$. Puisque x_2 et x_{i_2} ne sont pas adjacents dans $G^{\mathcal{Q}}$, il y a un sommet q_2' de C_2 qui n'est pas adjacent à y_2 dans G. Le sous-graphe de G induit par $V(H) \cup \{q_2'\} \setminus \{y_1, y_r, q_2\}$ est connexe, donc il contient un plus court chemin U de y_2 à u. Puisque y_3 est le seul voisin de y_2 dans ce sous-graphe, y_3 est le voisin de y_2 dans U. Le lemme 6.9 peut être appliqué à la clique C_1, au chemin U et au sommet y_r, ce qui implique que u est adjacent à tout C_1. Donc u doit être dans $\{q_2, \ldots, q_\ell\}$, sinon q_1 et u seraient adjacents dans $G^{\mathcal{Q}}$. □

De manière similaire, nous avons :

Affirmation 6.16 *Tout voisin de q_2 dans $V(H) \setminus \{y_{i_2-1}, y_{i_2+1}\}$ est dans $\{q_1, q_3, \ldots, q_\ell\}$.*

Maintenant, chaque sommet de $y_2, \ldots, y_{i_2-1}$ est de degré 2 dans H.

Colorons en bleu certains sommets du chemin $\mathcal{H} \setminus x_1 = x_2$-$\cdots$-$x_r$ de $G^{\mathcal{Q}}$ de la manière suivante. Les sommets x_2 et x_r sont coloriés en bleu. Pour

$j = 2, \ldots, \ell$, le sommet x_{i_j} est colorié en bleu si et seulement si tous les sommets de la clique C_j correspondante sont voisins de q_1. Tous les autres sommets de $\mathcal{H}$ restent non-coloriés. Appelons segment bleu, tout sous-chemin de longueur au moins 1 dans $\mathcal{H} \setminus x_1$ dont les deux extrémités sont bleues et dont les sommets intérieurs ne sont pas coloriés. Puisque $\mathcal{H} \setminus x_1$ est de longueur impaire et que ses extrémités sont bleues, il a un segment bleu impair. Soit x_h-$\cdots$-x_i un segment bleu impair, avec $2 \leq h < i \leq r$. Supposons que $i - h \geq 3$. Alors l'affirmation 6.15 implique que $q_1, x_h, x_{h+1}, \ldots, x_{i-1}, x_i$ induisent un trou impair dans $G^{\mathcal{Q} \setminus \{C_1\}}$, ce qui contredit l'hypothèse d'induction sur $|\mathcal{Q}|$. Donc $i - h = 1$. Puisque $i_{j+1} - i_j \geq 2$ pour tout j et $i_2 \geq 4$, cela est possible seulement si $h = n - 1$. Cela implique que x_{r-1}-x_r est le seul segment bleu impair et que tout sommet bleu x_s différent de x_r est d'indice s pair.

De manière similaire, nous colorons en rouge certains sommets du chemin $\mathcal{H} \setminus x_{i_2}$ de $G^{\mathcal{Q}}$ de la manière suivante. Les sommets x_{i_2-1} et x_{i_2+1} sont coloriés en rouge. Pour $j = 1, 2, \ldots, \ell$ et $j \neq 2$, le sommet x_{i_j} est colorié en rouge si et seulement si tous les sommets de la clique C_j correspondante sont adjacents à q_2. Appelons segment rouge, tout sous-chemin de longueur supérieure ou égale à 1 de $\mathcal{H} \setminus x_{i_2}$ dont les deux extrémités sont rouges et dont les sommets intérieurs ne sont pas rouges. Comme dans le paragraphe précédent, nous obtenons que x_{i_2+1}-x_{i_2+2} est le seul segment rouge impair, et que tout sommet rouge x_t différent de x_{i_2-1} et de x_{i_2+1} est soit d'indice t pair soit d'indice $t = 1$.

Si $i_2 = r - 1$, alors $\ell = 2$ et $V(R) \cup \{q_1, q_2, x_r\}$ induit un trou impair (si q_1, q_2 ne sont pas adjacents) ou une maison (si q_1, q_2 sont adjacents) dans G, une contradiction. Supposons donc que $i_2 \leq r - 3$. Puisque x_{i_2+2} est rouge et que x_{r-1} est bleu, il y a un sous-chemin x_s-$\cdots$-x_t de x_{i_2+2}-$\cdots$-x_{r-1} tel que x_s est rouge, x_t est bleu, et aucun des sommets intérieurs de x_s-$\cdots$-x_t n'est colorié. D'après les deux paragraphes précédents, s et t sont tous deux pairs. Si $s = t$, alors l'affirmation 6.15 implique qu'il y a une clique C_j telle que $s = i_j$, et alors $V(R) \cup \{q_1, q_2, q_j\}$ induit un trou impair (si q_1 et q_2 ne sont pas adjacents) ou une maison (si q_1 et q_2 sont adjacents), une contradiction. Donc $s \neq t$. Si q_1 et q_2 sont adjacents, alors les affirmations 6.15 et 6.16 impliquent que $\{q_1, x_s, \ldots, x_t, q_2\}$ induit un trou impair de $G^{\mathcal{Q} \setminus \{C_1, C_2\}}$, ce qui contredit l'hypothèse d'induction sur $|\mathcal{Q}|$. Si q_1 et q_2 ne sont pas adjacents, alors $V(R) \cup \{q_1, x_s, \ldots, x_t, q_2\}$ induit un trou impair dans $G^{\mathcal{Q} \setminus \{C_1, C_2\}}$, une contradiction. □

Lemme 6.17 *Soit G un graphe de Meyniel et $\mathcal{Q}$ une famille de cliques disjointes de G. Alors le graphe co-contracté $G^{\mathcal{Q}}$ ne contient pas de prisme.*

Preuve. Supposons que $G^{\mathcal{Q}}$ contient un prisme K formé des chemins $P_1 = u_0$-$\cdots$-u_r, $P_2 = v_0$-$\cdots$-v_s, $P_3 = w_0$-$\cdots$-w_t, avec $r, s, t \geq 1$ et des triangles

$A = \{u_0, v_0, w_0\}$ et $B = \{u_r, v_s, w_t\}$. D'après le lemme 6.12, le prisme K n'est pas un antitrou sur 6 sommets et nous pouvons donc supposer que l'un de r, s, t n'est pas égal à 1. D'après le lemme 6.13, le graphe $G^{\mathcal{Q}}$ ne contient pas de trou impair, donc r, s, t ont la même parité. Soit $\mathcal{Q} = \{C_1, \ldots, C_\ell\}$. Nous avons $\ell \geq 1$ puisqu'un graphe de Meyniel ne contient pas de prisme. Pour chaque $j = 1, \ldots, \ell$, appelons c_j le sommet de $G^{\mathcal{Q}}$ qui provient de la co-contraction de C_j, et soit $C = \{c_1, \ldots, c_\ell\}$. Notons que $V(K) \setminus C \subset V(G)$ et que $N(c_j) \subset V(G)$ pour chaque $j = 1, \ldots, \ell$ (puisque C est un stable). Montrons que :

Affirmation 6.18 *$|A \cap C| = 1$ et $|B \cap C| = 1$.*

Preuve. Notons que $|A \cap C| \leq 1$ et $|B \cap C| \leq 1$ puisque A, B sont des cliques et C est un stable de $G^{\mathcal{Q}}$. Maintenant, supposons par symétrie, que $A \cap C = \emptyset$. Pour chaque $j = 1, \ldots, \ell$, nous pouvons appliquer le lemme 6.10 dans G à la clique C_j, à l'ensemble connexe $A \setminus N(c_j)$ et à un voisin z de c_j dans K (plus précisément : si $c_j = u_i$ avec $i < r$, alors prenons $z = c_{i+1}$; si $c_j = u_r$ et si $s \geq 2$ ou $t \geq 2$, prenons $z = v_s$ ou $z = w_t$ respectivement ; si $c_j = u_r$ et $s = t = 1$, alors $r \geq 3$ et prenons $z = u_{r-1}$; un z similaire existe si $c_j \in (V(P_2) \cup V(P_3))$. Le lemme 6.10 implique qu'il y a un sommet $q_j \in C_j$ qui n'est voisin d'aucun sommet de $\mathcal{A} \setminus N(c_j)$. Soit P le sous-graphe de G induit par $(V(K) \setminus C) \cup \{q_1, \ldots, q_\ell\}$. Soit u'_1 le voisin de u_0 dans $P \setminus \{v_0, w_0\}$ (donc u'_1 est soit u_1 soit un q_j), et de même soit v'_1 le voisin de v_0 dans $P \setminus \{u_0, w_0\}$. Soit R un plus court chemin de u'_1 à v'_1 dans $P \setminus \{u_0, v_0, w_0\}$. Alors $V(R) \cup \{u_0, v_0, w_0\}$ induit une maison ou $V(R) \cup \{u_0, v_0\}$ induit un trou impair, une contradiction. □

D'après l'affirmation 6.18 et par symétrie, nous pouvons supposer que $u_0 = c_1$, et donc v_0, w_0 sont des sommets de G. Comme ci-dessus, d'après le lemme 6.10, pour $j = 2, \ldots, \ell$, nous pouvons choisir un sommet q_j dans C_j voisin d'aucun sommet de $\{v_0, w_0\} \setminus N(c_j)$. Montrons maintenant que :

Affirmation 6.19 *Les sommets v_1 et w_1 de $G^{\mathcal{Q}}$ sont dans C.*

Preuve. Supposons, par symétrie, que v_1 n'est pas dans C. Nous pouvons alors choisir un sommet $q'_1 \in C_1$ qui n'est pas adjacent à v_1. Soit P le sous-graphe de G induit par $(V(K) \setminus C) \cup \{q'_1, q_2, \ldots, q_\ell\}$. Soit R le plus court chemin de q'_1 à v_1 dans $P \setminus \{v_0, w_0\}$. Alors R est de longueur au moins 2 et $V(R) \cup \{v_0, w_0\}$ induit une maison ou $V(R) \cup \{v_0\}$ induit un trou impair, une contradiction. □

D'après l'affirmation 6.19, nous pouvons supposer que $v_1 = c_2$ et $w_1 = c_3$. Rappelons que q_2 est un sommet de C_2 qui n'est pas adjacent à w_0 et que q_3

est un sommet de C_3 qui n'est pas adjacent à v_0. Puisque v_1 et w_1 ne sont pas voisins, les longueurs s, t de P_2, P_3 ne peuvent pas être toutes deux égales à 1. Supposons par symétrie que $s \geq 2$. Nous montrons que :

Affirmation 6.20 *Le sommet q_2 est adjacent à tout C_1.*

Preuve. Supposons que q_2 n'est pas adjacent à un sommet $q_1'' \in C_1$. Soit P le sous-graphe de G induit par $(V(K) \setminus C) \cup \{q_1'', q_2, \ldots, q_\ell\}$. Soit R le plus court chemin de q_1'' à q_2 dans $P \setminus \{v_0, w_0\}$. Alors R est de longueur au moins 2 et $V(R) \cup \{v_0, w_0\}$ induit une maison ou $V(R) \cup \{v_0\}$ induit un trou impair, une contradiction. □

Maintenant, soit q_1 un sommet de C_1, et soit P le sous-graphe de G induit par $(V(K) \setminus C) \cup \{q_1, \ldots, q_\ell\}$. Nous montrons que :

Affirmation 6.21 *Tout voisin de q_2 dans $V(P) \setminus \{v_0, v_2\}$ est dans $\{q_1, \ldots, q_\ell\}$.*

Preuve. Soit x un voisin de q_2 dans $V(P) \setminus \{v_0, v_2\}$. Nous pouvons supposer que $x \neq q_1$. Le sous-graphe $P \setminus \{q_1, q_2, v_2\}$ est connexe, donc il contient un plus court chemin X de v_0 à x. Puisque v_0 n'a pas de voisin dans $V(P) \setminus \{q_1, q_2, w_0\}$, le voisin de v_0 dans X est w_0. Maintenant, le lemme 6.9 peut être appliqué à la clique C_2, au chemin X et au sommet v_2, ce qui implique que x est adjacent à tout C_2. Cela signifie que x est dans $\{q_1, \ldots, q_\ell\}$, car sinon v_1 et x seraient adjacents dans $G^{\mathcal{Q}}$. □

Dans $G^{\mathcal{Q}}$, marquons certains sommets de $K \setminus v_1$ de la manière suivante. Les sommets v_0 et v_2 sont marqués. Pour $j = 1, \ldots, \ell$ et $j \neq 2$, le sommet c_j est marqué si et seulement si, dans G, le sommet q_2 est voisin de tous les sommets de la clique C_j correspondante de G. Tous les autres sommets de K sont non marqués. Appelons *segment* un sous-chemin de longueur au moins 1 de $K \setminus v_1$ dont les deux extrémités sont marquées et les sommets intérieurs sont non-marqués. Supposons qu'il existe un segment impair X de longueur ≥ 3. Alors $V(X) \cup \{q_2\}$ induit un trou impair dans $G^{\mathcal{Q} \setminus \{C_2\}}$, ce qui contredit le lemme 6.13. Donc tout segment est de longueur paire ou égale à 1. Notons que $V(P_1) \cup V(P_2) \setminus \{v_0, v_1\}$ induit un chemin sans corde de longueur impaire (car r, s ont la même parité), et ses deux extrémités sont marquées ; donc ce chemin contient un segment impair, qui, comme noté ci-dessus est de longueur 1. Appelons y le voisin de v_2 sur ce chemin. Remarquons que nous avons soit $s \geq 3$ et $y = v_3$, soit $s = 2$ et $y = u_r$. D'après l'affirmation 6.21 et le fait que les sommets de C sont deux-à-deux non-adjacents, le seul segment possible de longueur 1 est v_2-y. Donc y est marqué, et l'affirmation 6.21

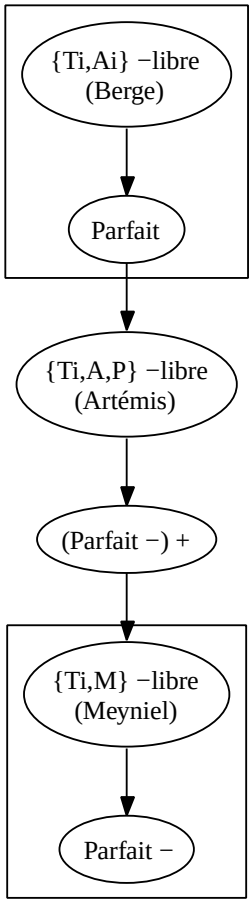
implique $y \in C$. Supposons que $s \geq 3$. Alors $V(P_3) \cup V(P_2) \setminus \{v_1, v_2\}$ induit un chemin sans corde impair, dont les extrémités sont marquées, et donc il contient un segment de longueur 1. Le seul tel segment possible est v_0-w_0. Donc w_0 est marqué, et l'affirmation 6.21 implique $w_0 \in C$, ce qui contredit l'affirmation 6.18. Donc $s = 2$ et $y = u_r$. Maintenant, puisque B contient u_r il ne peut pas contenir un autre sommet de C, donc w_t n'est pas dans C, ce qui, d'après l'affirmation 6.19, implique que $t \geq 2$. La symétrie entre s et t est rétablie, et comme ci-dessus, nous pouvons prouver que $t = 2$ et que q_3 est voisin de u_r. Mais alors q_2, v_0, w_0, q_3, u_r induisent un trou impair ou une maison de G, une contradiction. □

Les lemmes 6.12, 6.13 et 6.17 impliquent que $G^{\mathcal{Q}}$ est un graphe d'Artémis, ce qui prouve le théorème 6.3 et la réciproque du théorème 6.2.

6.4 Commentaires

Nous avons prouvé que Parfait$^-$ = Meyniel et que (Parfait$^-$)$^+$ = Meyniel$^+$ $\subseteq$ Artemis $\subsetneq$ Parfait. L'amélioration de Parfait vers Artémis (dans la dernière inclusion qui est stricte) a deux aspects intéressants : premièrement, la perfection de la classe des graphes d'Artémis [66] est plus facile à établir que la perfection des graphes de Berge [14]. Deuxièmement, puisque co-PrExt est polynomial sur les graphes de Meyniel avec la méthode des ellipsoïdes, la question qui se pose est de trouver un algorithme combinatoire pour résoudre ce problème. Nous avons vu au chapitre 5 un algorithme de partition en stables des graphes d'Artémis, mais ici il faudrait un algorithme de partition en cliques des graphes d'Artémis. Ce problème reste ouvert. Par ailleurs, la classe des graphes Meyniel$^+$ n'est pas complètement caractérisée, nous savons qu'elle est incluse dans la classe des graphes d'Artémis mais c'est probablement une sous-classe stricte des graphes d'Artémis.

La figure 6.1 résume les relations d'inclusion entre les classes de graphes considérées dans ce chapitre. Le cadre du haut correspond au théorème fort des graphes parfaits et le cadre du bas au théorème 6.2. Remarquons que les graphes de Meyniel et les graphes d'Artémis, deux classes de graphes étudiées par ailleurs dans cette thèse à propos du problème classique de la coloration, apparaissent naturellement dans le cadre du problème PrExt.



T = Trou A = Antitrou P = Prisme M = Maison i = impair

FIG. 6.1 – Relations d'inclusion entre les classes de graphes considérées

Conclusion et perspectives

La démarche globale de cette thèse est une méthode de recherche assez générale en théorie des graphes, valable aussi dans d'autres domaines.

Nous nous intéressons au problème de trouver un algorithme efficace de type combinatoire permettant de colorier la classe des graphes parfaits. Ce problème étant trop difficile à résoudre dans toute sa généralité, nous nous restreignons à des sous-classes de graphes parfaits.

Nous étudions les propriétés structurelles de ces classes qui permettent de définir des algorithmes efficaces pour résoudre le problème de la coloration. Plus ces algorithmes sont simples, plus ils seront susceptibles d'être utilisés et modifiés pour colorier d'autres classes de graphes.

D'autre part ces résultats permettent d'avoir une meilleure compréhension de la problématique générale. A titre d'exemple, nous pouvons mentionner que Chudnovsky et Seymour [15] ont récemment utilisé les résultats de Maffray et Trotignon [66] à propos de la coloration des graphes d'Artémis pour simplifier la preuve du théorème fort des graphes parfaits [14].

Dans cette thèse, plusieurs perspectives ont été données pour des recherches futures. Une question qui suscite actuellement beaucoup d'intérêt dans la communauté est la conjecture d'Everett et Reed [75] à propos des graphes parfaitement contractiles. Dans le chapitre 2, nous proposons une conjecture analogue concernant la notion de paire P_4-libre. La résolution d'une de ces conjectures serait un grand pas en avant vers la coloration des graphes parfaits.

Bibliographie

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Menlo Park, California, 1974.

[2] C. Berge, Les problèmes de coloration en théorie des graphes, *Publications de l'Institut de Statistiques de l'Université de Paris* (1960) 123–160.

[3] C. Berge, Some classes of perfect graphs, in *Six papers on Graph Theory*, Calcutta, 1963, Indian Statistical Institute.

[4] C. Berge, *Graphs*, North-Holland, Amsterdam/New York, 1985.

[5] M.E. Bertschi, Perfectly contractile graphs, *Journal of Combinatorial Theory B* 50 (1990) 222–230.

[6] D. Bienstock, On the complexity of testing for odd holes and induced odd paths, *Discrete Mathematics* 90 (1991) 85–92. See also Corrigendum by B. Reed, *Discrete Mathematics* 102 (1992) 109.

[7] M. Biró, M. Hujter, Zs. Tuza, Precoloring extension I : interval graphs, *Discrete Mathematics* 100 (1992) 267–279.

[8] H.L. Bodlaender, K. Jansen, G. Woeginger, Scheduling with incompatible jobs, *Discrete Applied Mathematics* 55 (1994) 219–232.

[9] M. Burlet, J. Fonlupt, Polynomial algorithm to recognize a Meyniel graph, *Annals of Discrete Mathematics* 21 (1984) 225–252.

[10] K. Cameron, J. Edmonds, Existentially polytime theorems, *DIMACS Series Discrete Mathematics and Theorical Computer Science* 1 (1990) 83–99.

[11] K. Cameron, J. Edmonds, Finding a strong stable set or a Meyniel obstruction in any graph, manuscript, 2005.

[12] K. Cameron, J. Edmonds, B. Lévêque, F. Maffray, Coloring vertices of a graph or finding a Meyniel obstruction, manuscript, 2007.

[13] M. Chudnovsky, G. Cornuéjols, X. Liu, P. Seymour, K. Vušković, Recognizing Berge graphs, *Combinatorica* 25 (2005) 143–186.

[14] M. Chudnovsky, N. Robertson, P. Seymour, R. Thomas, The strong perfect graph theorem, *Annals of Mathematics* 164 (2006) 51–229.

[15] M. Chudnovsky, P. Seymour, Even pairs in Berge graphs, manuscript, 2007.

[16] V. Chvátal, Perfectly ordered graphs, *Topics on perfect graphs* 63–65, North-Holland, Amsterdam-New York, 1984.

[17] V. Chvátal, A class of perfectly orderable graphs, Report 89573-OR, Forschungsinstitut für Diskrete Mathematik, Bonn (1989).

[18] V. Chvátal, I. Rusu, R. Sritharan, Dirac-type characterizations of graphs without long chordless cycles, *Discrete Mathematics* 256 (2002) 445–448.

[19] V. Chvátal, N. Sbihi, Bull-free Berge graphs are perfect, *Graphs and Combinatorics* 3 (1987) 127–139.

[20] S. Dasgupta, C H. Papadimitriou, U. Vazirani, *Algorithms*, McGraw-Hill, 2006.

[21] C.M.H. de Figueiredo, F. Maffray, Optimizing bull-free perfect graphs, *SIAM Journal on Discrete Mathematics* 18 (2004) 226–240.

[22] C.M.H. de Figueiredo, F. Maffray, O. Porto, On the structure of bull-free perfect graphs, *Graphs and Combinatorics* 13 (1997) 31–55.

[23] C.M.H. de Figueiredo, F. Maffray, O. Porto, On the structure of bull-free perfect graphs 2 : the weakly chordal case, *Graphs and Combinatorics* 17 (2001) 435–456.

[24] R. Diestel, *Graph Theory*, Springer, New York, second edition, 2000.

[25] J. Edmonds, Minimum partition of a matroid into independent subsets, *Journal of Research of the National Bureau of Standards Sect. B* 69 (1965) 67–72.

[26] J. Edmonds, Paths, trees and flowers, *Canadian Journal of Mathematics* 17 (1965) 449–467.

[27] Euclide, *Les éléments, Volume II, Livres V à IX*, traduction du texte de Heiberg et commentaires par Bernard Vitrac, Presses universitaires de France, Paris, 1994.

[28] H. Everett, C.M.H. de Figueiredo, C. Linhares Sales, F. Maffray, O. Porto, B.A. Reed, Even pairs, *Perfect Graphs* 67–92, Wiley Interscience, 2001.

[29] J. Fonlupt, J.P. Uhry, Transformations that preserve perfectness and h-perfectness of graphs, *Annals of Discrete Mathematics* 16 (1982) 83–85.

[30] D.R. Fulkerson, Anti-blocking polyhedra, *Journal of Combinatorial Theory B* 12 (1972) 50–71.

[31] T. Gallai, Transitiv Orientierbare Graphen, *Acta Mathematics Acad. Sci. Hungar.* 18 (1967) 25–66. See [65] for an English translation.

[32] M.R. Garey, D.S. Johnson, *Computers and Intractability*, Freeman, San Francisco, California, 1979.

[33] A. Ghouila-Houri, Sur une conjecture de Berge (mimeo), 1960.

[34] M.C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[35] M. Gröstchel, L. Lovász, A. Schrijver, The ellipsoid method and its consequences in combinatorial optimization, *Combinatorica* 1 (1981) 169–197.

[36] M. Habib, F. de Montgolfier, C. Paul, A simple linear-time modular decomposition algorithm for graphs, using order extension, Proceedings of SWAT 2004, *Lecture Notes in Computer Science* 3111 (2004) 187–198.

[37] R.B. Hayward, Weakly triangulated graphs, *Jounal of Combinatorial Theory B* 39 (1985) 200–208.

[38] R.B. Hayward, Meyniel weakly triangulated graphs I : co-perfect orderability, *Discrete Applied Mathematics* 73 (1997) 199–210.

[39] R.B. Hayward, Meyniel weakly triangulated graphs II : A theorem of Dirac, *Discrete Applied Mathematics* 78 (1997) 283–289.

[40] R.B. Hayward, Bull-free weakly chordal perfectly orderable graphs, *Graphs and Combinatorics* 17 (2000) 479–500.

[41] R.B. Hayward, C.T. Hoàng, and F. Maffray, Optimizing weakly triangulated graphs, *Graphs and Combinatorics* 5 (1989) 339–349. See also Erratum in vol. 6 (1990) 33–35.

[42] R.B. Hayward, J.P. Spinrad, R. Sritharan, Weakly chordal graph algorithms via handles, *Proceedings of the 11th annual ACM-SIAM Symposium on Discrete Algorithms* (2000) 42–49.

[43] R.B. Hayward, J.P. Spinrad, R. Sritharan, Improved algorithms for weakly chordal graphs, *ACM Transactions on Algorithms* 3 (2007).

[44] A. Hertz, Slim graphs, *Graphs and Combinatorics* 5 (1989) 149–157.

[45] A. Hertz, A fast algorithm for coloring Meyniel graphs, *Journal of Combinatorial Theory B* 50 (1990) 231–240.

[46] A. Hertz, COSINE : A new graph coloring algorithm, *Operation Research Letters* 10 (1991) 411–415.

[47] A. Hertz, D. de Werra, Perfectly orderable graphs are quasi-parity graphs : a short proof, *Discrete Mathematics* 68 (1988) 111–113.

[48] A. Hertz, D. de Werra, Connected sequential colorings, *Discrete Mathematics* 74 (1989) 51–59.

[49] C.T. Hoàng, On a conjecture of Meyniel, *Journal of Combinatorial Theory B* 42 (1987) 302–312.

[50] C. T. Hoàng, Efficient algorithms for minimum weighted colouring of some classes of perfect graphs, *Discrete Applied Mathematics* 55 (1994) 133–143.

[51] M. Hujter, Zs.M. Tuza, Precoloring extension II : Graphs classes related to bipartite graphs, *Acta Math. Univ. Comen., New Ser.* 62 (1993) 1–11.

[52] M. Hujter, Zs.M. Tuza, Precoloring extension III : Classes of perfect graphs, *Combinatorics, Probability and Computing* 5 (1996) 35–56.

[53] B. Jamison, S. Olariu, On the semi-perfect elimination, *Advances in Applied Mathematics* 9 (1988) 364–376.

[54] K. Jansen, The optimum cost chromatic partition problem, Proceeding of CIAC 3, *Lecture Notes in Computer Science* 1203 (1997) 25–36.

[55] V. Jost, B. Lévêque, F. Maffray, Precoloring extension of co-Meyniel graphs, *Graphs and Combinatorics* 23 (2007) 291–301.

[56] R. Karp, Reducibility among combinatorial problems, *Complexity of Computer Computations* 85–103, Plenum Press, 1972.

[57] L.G. Khachiyan, A polynomial algorithm in linear programming, *Soviet Mathematics Doklady* 20 (1979) 191–194.

[58] B. Lévêque, F. Maffray, Coloring bull-free perfectly contractile graphs, to appear in *SIAM Journal on Discrete Mathematics.*

[59] B. Lévêque, F. Maffray, B.A. Reed, N. Trotignon, Coloring Artemis graphs, manuscript, 2007.

[60] C. Linhares Sales, F. Maffray, and B.A. Reed, On planar perfectly contractile graphs, *Graphs and Combinatorics* 13 (1997) 167–187.

[61] L. Lovász, A characterization of perfect graphs, *Journal of Combinatorial Theory B* 13 (1972) 95–98.

[62] L. Lovász, Normal hypergraphs and the perfect graph conjecture, *Discrete Mathematics* 2 (1972) 253–267.

[63] A. Lubiw, Doubly lexical ordering of matrices, *SIAM Journal on Computing* 16 (1987) 854–879.

[64] F. Maffray, On the coloration of perfect graphs, *Recent Advances in Algorithmic Combinatorics* (Proceedings of the 1st Brazilian School on Combinatorics and Algorithms), Springer , 2003.

[65] F. Maffray, M. Preissmann, A translation of Tibor Gallai's paper : Transitiv orientierbare Graphen, *Perfect graphs* 25-66, Wiley Interscience, 2001.

[66] F. Maffray, N. Trotignon, A class of perfectly contractile graphs, *Journal of Combinatorial Theory B* 96 (2006) 1–19.

[67] S.E. Markosyan, I.A. Karapetyan, Perfect graphs, *Akad. Nauk Armjan. SSR Dokl.* 63 (1976) 292–296.

[68] H. Meyniel, On the perfect graph conjecture, *Discrete Mathematics* 16 (1976) 334–342.

[69] H. Meyniel, The graphs whose odd cycles have at least two chords, *Annals of Discrete Mathematics* 21 (1984) 115–119.

[70] H. Meyniel, A new property of critical imperfect graphs and some consequences, *European Journal of Combinatorics* 8 (1987) 313–316.

[71] M. Middendorf and F. Pfeiffer, On the complexity of recognizing perfectly orderable graphs, *Discrete Mathematics* 80 (1990) 327–333.

[72] C. H. Papadimitriou, *Computational complexity*, Addison-Wesley, Reading, 1994.

[73] V. Raghavan, J. Spinrad, Robust algorithms for restricted domains, *Journal of Algorithms* 48 (2003) 160–172.

[74] G. Ravindra, Meyniel's graphs are strongly perfect, *Annals of Discrete Mathematics* 21 (1984) 145–148.

[75] B.A. Reed, Problem session on parity problems (Public communication), *DIMACS Workshop on Perfect Graphs*, Princeton University, New Jersey, 1993.

[76] B. Reed, A gentle introduction to semi-definite programming, *Perfect graphs* 233–259, Wiley Interscience, 2001.

[77] B.A. Reed, N. Sbihi, Recognizing bull-free perfect graphs, *Graphs and Combinatorics* 11 (1995) 171–178.

[78] D.J. Rose, R.E. Tarjan, G.S. Lueker, Algorithmic aspects of vertex elimination of graphs, *SIAM Journal on Computing* 5 (1976) 266–283.

[79] F. Roussel, I. Rusu, Holes and dominoes in Meyniel graphs, *International Journal of Foundations of Computer Science* 10 (1999) 127–146.

[80] F. Roussel, I. Rusu, A linear algorithm to color i-triangulated graphs, *Information Processing Letters* 70 (1999) 57-62.

[81] F. Roussel, I. Rusu, An $\mathcal{O}(n^2)$ algorithm to color Meyniel graphs, *Discrete Mathematics* 235 (2001) 107–123.

[82] A. Schrijver, *Combinatorial Optimization : Polyhedra and Efficiency*, Springer, 2003.

[83] N. Trotignon, Graphes parfaits : structure et algorithmes, Thèse de doctorat, Université Joseph Fourier, Grenoble, France, 2004.

[84] Zs. Tuza, Graph colorings with local constraints—a survey, *Discuss. Math. Graph Theory* 17 (1997) 161–228.

Index

COLORATION DE GRAPHES : STRUCTURES ET ALGORITHMES

Résumé : De nombreux problèmes appliqués peuvent être modélisés par le problème de la coloration des sommets d'un graphe, qui est NP-complet en général mais polynomial sur la classe des graphes parfaits introduite par Berge. L'algorithme de coloration des graphes parfaits, de Grötschel, Lovász et Schrijver, n'est pas réellement efficace d'un point de vue pratique et il est toujours intéressant de trouver un algorithme "purement" combinatoire permettant de colorier les graphes parfaits en temps polynomial.

Dans cette thèse, nous donnons plusieurs algorithmes simples et rapides permettant de colorier des sous-classes de graphes parfaits. Ces algorithmes utilisent en particulier la notion de contraction de paire d'amis, introduite par Fonlupt et Uhry, à propos de laquelle plusieurs conjectures sont encore ouvertes. Nous utilisons aussi des algorithmes de parcours comme LexBFS, de Rose, Tarjan et Lueker, pour prouver des résultats structuraux sur les graphes considérés.

Mots-clés : graphe parfait, algorithme, coloration, paire d'amis, LexBFS, extension de pré-coloration

COLORING GRAPHS : STRUCTURES AND ALGORITHMS

Abstract : Many applied problems can be modelised by the vertex coloring problem of a graph, which is NP-complete in general but polynomial on the class of perfect graphs introduced by Berge. The coloring algorithm of perfect graphs, due to Grötschel, Lovász and Schrijver, is not really efficient in practice and it is still interesting to find a "purely" combinatorial algorithm to color perfect graphs in polynomial time.

In this thesis, we give several simple and fast algorithms that enable to color some subclasses of perfect graphs. These algorithms use in particular the notion of even pair contraction, introduced by Fonlupt and Uhry, about which several conjectures are still open. We also use search algorithms like LexBFS, due to Rose, Tarjan and Lueker, to prove some structural results on the considered graphs.

Keywords : perfect graph, algorithm, coloring, even pair, LexBFS, pre-coloring extension

COLORATION DE GRAPHES : STRUCTURES ET ALGORITHMES

Benjamin Lévêque

2007