UNIVERSITÉ DE LORRAINE

AVERTISSEMENT

Ce document est le fruit d'un long travail approuvé par le jury de soutenance et mis à disposition de l'ensemble de la communauté universitaire élargie.

Il est soumis à la propriété intellectuelle de l'auteur. Ceci implique une obligation de citation et de référencement lors de l'utilisation de ce document.

D'autre part, toute contrefaçon, plagiat, reproduction illicite encourt une poursuite pénale.

Contact : ddoc-theses-contact@univ-lorraine.fr

LIENS

Code de la Propriété Intellectuelle. articles L 122. 4
Code de la Propriété Intellectuelle. articles L 335.2- L 335.10
http://www.cfcopies.com/V2/leg/leg_droi.php
http://www.culture.gouv.fr/culture/infos-pratiques/droits/protection.htm

Thèse

présentée

devant l'Université de Lorraine

pour obtenir

le grade de Docteur de l'Université de Lorraine

Mention Informatique

par

Jean-François Couturier

Équipe d'accueil : Calculs, Modélisation et Interfaces - LITA
École Doctorale : IAEM
Composante universitaire : UFR - MIM

Titre de la thèse :

Algorithmes exacts et exponentiels sur les graphes : énumération, comptage et optimisation

Soutenue le 6 Décembre 2012 devant la commission d'examen.
Composition du jury :

Président		
Michel	HABIB	Pr, LIAFA Université Paris Diderot
Rapporteurs		
Cyril	GAVOILLE	Pr, LaBRI Université de Bordeaux
Christophe	PAUL	Dr, LIRMM Montpellier
Examinateurs		
Frédéric	MAFFRAY	Dr, G-SCOP Grenoble
Henning	FERNAU	Pr, Universität Trier - Allemagne
Anass	NAGIH	Pr, LITA Université de Lorraine
Dieter	KRATSCH	Pr, LITA Université de Lorraine (Directeur de thèse)

Table des matières

Remerciements

Pour commencer, je tiens à remercier Dieter Kratsch, mon directeur de thèse, sans qui je n'aurais même pas envisagé cette aventure. Son accompagnement, ses conseils, son soutien et toutes les choses qu'il m'a apprises ne pourrons se résumer en quelques lignes. Il est et restera toujours le meilleur mentor dont j'aurais pu rêver.

Je tiens aussi à remercier Nora Kratsch, et leur fils Stefan, pour l'accueil qu'ils mon réservé lors de mes voyages à Jena.

Je remercie chaleureusement Christophe Paul et Cyril Gavoille, rapporteurs de cette thèse, pour l'attention et l'intérêt qu'ils ont porté à mes travaux, ainsi que pour leurs précieux commentaires. Je tiens également à remercier sincèrement Michel Habib, Frédéric Maffray, Anass Nagih et Hening Fernau pour avoir accepté d'être les examinateurs de ma thèse.

Je tiens particulièrement à remercier toutes les personnes qui ont travaillé avec moi durant ces trois ans, et notamment mes co-auteurs : Petr Golovach, Daniel Paulusma, Pinar Heggernes, Pim van 't Hof et Yngve Villanger. Je remercie particulièrement Pinar Heggernes pour son invitation à Bergen durant le mois de janvier 2012.

Je remercie aussi les membres du laboratoire LIFO d'Orléans, et plus particulièrement Ioan Todinca pour son soutien et ses conseils avisés. Je remercie le projet ANR-blanc-AGAPE et tous ses membres pour le financement dont j'ai pu bénéficier et pour les discussions que nous avons pu avoir.

Je tiens aussi à avoir une pensée toute particulière pour Mathieu Liedloff, qui fut pour moi un modèle et dont les conseils et le soutien furent appréciables.

Bien entendu, je remercie les membres du LITA, et le département informatique de l'Université de Metz (puis Université de Lorraine). Je remercie aussi mes camarades doctorants de l'UFR MIM, et notamment Vincent Demange pour les bons moments passés à la cantine.

Enfin, je tiens à remercier ma famille et mes amis pour m'avoir supporté pendant ces trois ans. Je remercie ma chérie Marie Meier, pour son soutien et ses encouragements, ainsi que pour son aide sur les questions orthographiques. Je remercie mon ami Laurent Brunet de m'avoir prêté son imprimante pour les pages en couleurs. Je remercie mon père, d'avoir fait de son mieux pour que je puisse accomplir cette thèse dans les meilleures conditions possibles, et j'ai une pensée pour ma mère, qui, si elle le pouvait, serait sans doute très fière de moi.

À vous tous, Merci ...
Jean-François Couturier.

Première partie

Prologue

Chapitre 1

Introduction

1.1 Les problèmes difficiles

En 1971, S. Cook publie un document [17], dans lequel il démontre la NP-complétude du problème SAT. Peu de temps après, R. Karp [58] établira une liste de vingt-et-un problèmes telle qu'il y a pour chacun de ces problèmes une réduction en temps polynomial de SAT à ce problème, et qu'ils sont donc à leur tour NP-complet. C'est le début de l'engouement pour l'étude des problèmes de la classe de complexité NP. Depuis, l'intérêt pour ces problèmes, mais aussi le nombre de problèmes prouvés être difficile n'a cessé d'augmenter.

Malgré de nombreuses tentatives, personne encore n'a su répondre à ce qui est peut-être la plus grande question actuelle en informatique : est-ce que $P = NP$? Il s'agit de savoir si deux classes de complexité : la classe P représentant les problèmes pouvant être résolus en temps polynomial sur des machines de Turing déterministes, et la classe NP représentant les problèmes pouvant être résolus en temps polynomial sur des machines de Turing non-déterministes, sont égales ou pas. L'égalité signifierait que chaque problème dans NP, et donc chaque problème NP-complet est solvable en temps polynomial. Au contraire, l'inégalité signifierait qu'aucun problème NP-complet ne peut être résolu par un algorithme polynomial. Ces notions sont plus amplement expliquées dans l'ouvrage de C.H. Papadimitriou [83].

Même si cette question reste pour l'instant sans réponse, elle continue à susciter énormément d'intérêt, notamment à cause des nombreuses applications possibles à ces problèmes. En mathématiques et en informatique, bien sûr, mais aussi en biologie, en astrophysique, ou même en stratégie.

Regardons des problèmes NP-complet classiques. Étant donné un graphe à n sommets, existe-t-il un chemin passant par tous les sommets, mais une

seule fois pour chaque sommet (Chemin Hamiltonien) ? Ou encore, étant donnés un graphe G et un entier k, peut-on choisir au plus k sommets tels que toutes les arêtes du graphe soient adjacentes à l'un des sommets choisis (Vertex Cover) ? Bien que ces problèmes soient relativement simples à décrire, il n'est généralement pas évident de déterminer à quel point leur résolution sera facile ou difficile.

Même si la résolution est difficile, elle n'est pas impossible, et une énumération exhaustive de toutes les solutions possibles à fin de comparaison permet toujours de trouver la bonne solution.

Mais cette énumération pose une difficulté : le nombre de solutions qui peuvent exister peut être très important ! Si l'instance du problème est importante, le nombre de solutions possibles l'est d'autant plus, et le temps pour les énumérer peut vite devenir monstrueux.

Le sujet de cette thèse est d'étudier un certain nombre d'algorithmes pour permettre de résoudre différents problèmes difficiles, par des méthodes plus efficaces qu'une simple recherche exhaustive. Bien que plus efficaces que la recherche exhaustive, nos algorithmes gardent un temps d'exécution important, dans le pire des cas exponentiel au rapport de la taille de l'instance du problème.

1.1.1 Problèmes difficiles et NP-complétude

Les problèmes NP-complets sont, par définition, des problèmes de décision. Ils sont posés de façon à ce que l'on puisse répondre simplement par oui ou par non, comme on peut le voir dans la définition de deux importants problèmes NP-complets. Une formule logique donnée en forme normale conjonctive dispose-t-elle d'au moins une assignation possible de ses variables de façon à ce que la formule soit vraie (SAT) ? Existe-t-il dans un graphe $G = (V, E)$ un ensemble $S \subseteq V$ de taille au plus k tel que quelque soit $x \in S$ et $y \in S$, alors $xy \notin E$ (Ensemble Stable) ?

Mais les problèmes de décision ne sont pas les seuls problèmes intéressants. Certaines questions nécessitent une réponse plus complète et plus complexe qu'un simple oui ou non. Dans cette thèse, nous étudierons d'autres formes de problèmes que nous appellerons des problèmes d'optimisation, des problèmes de comptage et des problèmes d'énumération.

Pour tous les problèmes difficiles que nous étudierons dans cette thèse, il existe un problème de décision associé qui est NP-complet. On dit que les problèmes que nous étudions sont NP-difficiles, parce qu'ils permettent de donner une réponse aux problèmes de décision NP-complets qui leurs sont

associés. Mais les problèmes NP-difficiles, ne font pas forcément eux-même partie de la classe NP.

Les problèmes d'optimisation

Les problèmes d'optimisation sont une extension naturelle des problèmes de décision. En effet, à la question de savoir s'il existe une solution de taille k possédant une propriété donnée, il est naturel de se demander quelle est la plus petite solution, ou la plus grande, qui possède la dite propriété.

Chaque problème d'optimisation dispose d'une version de décision qui lui est associée. Par exemple, le problème de domination consiste à trouver la taille du plus petit ensemble dominant dans un graphe. Sa version de décision associée est la suivante : étant donnés un graphe G et un entier k, existe-t-il un ensemble dominant de taille au plus k. Typiquement, la complexité de temps d'un problème d'optimisation et la complexité de temps de sa version associée de décision sont équivalentes, si l'un est polynomial, alors l'autre l'est aussi.

On considèrera que les problèmes de décision et d'optimisation sont de complexités équivalentes. Par abus de langage, les problèmes d'optimisation sont aussi désignés comme étant NP-complets.

Les problèmes de comptage

Les problèmes de comptage sont une autre extension naturelle des problèmes de décision. Si l'on sait répondre à la question consistant à chercher s'il existe une solution de taille k possédant une propriété particulière, une autre question survient. Si l'on sait qu'il existe au moins une solution, alors combien de solutions de taille k y a-t-il réellement au total ? Bien entendu, le fait de savoir répondre au problème de comptage permet de répondre au problème de décision.

Pour chaque problème de décision, il existe un problème de comptage associé qui demande à compter le nombre de solutions pour chaque entrée.

Si un problème de décision est NP-complet, alors le problème de comptage qui lui est associé est $\#P$-complet. Cependant, il existe des problèmes de décision polynomiaux pour lesquels le problème de comptage associé est $\#P$-complet, par exemple : couplage dans un graphe bipartie. Ces notions sont expliquées plus amplement dans le document de L.G. Valiant [92].

Les problèmes d'énumération

Un problème d'énumération consiste à générer une liste de toutes les solutions possibles pour l'entrée du problème. Ainsi, les problèmes d'énumération sont une extension des problèmes de décision, d'optimisation et de comptage.

Une fois la liste des solutions connue, on peut facilement compter le nombre d'éléments sur cette liste, résolvant ainsi le problème de comptage. On peut aussi trouver quelle est la solution de plus petite ou de plus grande taille, résolvant ainsi le problème d'optimisation. Enfin, évidemment, on peut déterminer si une solution existe ou pas. D'une certaine manière, on pourrait considérer que le problème d'énumération est plus fort que les autres problèmes.

Même si ce n'est pas le sujet de cette thèse, il existe une distinction supplémentaire et très intéressante dans les problèmes d'énumération, qui consiste à estimer s'il est possible ou non de mesurer le temps d'exécution en fonction, non plus de la taille de l'instance d'entrée, mais de la taille du résultat attendu en sortie. La classe de complexité des problèmes dont le temps d'exécution peut être donné en prenant en compte le nombre d'éléments dans le résultat est appelée *sensible à la sortie* (*output sensitive* en anglais).

On peut aussi chercher à ce que les éléments de la solution soient découverts à intervalles réguliers, l'algorithme correspondant étant alors donné avec une fonction estimant le temps entre la découverte de deux résultats successifs. On parlera alors d'algorithmes à *délai polynomial* (*polynomial delay* en anglais).

Ces deux dernières notions peuvent être retrouvées dans un papier de D.S. Johnson, C.H. Papadimitriou et M. Yannakakis [56].

1.2 Notations et définitions en complexité

La valeur d'un algorithme se mesure en fonction de plusieurs critères. Tout d'abord, il faut s'assurer que l'algorithme arrive à résoudre correctement le problème qui lui correspond, et cela pour toutes les entrées susceptibles de lui être attribuées. Ensuite, et c'est surtout cela qui nous intéresse, on jaugera le temps nécessaire à l'algorithme pour trouver la solution du problème.

En fait, ce n'est pas tellement le temps réel que l'on mesure, mais plutôt le nombre d'opérations nécessaires à notre algorithme pour venir à bout du calcul dans le pire des cas. Le nombre d'instructions à exécuter permettant néanmoins de déterminer le temps nécessaire à l'exécution en fonction de la machine, on parlera quand même de temps d'exécution.

Les temps d'exécution sont donnés sous la forme d'une fonction $f(n)$ dont

le paramètre n est un entier permettant de représenter la taille de l'instance du problème. Dans cette thèse, la taille de l'instance d'un problème sera généralement le nombre de sommets du graphe sur lequel l'algorithme doit s'exécuter.

Les temps d'exécution au pire des cas que nous calculons sont théoriques, et sont donnés sous forme de bornes asymptotiques. Il y a globalement trois formes de notation asymptotique qui peuvent être utilisées pour donner un tel temps d'exécution. On note par $T(n)$ le temps d'exécution au pire des cas d'un algorithme sur une entrée de taille n.

- O : La notation O est celle qui sera la plus utilisée dans cette thèse, et qui est la plus utilisé en général. Si le temps d'exécution (au pire des cas) d'un algorithme est $O(f(n))$, cela signifie qu'il existe deux entiers c et n_0 tels que quelque soit $n \geq n_0$, alors le temps d'exécution $T(n)$ sera inférieur à la valeur donnée par la fonction $c \cdot f(n)$.
- Ω : la notation Ω est utile lorsqu'il s'agit de donner une borne inférieure à un temps d'exécution. Si le temps d'exécution est en $\Omega(f(n))$, cela signifie qu'il existe deux constantes c et n_0 telles que quelque soit $n \geq n_0$, alors le temps d'exécution au pire des cas $T(n)$ sera supérieur à la valeur donnée par $c \cdot f(n)$.
- Θ : Si le temps d'exécution $T(n)$ est en $\Theta(f(n))$, cela signifie que $T(n)$ est en $O(f(n))$ et en $\Omega(f(n))$. On peut dire que les fonctions $T(n)$ et $f(n)$ sont asymptotiquement équivalentes. Cette notation ne sera pas utile dans cette thèse.

Ces notations sont définies plus précisément dans les ouvrages de T.H. Cormen, C.E. Leiserson, R.L. Rivest et C. Stein [18], ou de J. Kleinberg et É. Tardos [60].

Pour analyser les algorithmes exacts et exponentiels que nous utiliserons dans cette thèse, nous aurons besoin d'un complément pour les notations que nous venons de définir. Nous utiliserons donc la notation O^* définie par G. Woeginger [98] pour ignorer les termes bornés par un polynôme en fonction de la taille de l'entrée. De cette manière, si le temps d'exécution $T(n)$ de notre algorithme est de la forme $poly(n) \cdot n^t$, avec $poly(n)$ un polynôme, nous noterons simplement que $T(n)$ est en temps $O^*(n^t)$. Dans certains cas, les valeurs que nous donnerons seront arrondies, cet arrondissement nous permet d'utiliser la notation O classique plutôt que la notation O^*, comme expliqué dans le livre de F. Fomin et D. Kratsch [38].

1.3 Les graphes

Dans cette thèse, nous travaillerons essentiellement sur des problèmes de graphe. Les problèmes auxquels nous nous sommes intéressés consistent principalement à rechercher un ensemble de sommets avec une propriété particulière dans le graphe d'entrée, ou à partitionner le graphe d'entrée en ensembles distincts ayant certaines propriétés.

1.3.1 Définitions et notions de bases

Commençons déjà par définir ce qu'est un graphe. Un graphe $G = (V, E)$ est un couple composé d'un ensemble de sommets $V(G)$ et d'un ensemble d'arêtes $E(G)$. Lorsqu'il n'y a aucune ambiguïté sur le graphe auquel correspond un ensemble de sommets $V(G)$ (respectivement d'arêtes $E(G)$), alors nous noterons tout simplement V (respectivement E).

Le nombre de sommets de l'ensemble $V(G)$ sera désigné par la lettre n, et le nombre d'arêtes dans l'ensemble $E(G)$ par la lettre m. Les graphes que nous considérons sont finis, c'est-à-dire que les ensembles V et E sont des ensembles finis. De plus, nous ne considérons que des graphes simples, c'est-à-dire sans boucles ni arêtes multiples.

Les sommets $v \in V$ d'un graphe sont les éléments de base du graphe. La plupart du temps, on définira la taille d'un graphe comme étant le nombre de ses sommets. Dans la plupart des problèmes que nous étudierons, le nombre de sommets du graphe définira la taille de l'instance du problème, et les temps d'exécutions seront donnés en fonction du nombre de sommets dans l'instance étudiée.

Une arête uv est un couple de sommets $u \in V$ et $v \in V$ du graphe G. Si cette arête existe dans l'ensemble E, alors on dit que les sommets u et v sont *adjacents*, on peut aussi dire que u et v sont *voisins*. Si pour deux sommets u' et v', l'arête $u'v'$ n'existe pas dans l'ensemble E, alors on dit que ces deux sommets sont *non-adjacents*.

On appelle *voisinage ouvert* du sommet $v \in V(G)$ l'ensemble $N_G(v) = \{x \in V(G) | xv \in E(G)\}$. On appelle *voisinage fermé* du sommet $v \in V(G)$ l'ensemble $N_G[v] = \{v\} \cup \{x \in V(G) | xv \in E(G)\}$. On peut aussi remarquer que $N_G[v] = N_G(v) \cup \{v\}$.

Pour un ensemble de sommets $S \subseteq V(G)$, nous définissons aussi le *voisinage ouvert* de l'ensemble S comme étant $N_G(S) = \bigcup_{v \in S} N_G(v) \backslash S$ et le *voisinage fermé* de l'ensemble S comme étant $N_G[S] = N_G(S) \cup S$.

Le nombre de sommets dans le voisinage ouvert d'un sommet v est appelé le degré $d_G(v) = |N_G(v)|$. Si un sommet est de degré $d_G(v) = 0$, cela signifie

qu'il n'a aucun voisin dans le graphe G, on dit que le sommet est *isolé*. Si chaque sommet est voisin de tous les autres sommets du graphe, alors le degré de chaque sommet est égal à $n-1$ et on dit que le graphe est complet.

On appelle degré minimum $\delta(G) = min\{d_G(v)|v \in V(G)\}$ le degré du sommet ayant le plus petit degré dans le graphe G. De même, on appelle degré maximum $\Delta(G) = max\{d_G(v)|v \in V(G)\}$ le degré du sommet ayant le plus grand degré dans le graphe G.

Pour toutes ces notations, s'il n'y a pas d'ambiguïté sur le graphe qui correspond aux indications que l'on souhaite donner, alors on supprimera l'indice G de la notation.

On appelle *chemin* dans le graphe $G = (V, E)$ une séquence de sommets $(v_1, \cdots, v_k)$ telle que $v_i v_{i+1} \in E$ pour $1 \leq i < k$. La longueur d'un chemin est un entier défini par le nombre d'arêtes présentes dans ce chemin, c'est-à-dire $k-1$.

Étant donné deux sommets u et v, la distance $d(u, v)$ entre ces deux sommets est égale à la longueur du plus court chemin existant entre ces deux sommets.

Si $v_1 v_k \in E$ alors la séquence de sommets définit un *cycle*. La *taille* d'un cycle est égale au nombre de ses arêtes, qui est lui-même égal au nombre de ses sommets, c'est-à-dire k.

On dit qu'un graphe G est connexe si pour toutes paires de sommets $u, v \in V$ dans le graphe, il existe un chemin entre les sommets u et v.

On dit que le graphe $H = (V', E')$ est un *sous-graphe* du graphe $G = (V, E)$ si $V' \subseteq V$ et $E' \subseteq E$. On dit que le graphe H est un sous-graphe *induit* du graphe G si H est un sous-graphe de G et que quelque soit $u, v \in V'$, alors $uv \in E \Rightarrow uv \in E'$.

On dit qu'un ensemble S est *maximal*, ou *maximal par inclusion* pour une certaine propriété Π si S possède la propriété Π, et il n'existe aucun ensemble S' tel que $S \subset S'$ et que S' possède la propriété Π.

De même, un ensemble S est *minimal*, ou *minimal par inclusion* pour une propriété Π si S possède la propriété Π, et il n'existe aucun ensemble S' tel que $S' \subset S$ et S' possède la propriété Π.

On appelle *composante connexe* un sous-graphe connexe maximal dans G. On appelle *graphe complémentaire* de $G = (V, E)$ le graphe noté $\overline{G}$ tel que $\overline{G} = (V, \{uv|uv \notin E(G)\}, u \neq v)$.

Soit $S \subseteq V$. Si le sous-graphe $G[S]$ induit par S est un graphe complet, alors on dit que S est une clique. Si le sous-graphe $\overline{G[S]}$, le graphe complémentaire de $G[S]$, est un graphe complet, alors $G[S]$ est composé uniquement de sommets isolés et on dit que S est un ensemble stable dans G.

On dit que deux graphes $G = (V, E)$ et $G' = (V', E')$ sont isomorphes s'il existe une bijection $b : V \to V'$ entre les sommets de V et V' telle que $uv \in E \iff b(u)b(v) \in E'$.

Ces notions de base pourront le cas échéant être précisées à nouveaux dans les chapitres où elles sont utilisées. D'autres notions moins générales pourront aussi être présentées aux moments où elles seront nécessaires. Dans l'ensemble, plusieurs ouvrages regroupent les notions et/ou les notations qui nous seront indispensables dans cette thèse. Nous invitons notamment le lecteur à consulter les ouvrages de R. Diestel [31], de M.C. Golumbic [50], et de D.B. West [96].

1.3.2 Les classes de graphe particulières

Au cours de nos travaux, nous avons beaucoup utilisé les classes de graphe particulières. Afin de pouvoir présenter nos recherches effectuées, il convient d'abord de définir ce qu'est une classe de graphe, et de décrire un certain nombre de celles-ci, celles qui seront les plus importantes pour nous.

Une classe de graphe, que nous pourrons aussi par moment appeler une famille de graphes, est un ensemble de graphes défini par une ou plusieurs propriétés sur la structure du graphe. Tout graphe G possédant la ou les propriétés requises appartient de fait à la classe correspondante. *A contrario*, tout graphe G pour lequel au moins une propriété définissant la classe de graphe ne correspondrait pas sera de fait hors de cette classe de graphe.

Dans cette thèse, les propriétés qui nous serviront à définir la plupart des classes de graphes pourront être décrites sous la forme de sous-graphes induits interdits, mais nous travaillerons aussi sur certaines des classes les plus utilisées dans la littérature. Nous ferons souvent référence au livre de A. Brandstädt, V.B. Le et J.P. Spinrad [15] et de M.C. Golumbic [50] pour ce qui concerne les classes de graphe, leurs propriétés ou leurs particularités. On notera aussi l'utilité du site internet *graphclasses.org*.

Voyons maintenant quelques unes des plus importantes classes de graphes que nous utiliserons dans cette thèse.

Les graphes biparties et co-biparties

Un graphe $G = (V, E)$ est un graphe *bipartie* si et seulement si l'ensemble de ses sommets V peut être partitionné en deux sous-ensembles A et B tels que chacun de ces deux sous-ensembles soit un ensemble stable dans G. Un tel graphe pourra alors être décrit comme le graphe $G = (A, B, E)$.

On appellera *co-bipartie* le graphe complémentaire d'un graphe bipartie, qui sera donc constitué d'une partition en deux cliques.

Si toutes les arêtes possibles entre les deux ensembles A et B sont présentes dans le graphe $G = (A, B, E)$, c'est-à-dire si $E = \{uv | u \in A \wedge v \in B\}$, alors le graphe est bipartie complet. Dans certains cas, comme par exemple si le graphe bipartie complet H est un sous-graphe induit d'un graphe G, on pourrait appeler H une *biclique* de G.

Les graphes splits

Un graphe $G = (V, E)$ est un graphe *split* si et seulement si l'ensemble de ses sommets V peut être partitionné en deux sous-ensembles C et S tels que C est une clique, et S est un ensemble stable. On pourra alors décrire ce graphe comme étant le graphe $G = (C, S, E)$.

Dans un graphe split, la partition en deux ensembles C et S n'est pas forcément unique.

Dans les chapitres utilisant fortement les propriétés des classes de graphe, ou les sous-graphes interdits, nous utiliserons le théorème disant que les graphes splits sont exactement les graphes sans $2K_2$, sans C_4 et sans C_5.

Les co-graphes

Définissons deux opérations sur les graphes permettant de construire un graphe à partir de deux graphes distincts :

- $G = G_1 \bowtie G_2 : V(G) = V(G_1) \cup V(G_2)$ et $E(G) = E(G_1) \cup E(G_2) \cup \{xy | x \in V(G_1), y \in V(G_2)\}$. Le graphe G est une *jonction* entre les graphes G_1 et G_2.
- $G = G_1 \uplus G_2 : V(G) = V(G_1) \cup V(G_2)$ et $E(G) = E(G_1) \cup E(G_2)$. Le graphe G n'est pas connexe. Le graphe G est une *union disjointe* entre les graphes G_1 et G_2.

Un graphe $G = (V, E)$ est un co-graphe si et seulement si il peut être construit à partir de sommets isolés en utilisant uniquement les opérations de jonction ou d'union disjointe. Cette succession d'opérations peut être encodée sous la forme d'un arbre que nous appellerons l'arbre de construction du co-graphe, ou le *co-tree*.

Dans les chapitres utilisant fortement les propriétés des classes de graphe, ou les sous-graphes interdits, nous utilisons le théorème disant que les co-graphes sont exactement les graphes sans P_4.

Les graphes cordaux

Un graphe $G = (V, E)$ est un graphe cordal si et seulement si tout cycle de longueur supérieure ou égale à 4 dans le graphe a une corde. Une corde est une arête qui relie deux sommets non consécutifs dans le cycle. Cela signifie que le cycle sans corde le plus grand possible est de taille 3.

Un sommet est dit simplicial si son voisinage est une clique. Un graphe cordal dispose toujours d'au moins 1 sommet simplicial.

Les graphes cubiques

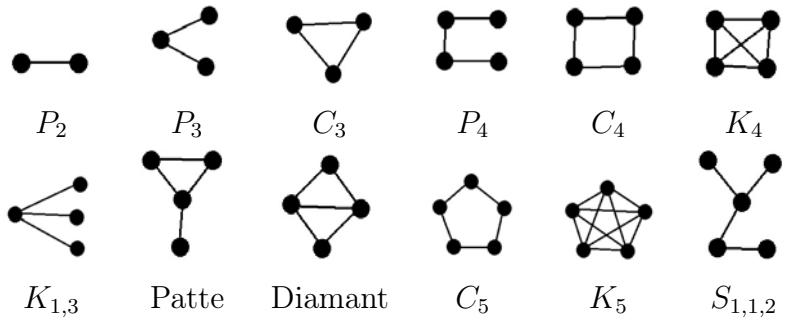
Un graphe $G = (V, E)$ est cubique si et seulement si pour chaque sommet $v \in V$ le degré $d(v) = 3$. Il s'agit ici d'une propriété sur le degré des sommets.

Quelques graphes spéciaux

Certains graphes utiles, à la construction généralement très simple mais dont la structure revient souvent ou présente un intérêt particulier, disposent d'une désignation unique pour pouvoir les décrire et les distinguer rapidement.

Nous allons lister ici un certain nombre de ces graphes, ou types de graphe, qui nous servirons au cour de cette thèse.

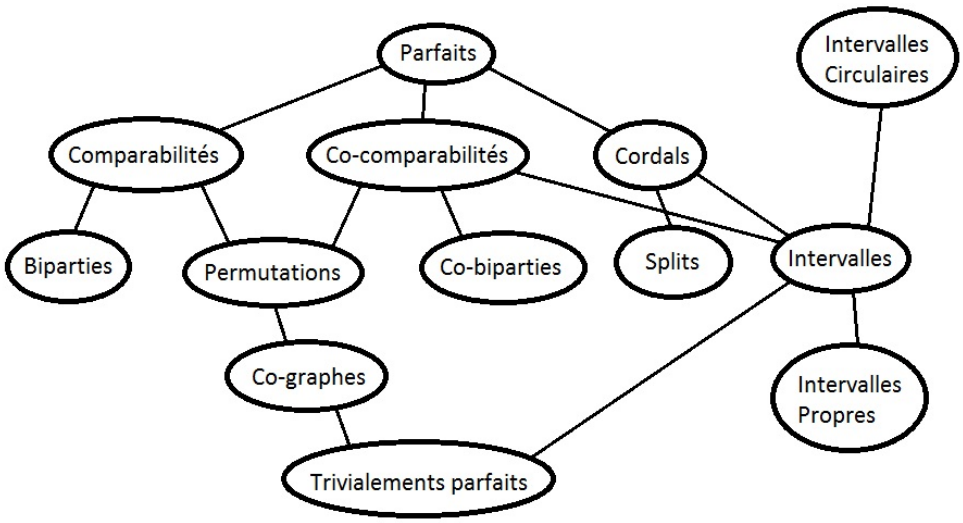
- Un graphe dont la totalité des sommets forme une clique de n sommets est un graphe complet noté K_n.
- Un graphe $G = (V, E)$ avec $V = \{v_1, \cdots, v_n\}$ et $E = \{v_i v_{i+1} | i \in \{1, \cdots, n-1\}\}$ est appelé un P_n.
- Un cycle de n sommets est appelé un C_n.
- Un graphe bipartie complet qui dispose de k sommets dans la partie A et de l sommets dans la partie B est appelé un $K_{k,l}$. Ainsi, la *griffe* (*claw* en anglais), qui est un cas particulier de graphe bipartie complet avec 1 sommet dans la partie A et 3 sommets dans la partie B sera désigné par $K_{1,3}$.
- Un graphe de 4 sommets constitué d'un triangle dont l'un des sommets a un voisin pendant est appelé un graphe *patte* (*paw* en anglais).
- Un graphe de 4 sommets représentant deux triangles avec une arête en commun est un graphe *diamant*.
- Un graphe dont l'un des sommet serait le point de départ de trois chemins de longueurs respectives i, j et k sera appelé un $S_{i,j,k}$. Ainsi, le graphe *chaise* (*fork* en anglais) est un $S_{1,1,2}$.



Il existe bien évidemment de nombreux autres graphes possédant une désignation propre dans la littérature, mais nous nous contenterons de ceux que nous pourrons retrouver au fil des chapitres de ce document.

Les relations d'inclusions entre les classes de graphe

Il existe entre les différentes classes de graphe, des relations d'inclusion. Cette relation d'inclusion définit en fait une relation d'ordre partielle sur l'ensemble des classes de graphe. On peut par exemple constater que si un graphe G est un graphe split, alors le cycle induit de plus grande taille dont il dispose est de longueur 3, c'est donc un graphe cordal. Des précisions supplémentaires seront données en ce qui concerne cette relation sur les classes de graphe définies par des sous-graphes interdits dans le chapitre 6.1.



Quelques relation d'inclusion parmi des graphes de cette thèse

Toutes les informations ou les précisions supplémentaires que le lecteur

pourrait vouloir obtenir peuvent être trouvées dans le livre de A. Brandstädt, V.B. Le et J.P. Spinrad [15].

1.3.3 Quelques problèmes principaux

Ensemble stable

Parmi les problèmes de graphe les plus connus figure celui de l'ensemble stable. Un ensemble stable est un sous-ensemble de sommets du graphe tel qu'il n'y a pas deux sommets dans cet ensemble qui sont voisins.

Il est facile de vérifier si un ensemble est stable ou non, et il est facile de voir qu'il existe toujours un ensemble stable dans un graphe (un ensemble de cardinalité 1 sera toujours stable).

Le problème devient NP-complet si l'on cherche un ensemble stable d'une taille précise, ou tout simplement un ensemble stable avec la plus grande taille possible.

Problème INDEPENDENT SET
Entrée : Graphe $G = (V, E)$
Question : Quel est la taille du plus grand ensemble stable possible dans le graphe G ?

Une solution triviale consiste à essayer pour tous les sous-ensembles S possibles de V, si S est un ensemble stable et s'il est le plus grand vu jusque-là. Cette solution triviale donne naturellement un algorithme en temps $O^*(2^n)$ où n représente le nombre de sommets dans V.

Ce problème a été souvent étudié depuis une quarantaine d'années, et l'on peut trouver des algorithmes beaucoup plus rapides que l'algorithme trivial. L'algorithme publié en 2010 par N. Bourgeois et al. [13] permet de trouver un ensemble stable de taille maximum en temps $O^*(1.2127^n)$ tout en utilisant un espace polynomial.

Coloration, nombre chromatique

Une généralisation courante du problème d'ensemble stable est la coloration de graphe. Plutôt que de chercher un ensemble stable de taille maximum, on cherche à partitionner le graphe en un groupe d'ensembles stables.

La question qui se pose naturellement est de combien d'ensembles avons-nous besoin pour pouvoir partitionner le graphe de façon à ce que chaque ensemble soit stable ?

La plupart du temps, ce problème est présenté sous forme d'attribution de couleur dans le graphe. Il faut que chaque sommet ait une couleur de façon

à ce que deux sommets voisins soient de couleurs différentes. Est-il possible de colorier le graphe avec moins de k couleurs ?

Problème CHROMATIC NUMBER
Entrée : Graphe $G = (V, E)$
Question : Quel est le plus petit entier k tel que l'on puisse trouver une coloration propre du graphe G avec moins de k couleurs ?

Ce problème est intéressant aussi lorsque l'entier k est fixé comme un paramètre de l'entrée :

Problème k-CHROMATIC NUMBER
Entrée : Graphe $G = (V, E)$, entier k
Question : Est-il possible de trouver une coloration propre du graphe G avec moins de k couleurs ?

Le problème k-CHROMATIC NUMBER est NP-complet, des que $k \geq 3$ (si $k = 2$, cela revient à vérifier si le graphe est bipartie, il existe alors un algorithme en temps linéaire).

Une solution triviale pour le problème k-CHROMATIC NUMBER consiste à énumérer toutes les colorations possibles, c'est-à-dire essayer chaque couleur pour chaque sommet, mais cette approche donne un algorithme en $O^*(k^n)$. Pour le problème CHROMATIQUE NUMBER, cette énumération triviale s'exécute en $O^*(2^{n \cdot \log(n)})$.

En 2006, A. Björklund et T. Husfeldt [10], en même temps que M. Koivisto [61], ont réussi à obtenir un algorithme qui s'exécute en $O^*(2^n)$ pour résoudre le problème du NOMBRE CHROMATIQUE grâce à la méthode d'inclusion-exclusion.

Ensemble dominant

L'un des problèmes les plus importants à l'heure actuelle est le problème d'ensemble dominant. On cherche un ensemble de sommets tel que tous les sommets du graphe aient au moins un de leurs voisins dans cet ensemble, ou en font partie.

Il existe bien entendu plusieurs versions de ce problème, et plusieurs façon de le décliner, selon que l'on souhaite avoir ou non des propriété supplémentaires pour l'ensemble dominant, comme de la connexité, ou le fait de devoir dominer aussi les éléments de l'ensemble.

Tout comme pour l'ensemble stable, il est facile de vérifier si un ensemble est dominant ou pas, et on peut facilement voir qu'il existe toujours au moins un ensemble dominant dans un graphe. Le problème est NP-complet si l'on cherche pour un entier k donné un ensemble dominant de taille au plus k.

La version optimisation consiste à chercher un ensemble dominant de la plus petite taille possible.

Problème Dominating Set
Entrée : Graphe $G = (V, E)$
Question : Quelle est la taille du plus petit ensemble dominant dans le graphe G ?

Une solution triviale pour ce problème, qui donne une fois encore un algorithme en temps $O^*(2^n)$, est d'essayer tous les sous-ensembles possibles de V afin de tester s'ils sont dominants, et de comparer leur taille aux autres ensembles dominants.

Ce problème est très étudié et a été au cours de cette dernière décennie l'objet de plusieurs thèses. Notamment celle de J.M.M. Van Rooij [93] et de M. Liedloff [69]. La résolution de ce problème a été récemment améliorée par Y. Iwata [55] qui, grâce à une variante de la méthode *measure and conquer*, obtient un algorithme qui s'exécute en $O^*(1.4864^n)$ avec un espace polynomial.

1.4 Les algorithmes exacts et à temps d'exécution exponentiel

Lorsque l'on est confronté à un problème difficile, soit NP-complet, soit NP-difficile, soit #P-complet, etc., on ne sait pas construire d'algorithme qui puisse le résoudre de façon exacte et en temps polynomial. Sinon, cela signifierait que $P = NP$.

Dans ce cas, plusieurs approches sont possibles afin de pouvoir traiter le problème tout de même :

- Ne prendre en compte que les entrées de taille modérée.
- Chercher une solution raisonnablement *petite*.
- Ne prendre en compte que des entrées disposant d'une propriété Π particulière (par exemple, une classe de graphe).
- Utiliser des heuristiques.
- Utiliser des algorithmes randomisés.
- Utiliser des algorithmes d'approximation.
- Utiliser des algorithmes à paramètre fixe.
- Utiliser des algorithmes exacts et à temps d'exécution exponentiel.

Pour les algorithmes randomisés, nous conseillons au lecteur le livre de R. Motawni et P. Ragawan [77]. En ce qui concerne les algorithmes d'approximation, nous conseillerons le livre de D.P. Williamson et D.B. Shmoys [97].

Enfin, pour les algorithmes à paramètre fixe, on citera les livres de R. Niedermeier [79] et de J. Flum et M. Grohe [37].

Pour les algorithmes exacts et exponentiels, nous conseillons au lecteur de se référer au livre de F. Fomin et D. Kratsch [38].

Dans cette thèse, nous avons choisi de sacrifier le temps d'exécution de façon à obtenir quoi qu'il en coûte, la solution optimale au problème donné. Certes, c'est bien joli de dire cela mais il serait bien tout de même de ne pas trop sacrifier le temps. Même à temps d'exécution exponentiel au rapport de l'entrée, notre but reste d'essayer d'obtenir un algorithme le plus rapide possible, au moins disons exécutable a l'échelle d'une vie humaine. Surtout qu'en raison de l'exponentiel, un progrès qui peut paraître infime se traduirait en pratique par un gain de temps colossal.

Mais alors que faire ? Et comment le faire ? Il existe plusieurs méthodes pour la construction et l'analyse des algorithmes exacts à temps d'exécution exponentiel. Les trois méthodes principales sont la méthode de programmation dynamique, la méthode d'inclusion-exclusion, et enfin la méthode de branchement et réduction.

Bien entendu, chaque méthode a ses avantages et ses inconvénients, et pour chaque méthode, il existe des astuces et des évolutions permettant d'essayer de contourner quelque peu ses faiblesses.

Il est aussi possible de combiner les différentes méthodes, comme ont pu le faire F. Fomin et A. Stepanov [44] qui ont combiné la méthode de branchement et réduction avec celle de programmation dynamique. Ou encore J. Van Rooij et al. [94] qui ont combiné la méthode d'inclusion-exclusion avec celle de branchement et réduction.

Enfin, si ces méthodes sont les plus classiques, rien n'empêche, si on y arrive, d'innover et d'utiliser une autre méthode.

Nous allons maintenant présenter les idées générales de ces trois méthodes, ainsi que certaines propriétés typiques des algorithmes obtenus par ces méthodes. Pour des informations plus détaillées, sachez qu'il existe un chapitre dédié à chacune de ces méthodes dans le livre de F. Fomin et D. Kratsch [38].

1.4.1 La programmation dynamique

On connait la programmation dynamique pour construire des algorithmes polynomiaux, un tel algorithme est basé sur une ou plusieurs récurrences et applique une approche ascendante. Le principe est simple : il est plus facile de résoudre un problème d'une certaine taille, si on connait déjà la solution

de tous les sous-problèmes de taille inférieure. (Voir Cormen [18] pour plus de détails).

Ce principe a été utilisé pour la première fois dans les années 1950 par R. Bellman, pour optimiser des sommes de fonctions monotones croissantes [6]. R. Bellman se serait inspiré de la méthode classique *diviser pour régner* dont il se différencie par le fait qu'au lieu de définir, à partir d'un problème donné, quels sont les sous-problèmes à résoudre, il commence par résoudre localement des problèmes de petite taille afin de pouvoir utiliser les résultats plus tard.

Le principal avantage de cette approche est que l'on évite, le cas échéant, d'avoir à rechercher plusieurs fois la solution à un même problème. L'inconvénient majeur étant que cette méthode est gourmande en espace mémoire, puisqu'il faut enregistrer et conserver les résultats de tous les sous-problèmes déjà résolus.

Lorsque l'on utilise la programmation dynamique pour construire un algorithme à temps d'exécution exponentiel, le principe reste le même. On cherche à résoudre localement les problèmes de petites tailles pour pouvoir utiliser les résultats plus tard. Seulement, si le problème est difficile, alors le nombres de sous-problèmes à résoudre est lui-même typiquement exponentiel.

Par exemple, l'algorithme de M. Held et R.M. Karp [52] permet de résoudre le problème du voyageur de commerce en utilisant la méthode de programmation dynamique avec un temps d'exécution en $O^*(2^n)$.

Comme on doit stocker tous les résultats obtenus, alors l'espace mémoire nécessaire à l'exécution d'un algorithme de programmation dynamique sera généralement lui aussi exponentiel.

Malgré cet inconvénient majeur, la programmation dynamique reste un outil très puissant, qui peut en plus être associé à des structures particulières, comme des décompositions arborescentes de graphes, afin de donner des résultats très intéressants.

1.4.2 L'inclusion-exclusion

Le principe d'inclusion-exclusion est un principe mathématique, ou plus précisément, de combinatoire. Il permet de compter le nombre d'éléments appartenant à une réunion d'ensembles finis, à partir du nombre d'éléments de chaque ensemble et de leurs intersections respectives. On attribue généralement ce principe à Abraham de Moivre, un mathématicien du 17^e siècle.

En informatique, ce principe peut être utiliser pour compter le nombre de solutions d'un problème à condition que l'on réussisse à formuler ce dernier sous forme de problème de réunion d'ensembles. Soit un univers $\mathcal{U}$, et une

liste de n ensembles d'éléments de cet univers. Ces ensembles, tous différents, seront noté A_i, pour $i \in \{1, \cdots, n\}$. On note par $\overline{A_i}$ le complémentaire de l'ensemble A_i, c'est-à-dire l'ensemble des éléments de $\mathcal{U}$ qui ne sont pas dans l'ensemble A_i. Le principe fondamental repose sur la formule suivante :

$$|\bigcap_{i\in\{1,\ldots,n\}} A_i| = \sum_{X\subseteq\{1,\ldots,n\}} (-1)^{|X|} |\bigcap_{j\in X} \overline{A_j}|$$

On peut retrouver cette formule fondamentale de combinatoire, par exemple dans un document de J. Nederlof [78]. La partie de gauche représente le résultat que l'on recherche, et si dans la partie de droite on arrive à montrer que la valeur de $|\bigcap_{j\in X} \overline{A_j}|$ peut être calculée en temps polynomial, et sachant que tous les $X \subseteq \{1, ..., n\}$, c'est-à-dire tous les groupes d'ensembles possibles à faire avec les ensembles A_i de l'univers $\mathcal{U}$ doivent être pris en compte, et donc qu'il y a en fait 2^n groupes X différents, contenant de 1 à n ensembles, alors on a automatiquement un algorithme qui s'exécute en temps $O^*(2^n)$ pour trouver la solution de notre problème.

Bien que nous en ayons déjà parlé dans 1.3.3, l'un des plus grands résultats de la méthode inclusion-exclusion est l'algorithme en $O^*(2^n)$ de A. Björklund et T. Husfeldt [10], ou encore celui de M. Koivisto [61] pour trouver le nombre chromatique d'un graphe. Les trois auteurs ont d'ailleurs uni leurs travaux lors de la publication journal [11]. Mais on peut aussi citer d'autres problèmes comme Steiner Tree [78], ou encore Hamiltonian Path dans un graphe orienté [59] que cette méthode permet d'appréhender.

1.4.3 La méthode branchement et réduction

La dernière méthode dont nous allons parler ici est celle qui semble a priori la plus intuitive, la méthode de branchement et réduction. Cette méthode est celle que nous utiliserons le plus dans cette thèse, notre description sera donc plus détaillée. L'idée est simple : il s'agit de prendre un des éléments du problème et de lui imposer successivement toutes les valeurs qu'il pourrait avoir dans une solution. Cela permet alors de déterminer un certain nombre de sous-problèmes, un par valeur imposée, dont il nous suffira de comparer les résultats lorsqu'ils seront eux-même résolus.

Dans des problèmes de graphe, comme Ensemble Stable ou Ensemble Dominant, cela se traduit par choisir si un sommet sera ou non dans l'ensemble que l'on recherche. En général, faire un tel choix a un impact immédiat sur le reste du graphe. Choisir qu'un sommet sera dans un ensemble stable nous interdit immédiatement tous les voisins de ce sommet dans l'ensemble stable. Ainsi nous n'éliminons pas forcément qu'un seul sommet en faisant notre choix.

Pour chaque étape, selon que le nombre de sous-problèmes créés à cette étape est supérieur à 1, ou égal à 1, on parlera de branchement ou de réduction. C'est d'ailleurs de là que vient le nom de la méthode. Si jamais il advenait à une étape, que l'un des éléments ne puisse plus prendre de valeur du tout, on parlera alors de *feuille négative*.

Les branchements. Ils sont le cœur de la méthode, et c'est aussi à partir d'eux que l'on estimera le temps d'exécution de l'algorithme. La valeur d'un branchement particulier sera mesurée en fonction du nombre de sous-problèmes créés, et du nombre d'éléments pouvant être éliminés dans chaque sous-problème obtenu par ce branchement. On parlera alors d'un *vecteur de branchement*, qui est simplement une liste du nombre de sommets éliminés pour chaque branche créée.

Par exemple, si un branchement consiste à choisir la valeur d'un élément entre 2 possibilités, et que l'on ne gagne (cela signifie élimine) qu'un seul élément pour chacune, le vecteur de branchement correspondant sera le vecteur $(1, 1)$. La valeur qui correspond à un vecteur de branchement est appelée le *nombre de branchement*. Dans notre exemple, le nombre de branchement du vecteur $(1, 1)$ est 2. Le temps d'exécution d'un algorithme qui n'utiliserait que ce branchement serait $O^*(2^n)$.

Bien entendu, un algorithme contient en général plus d'une règle de branchement, et l'on choisira alors le nombre de branchement le plus élevé de tous les branchements pour établir la borne supérieure du temps d'exécution de l'algorithme.

Les réductions. Lorsqu'il y a, dans un sous-problème, un élément ne pouvant prendre qu'une seule valeur, on peut effectuer une réduction. Cela revient tout simplement à dire que si cette valeur est la dernière possible pour l'élément, alors c'est celle qu'il aura dans une éventuelle solution. Cette conclusion est évidente et semble presque être une "vérité de La Palisse". Pourtant, il sera nécessaire de vérifier à chaque étape si une telle situation survient.

Pour notre analyse, il est nécessaire qu'une réduction, ou une succession quelconque de réductions, s'effectue en temps polynomial.

Les feuilles négatives. Tout comme il est nécessaire de vérifier s'il existe un élément n'ayant plus qu'une valeur possible, il est impératif de vérifier si, au cours des branchements précédents, nous n'aurions pas créé un élément qui ne puisse plus prendre aucune valeur dans une éventuelle solution. Dans ce cas, il devient impossible de créer un ou plusieurs sous-problèmes réalisables,

et l'on est dans une feuille négative. L'algorithme n'a alors d'autre possibilité que de chercher la solution dans une autre branche. Si toutes les feuilles sont négatives, alors cela signifie tout simplement qu'il n'existe pas de solution à l'instance du problème sur lequel l'algorithme s'exécute.

Les feuilles négatives jouent un rôle important dans les algorithmes d'énumération.

L'arbre de recherche. À plusieurs reprises déjà, nous avons parlé de feuilles, ou de branches. Dans la technique de branchement et réduction, l'exécution de l'algorithme peut être vue comme un arbre enraciné.

En ce qui concerne l'exécution d'un l'algorithme, l'instance initiale du problème peut être représentée par la racine de l'arbre.

Chaque sous-problème sera représenté par un nœud dans l'arbre, et pour chaque branchement effectué sur un sous-problème, les sous-problèmes obtenus seront représentés par les fils de ce nœud dans l'arbre. Les solutions seront représentées dans les feuilles de l'arbre.

Dans la plupart des algorithmes de branchement et réduction, chaque branchement produit des sous-problèmes qui n'ont jamais de solutions communes, et les solutions dans les feuilles sont donc différentes deux à deux.

Le calcul du temps d'exécution. Calculer le temps d'exécution d'un algorithme de branchement et réduction revient aussi à faire une estimation du nombre de nœuds dans l'arbre de recherche qui correspondra à cette exécution.

Chaque branchement dans l'arbre dépend de l'une des règles de récurrence définies par l'algorithme et correspond à un vecteur de branchement. Pour chaque règle de récurrence, il faut alors calculer le nombre de branchements correspondants, le plus grand nombre de branchements obtenus sera alors utilisé pour définir le temps d'exécution de l'algorithme.

On désigne par $T(n)$ le temps requis pour résoudre un problème de taille n. Le vecteur de branchement $(t_1, t_2, \cdots, t_k)$ correspondant à une règle de récurrence nous permet de connaître la taille de chacun des sous-problèmes (et le nombre de sous-problèmes) générés par cette récurrence. On peut alors écrire la récurrence sous la forme suivante :

$$T(n) \leq T(n - t_1) + T(n - t_2) + \cdots + T(n - t_k)$$

Cette récurrence nous permet alors de définir une équation correspondante de la forme :

$$\alpha^n \leq \alpha^{n-t_1} + \alpha^{n-t_2} + \cdots + \alpha^{n-t_k}$$

On cherche alors la valeur de la plus grande racine positive de cette équation, que l'on peut par exemple trouver à l'aide de la méthode de Newton.

Dans cette thèse, la plupart des algorithmes utilisent la méthode de branchement et réduction, parfois de façon exclusive, parfois combinée avec d'autres méthodes. Souvent, l'estimation de la taille de l'arbre de recherche, ou le calcul du temps d'exécution, se fera avec un outil que l'on appelle *mesurer et conquérir*.

1.5 Présentation des chapitres de la thèse

1.5.1 Chapitre 2 : Ensembles stables bicolorés

Nous avons défini une nouvelle variante du problème d'Ensemble Stable 1.3.3. Le problème d'Ensemble Stable Bicoloré consiste à chercher dans un graphe $G = (R \cup B, E)$, dont l'ensemble de sommets est partitionné en deux ensembles disjoints R et B, un ensemble stable contenant k_1 sommets dans R et k_2 sommets dans B.

La coloration (ou partition) des sommets du graphe fait ici partie de l'entrée du problème, et n'a aucune structure particulière. Ce n'est pas nécessairement une coloration propre. Les entiers k_1 et k_2 font aussi partie de l'entrée du problème.

Le problème d'ensemble stable bicoloré est NP-complet.

Nous avons utilisé la technique de branchement et réduction, combinée à la technique de programmation dynamique sur les décompositions arborescentes de graphe, plus précisément le *pathwidth*, la décomposition en chemin.

Nous avons effectué deux versions de l'analyse du temps d'exécution : une version simplifiée, où l'on se contente de calculer séparément les temps d'exécution des deux parties de l'algorithme. Le temps d'exécution de la partie branchement, due au vecteur de branchement $(1, 6)$, est de $O^*(1.2852^n)$. Quant à la partie programmation dynamique sur la décomposition en chemin, elle s'exécute en $O^*(1.26^n)$.

La version complexe de la mesure se fait à l'aide de la méthode *measure and conquer* que nous avons dû adapter afin de pouvoir effectuer l'analyse amortie malgré la transition d'une méthode à l'autre. Les poids attribués aux sommets d'un certain degré permettent d'affiner la mesure en essayant, autant que possible, de prendre en compte les effets de bord. L'algorithme que nous avons proposé permet de résoudre le problème d'ensemble dominant bicoloré avec un temps d'exécution de $O^*(1.2691^n)$.

L'étude de ce problème fut, à l'origine, motivée par son lien avec un autre problème connu de la théorie des graphes : le problème des bicliques.

L'algorithme permettant de résoudre le problème d'ensemble stable bicoloré nous permet de résoudre le problème de bicliques dans un graphe bipartite avec le temps $O^*(1.2691^n)$ grâce à une réduction polynomiale conservant la taille de l'entrée. Une autre réduction s'appliquant cette fois du problème de bicliques sur un graphe bipartite au problème de bicliques non-induites dans un graphe classique, nous permet de résoudre ce dernier en doublant la taille de l'entrée, et donc de trouver la solution en $O^*(1.2691^{2n})$, ou autrement dit, en $O^*(1.6107^n)$.

Ces résultats ont fait l'objet de deux publications :

[26] **Bicolored independent set and bicliques**,
J.F. Couturier et D. Kratsch,
Proceedings of CTW 2011, pp. 130–133.

[27] **Bicolored independent set and bicliques**,
J.F. Couturier et D. Kratsch,
Information Processing Letters, 112 (2012), pp. 329–334.

1.5.2 Chapitre 3 : Coloration et étiquetage de graphes cubiques

Lorsque l'on parle de coloration de graphes, le problème auquel on pense tout de suite est celui du nombre chromatique, c'est-a-dire de la coloration propre des sommets d'un graphe avec le plus petit nombre possible de couleurs. Mais il existe bien d'autres façons de colorer un graphe.

Nous avons travaillé sur plusieurs de ces autres colorations possibles, en nous concentrant sur le cas particulier des graphes cubiques, c'est-a-dire les graphes dont tous les sommets sont de degré 3.

Nous avons commencé par nous intéresser à la 3-coloration d'arêtes, c'est-à-dire de savoir s'il est possible ou non de colorer les arêtes d'un graphe avec seulement trois couleurs. L'algorithme de branchement que nous avons développé pour ce problème permet d'énumérer toutes les 3-arêtes colorations d'un graphe cubique en construisant un arbre de recherche de $O^*(2^{\frac{5n}{8}})$ feuilles. Cela nous donne une borne supérieure pour le temps d'exécution de l'algorithme, mais aussi une borne supérieure pour le nombre de 3-arêtes colorations dans un graphe cubique. Nous avons aussi, concernant le nombre de de 3-arêtes colorations dans un graphe, pu établir une borne inférieure. Il y a un graphe cubique qui a au moins $12^{\frac{n}{10}}$ 3-arêtes colorations différentes.

Ensuite, nous nous sommes intéressé au problème L(2,1)-Labelling. Il s'agit de faire un étiquetage des sommets d'un graphe avec des entiers de façon à ce que deux sommets adjacents aient une étiquette dont la différence

est supérieure ou égale à 2, et que deux sommets à distances 2 aient des étiquettes dont la différence est supérieure ou égale à 1. L'algorithme de branchement que nous avons défini pour énumérer tous les L(2,1)-labelling pour des étiquettes de 0 à 5 sur graphe cubique, s'exécute en $O^*(1.8613^n)$. Cette valeur nous donne aussi une borne supérieure pour le nombre de L(2,1)-labelling possible dans un graphe cubique.

Nous avons aussi pu établir que le nombre de L(2,1)-Labelling dans un graphe cubique pouvait être supérieur ou égal à $2^{\frac{n}{6}}$, définissant ainsi une borne inférieure pour le problème.

Ces résultats ont fait l'objet d'une publication, et sont actuellement en cours de publication dans le journal *Theory of Computing Systems*.

[49] **Colorings with few colors : counting, enumeration and combinatorial bounds**,
P.A. Golovach, D. Kratsch, J.F. Couturier,
Proceedings of WG 2010,
Springer, Lecture Notes in Computer Science 6410, pp. 39–50.

1.5.3 Chapitre 4 : Ensembles dominants minimaux par inclusion

Nous avons travaillé sur des algorithmes permettant de donner une borne supérieure au nombre d'ensembles dominants minimaux par inclusion dans un graphe. Cela non pas pour le cas général, pour lequel les meilleures bornes connues sont pour l'instant celles de Fomin et al. [43], mais pour des classes de graphe particulières.

Nous avons aussi cherché pour chacune des classes de graphe considérées, des exemples de graphes disposant du plus grand nombre possible d'ensembles dominants minimaux par inclusion afin de donner des bornes inférieures au problème. Le but implicite étant de trouver des bornes supérieures et des bornes inférieures les plus proches possibles. Ce but, même s'il n'a pas pu être atteint dans tous les cas, a tout de même pu l'être pour quelques-unes des classes de graphe.

Pour certaines classes de graphe, à savoir les co-graphes et les graphes trivialement parfaits, les bornes inférieures et supérieures se rejoignent.

Ces résultats ont fait l'objet d'une publication à SOFSEM 2012, une version journal est actuellement en cours de réalisation.

[24] **Minimal dominating set in graph classes : combinatorial bounds and enumeration**,
J.F. Couturier, P. Heggernes, P. van 't Hof, D. Kratsch,

Proceedings of SOFSEM 2012,
Springer, Lecture Notes in Computer Science 7147, pp. 202–213.

1.5.4 Chapitre 5 : Ensembles coupe cycles

Il a été démontré par B. Schwikowski et E. Speckenmeyer [90], que les ensembles coupe cycles minimaux par inclusion dans un graphe peuvent être énumérés avec un délai polynomial. Cela signifie, entre autre, que si l'on est capable de donner une estimation du nombre de ces ensembles coupe cycles minimaux par inclusion dans un graphe, alors on a par la même occasion, grâce à l'algorithme de B. Schwikowski et E. Speckenmeyer, une estimation du temps nécessaire pour tous les trouver.

Forts de cette observation, nous nous sommes appliqué à chercher des bornes supérieures pour le nombre d'ensembles coupe cycles pour un certain nombre de classes de graphe particulières.

Nous avons donc démontré que, pour les graphes cordaux et pour les co-graphes, le nombre maximum d'ensembles coupe cycles peut être borné par 1.585^n. Ces valeurs ont pu être trouvées grâce à des outils combinatoires.

Il est intéressant de noter que nous avons dans le même temps pu fournir plusieurs familles de graphes cordaux ou de co-graphes dont le nombre d'ensembles coupe cycles est effectivement 1.585^n, donnant ainsi une borne inférieure et une borne supérieure de complexités équivalentes. Cette valeur est donc ajustée.

Ces résultats ont fait l'objet d'une publication présentée cet été à la conférence COCOON 2012.

[25] **Maximum Number of Minimal Feedback Vertex Sets in Chordal Graphs and Cographs**,
Jean-François Couturier, Pinar Heggernes, Pim van 't Hof et Yngve Villanger,
Proceedings of COCOON 2012,
Springer, Lecture Notes in Computer Science 7434, pp. 133–144.

1.5.5 Chapitre 6 : Classification de complexité pour domination et variantes

Motivés par la dichotomie établie par P. Hell et J. Nesetril [53] sur le problème de H-Coloration, et par les travaux de V.V. Lozin et R. Mosca [71] sur la recherche de complexité des problèmes d'ensembles stables et d'ensembles stables dominants sur certaines classes de graphe, nous avons

souhaité étudier la complexité de différentes variantes du problème de domination sur les classes de graphe définies par l'absence de sous-graphes induits.

Étant données des classes de graphe, définies par l'absence de sous-graphes H induits, notre but est d'établir pour un problème X s'il est solvable en temps polynomial ou s'il est NP-complet sur la classe des graphes sans H. Les problèmes que nous avons étudiés sont les problèmes de domination, de domination stable, de domination connexe, de domination totale et de clique dominante.

Ici, nous appelons dichotomie une classification complète des classes de graphe sans H en fonction du fait que le problème étudié reste NP-complet ou devient polynomial.

Après nous être appliqué à rassembler de nombreux résultats connus de la littérature sur la complexité du problème de domination et de ses variantes sur les différentes classes de graphe, et dans quelques cas, après en avoir ajoutés nous-même, nous avons réussi à établir une dichotomie sur les classes de graphe, définies par des sous-graphes interdits, pour les problèmes de domination, de domination totale, et de domination connexe. Bien qu'elles soient incomplètes, nous donnons aussi les classifications pour le problème de domination stable et de clique dominante.

Nous donnons aussi, à la fin de ce chapitre, un algorithme permettant de résoudre le problème de domination sur les graphes sans $P_2 + P_3$, c'est-à-dire les graphes dont il n'existe pas de sous-graphes induits constitué de l'union disjointe d'un chemin de taille 2 et d'un chemin de taille 3, avec un temps d'exécution en $O^*(1.2279^n)$.

Deuxième partie

Problèmes de comptage

Chapitre 2

Ensemble stable bicoloré

2.1 Introduction

Un graphe *bicolore* est un graphe $G = (V, E)$ dans lequel chaque sommet du graphe est coloré par l'une des deux couleurs possibles, nous appellerons ces couleurs le **rouge** et le **bleu**. Pour désigner un tel graphe, nous utiliserons la notation $G = (R, B, E)$ où R représentera l'ensemble des sommets rouges, B représentera l'ensemble des sommets bleus et E représente, comme d'habitude, l'ensemble des arêtes. Remarquez que la coloration de G en rouge et bleu n'a pas besoin d'être une coloration propre, il s'agit d'une partition des sommets du graphe de façon totalement arbitraire.

Nous définissons une généralisation d'un problème NP-complet classique, à savoir le problème d'ensemble stable d'un graphe.

Bicolored Independent Set (BIS) : L'entrée du problème est composée d'une graphe bicolore $G = (R, B, E)$, où R et B sont une partition de l'ensemble des sommets, et de deux entiers k_1 et k_2. Le problème BIS consiste à décider s'il existe dans G un ensemble stable $I \subseteq V$ tel que $|I \cap R| = k_1$ et $|I \cap B| = k_2$.

Faire une réduction au problème NP-complet ENSEMBLE STABLE classique est facile. Pour n'importe quel graphe $G = (V, E)$, appliquez la coloration $R = V$ et $B = \emptyset$. Il existe dans G un ensemble stable de taille k si, et seulement si, il existe dans $G' = (R, B, E)$ un BIS de taille $k_1 = k$ et $k_2 = 0$.

Notre motivation pour étudier le problème des ensembles stables bicolorés est due à sa relation importante avec un autre problème très étudié, le problème des bicliques induites dans un graphe.

2.1.1 Les *Bicliques*

Soit deux ensembles de sommets $X \subseteq V$ et $Y \subseteq V$, appartenant au graphe $G = (V, E)$, tels que X et Y soient des ensembles stables, et que $\forall x \in X, \forall y \in Y$, il existe une arrête $xy \in E$. Dans ce cas, (X, Y) est appelé une *biclique induite* de G. Si X et Y sont juste des sous-ensembles de sommets de G, sans être forcément stables, mais qu'ils répondent à la condition $\forall x \in X, \forall y \in Y$ il existe une arrête $xy \in E$, alors (X, Y) est appelé une *biclique non-induite* de G. Des applications peuvent utiliser des bicliques pour résoudre des problèmes de Data mining, intelligence artificielle, biologie, automatisme ou encore théorie des langages, comme on peut le voir dans [3]. La complexité des algorithmes pour résoudre les problèmes de bicliques a été étudiée de plus en plus. On sait que le problème de trouver une biclique avec un nombre de sommets maximum dans un graphe bipartie est polynomial [28], et NP-complet pour les graphes dans le cas général [100]. Le problème de trouver une biclique avec un nombre d'arêtes maximum a été montré comme étant NP-complet par Peeters en 2003 [84]. Au cours de la dernière décennie, l'intérêt pour l'énumération de toutes les bicliques maximales par inclusion dans un graphe, qu'il soit bipartie ou pas, fut relativement important [2, 29, 30, 72].

Ce qui nous intéresse est de trouver des bicliques induites dans un graphe bipartie, ou de trouver des bicliques non induites dans un graphe.

Bipartite Biclique L'entrée du problème est un graphe bipartie $G = (X, Y, E)$ et deux entiers positifs k_1 et k_2. Le problème consiste à trouver s'il existe dans G deux sous-ensembles de sommets $X' \subseteq X$ et $Y' \subseteq Y$, tels que $|X'| = k_1, |Y'| = k_2$ et que le couple (X', Y') forme une biclique.

Non-induced Biclique L'entrée du problème est un graphe $G = (V, E)$ et deux entiers positifs k_1 et k_2. Le problème consiste à trouver s'il existe dans G deux sous-ensembles de sommets $X' \subseteq V$ et $Y' \subseteq V$ tels que $\forall x \in X', \forall y \in Y'$ alors $xy \in E$, c'est-à-dire (X', Y') forme une biclique non induite.

Le problème qui consiste à décider si, pour un graphe bipartie $G = (X, Y, E)$ et un entier k, G contient une biclique de taille (k, k), est connu pour être NP-complet [46]. Il est assez facile de voir que cela implique directement que les problèmes Bipartite Biclique et Non-induced Biclique sont eux-aussi NP-complets.

2.1.2 Problèmes de comptage

Tous les problèmes de décision que nous venons jusque-là de définir ont naturellement des problèmes de comptage correspondant.

#Bicolored Independent Set (#BIS) L'entrée est un graphe bicolore $G = (R, B, E)$ avec deux entiers positifs k_1 et k_2. La question est de savoir combien il y a d'ensembles stables bicolorés k_1, k_2 dans le graphe G.

#Bipartite Biclique L'entrée est un graphe bipartie $G = (X, Y, E)$ avec deux entiers positifs k_1 et k_2. La question est de savoir combien il y a de bicliques k_1, k_2 dans G.

#Non-Induced Biclique L'entrée est un graphe $G = (V, E)$ avec deux entiers positifs k_1 et k_2. La question est de savoir combien il y a de bicliques non induites de taille k_1, k_2 dans le graphe G.

Il est évident que, si l'on sait résoudre un problème de comptage, alors on sait automatiquement résoudre le problème de décision correspondant. Il suffit de répondre que la solution existe lorsque le nombre compté est supérieur à zéro.

2.1.3 Travaux précédant sur les bicliques

Des algorithmes exacts et des variantes difficiles des problèmes de biclique ont été étudiés par S. Gaspers dans [47]. Fernau et al. proposent dans [35] un algorithme en espace polynomial avec un temps d'exécution de $O^*(1.8899^n)$ et un algorithme en espace exponentiel avec un temps d'exécution de $O^*(1.8458^n)$ pour le problème des Non-Induced Bicliques. Les meilleurs algorithmes exacts connus avant nos travaux résolvaient le problème de non Inducedd Biclique avec un temps d'exécution de $O^*(1.6914^n)$, et le problème de Bipartie Biclique avec un temps d'exécution de $O^*(1.3006^n)$. Ces deux derniers algorithmes utilisant juste un espace polynomial [9].

2.1.4 Nos résultats

Notre principal résultat, dans ce chapitre, consiste en un algorithme exact qui résout le problème de comptage #Ensemble Stable Bicoloré (en anglais, #Bicolored Independent Set) avec un temps d'exécution de $O^*(1.2691^n)$. Cet algorithme fonctionne en deux parties, la première partie étant un branchement classique de la méthode *branche and reduce* et la

deuxième partie consistant en une programmation dynamique sur une décomposition du graphe en chemin. L'analyse du temps d'exécution de l'algorithme en global est faite grâce à la méthode *measure and conquer* désormais bien connue [38]. La technique que nous avons utilisé pour construire et analyser notre algorithme avait été initiée dans [41, 94].

Notre algorithme, qui résout le problème de comptage #Bicolored Independent Set, peut être utilisé pour établir un algorithme qui résout le problème #bipartie biclique avec un temps d'exécution équivalent, c'est-à-dire $O^*(1.2691^n)$, et nous permet aussi de définir un algorithme résolvant le problème des Non-Induced Bicliques avec un temps d'exécution de $O^*(1.6107^n)$. Malheureusement, étant donné qu'une partie de l'algorithme utilise de la programmation dynamique, l'espace nécessaire est quant à lui exponentiel.

2.2 Bicolored Independent Set

Je vais maintenant vous présenter l'algorithme exact permettant de résoudre le problème de compter les ensembles stables bicolorés.

Algorithme `BIS`(G, k_1, k_2)**.**
Entrée : un graphe bicoloré $G = (R, B, E)$,
deux entiers positifs k_1 et k_2.
Résultat : un entier représentant le nombre d'ensemble stable
dans $I \subseteq V(G)$ et tel que $|I \cap R| = k_1$ et $|I \cap B| = k_2$.

1 : Choisir un sommet v de degré maximum ;
2 : Si $d(v) \leq 4$
3 : Résultat = `BIS-dynamique`(k_1, k_2, G) ;
4 : Sinon
5 : `temporaire` $\leftarrow$ `BIS`$(k_1,k_2,G\backslash\{v\})$;
6 : Si $v \in R$
7 : Résultat = `BIS`$(k_1\text{-}1,k_2,G\backslash N[v])$ + `temporaire` ;
8 : Else
9 : Résultat = `BIS`$(k_1,k_2\text{-}1,G\backslash N[v])$ + `temporaire` ;

Algorithme `BIS` pour ensemble stable bicoloré

Notre algorithme combine une partie de branchement avec une partie de

programmation dynamique. Tant qu'il y a, dans le graphe du sous-problème courant, un sommet de degré au moins égal à 5, l'algorithme fait un branchement basé sur un sommet v de degré maximum pour définir deux nouveaux sous-problèmes.

Dans le premier cas, il élimine v et décide qu'il ne fera jamais partie d'un ensemble stable bicoloré. Le sommet v est donc supprimé du graphe et l'on continue à chercher le nombre d'ensembles stables composés de k_1 sommets rouges et de k_2 sommets bleus dans le sous-graphe $G\backslash\{v\}$. Comme on le voit à la ligne 5 de l'algorithme, le résultat de ce sous-problème est alors stocké temporairement dans une variable.

Dans le deuxième cas, l'algorithme décide que le sommet v fera partie de la solution, et l'ajoute donc virtuellement à tous les ensembles stables bicolorés qui pourraient être créés dans cette branche. Si v fait partie de l'ensemble stable, alors aucun de ses voisins ne pourra plus en faire partie. On supprime donc $N[v]$ du graphe, le voisinage fermé de V, c'est-à-dire v et l'ensemble de ses voisins. Il faut ensuite distinguer deux possibilités en fonction de la couleur du sommet v. Si v est rouge, alors nous savons que nous avons besoin d'un sommet rouge en moins, et comme on peut le voir à la ligne 7, on cherche donc dans le sous-problème défini par $G\backslash\{v\}$ le nombre d'ensembles stables contenant $k_1 - 1$ sommets rouges et k_2 sommets bleus. A l'inverse, si v est bleu, nous savons que nous avons besoin d'un sommet bleu en moins, et, comme à la ligne 9, on cherche donc dans le sous-problème défini par $G\backslash\{v\}$ le nombre d'ensembles stables contenant k_1 sommets rouges et $k_2 - 1$ sommets bleus.

Le branchement s'arrête lorsque le degré maximum $\Delta(G')$ du graphe courant G' est inférieur ou égal à 4. A ce moment, on passe à l'utilisation de la partie de programmation dynamique.

BIS-dynamique : La partie de programmation dynamique s'effectue sur une décomposition arborescente du graphe en chemin, communément appelée le *Pathwidth*. Il est donc nécessaire de commencer par déterminer cette décomposition arborescente. Dans [41], Fomin at al. proposent un algorithme polynomial permettant de décomposer un graphe pour obtenir sa décomposition arborescente en chemin. Ils donnent aussi une borne sur la taille des sacs du *Pathwidth* que nous utiliserons lors de l'analyse du temps d'exécution.

Il n'est pas difficile, une fois que l'on a une décomposition, de s'assurer qu'elle soit *jolie*. Cela signifie qu'il n'y a qu'un seul sommet de différence entre deux sacs consécutifs dans le *Pathwidth*.

Pour chaque sac, nous allons donc définir une série d'enregistrements. Un enregistrement se compose de quatre parties :

– $\mathcal{S}$ le sous-ensemble de sommets du sac concerné.
– k_1 le nombre de sommets de $\mathcal{S}$ qui sont rouges.
– k_2 le nombre de sommets de $\mathcal{S}$ qui sont bleus.
– *nbsol* le nombre de solutions que représente cet enregistrement.

Nous avons donc une série d'enregistrements de la forme :

$$[\mathcal{S} \cap sac, k_1, k_2, nbsol]$$

Dans un sac, chaque sommet peut être :
– Soit ne faisant pas partie des solutions représentées par cet enregistrement.
– Soit faisant partie des solutions représentées par cet enregistrement et :
  · Être un sommet rouge.
  · Être un sommet bleu.

Le fait de pouvoir faire ou pas partie des solutions représentées par un enregistrement définit $2^{Pathwidth}$ possibilités de sous-ensembles. Le nombre de sommets rouges peut être compris entre 0 et n, de même que pour le nombre de sommets bleus, ce qui nous permet de conclure qu'il y a au plus n^2 enregistrements pour chaque possibilité de sous-ensemble. Le fait d'enregistrer le nombre de solutions correspondant à un enregistrement ne fait pas augmenter le nombre d'enregistrements. On peut donc en conclure qu'il y a au maximum $n^2 \cdot 2^{Pathwidth}$ enregistrements. Cette valeur permettra de définir le temps d'exécution de la partie de programmation dynamique, mais définit aussi l'espace mémoire nécessaire à l'exécution de l'algorithme.

Passer à un sac plus grand : Comme nous travaillons avec une décomposition *jolie*, le fait de passer d'un sac X à un sac plus grand Y veut dire que l'on a ajouté un seul sommet z (et que l'on n'a supprimé personne). Ou autrement dit : $Y = X \cup \{z\}$. Bien entendu, puisque l'on avance le long du chemin, on connait à ce moment tous les enregistrements possibles de X, et on veut déterminer tous les enregistrements possibles de Y.

Pour chaque enregistrement $[\mathcal{S}, k_1, k_2, nbsol]$ de X :
– Si on décide que z ne fait pas partie des solutions, ajouter $[\mathcal{S}, k_1, k_2, nbsol]$ aux enregistrements de Y.
– Si on décide que z fait partie des solutions :
  · z est rouge, on ajoute $[\mathcal{S} \cup \{z\}, k_1 + 1, k_2, nbsol]$ aux enregistrements de Y.
  · z est bleu, on ajoute $[\mathcal{S} \cup \{z\}, k_1, k_2 + 1, nbsol]$ aux enregistrements de Y.

Passer à un sac plus petit : Comme précédemment, le fait de travailler sur une décomposition *jolie* nous permet d'affirmer que si l'on passe d'un

sac X à un sac plus petit Y, cela signifie qu'il y a un sommet z dans X qui n'apparaît plus dans Y, et que c'est la seule différence entre les deux ensembles. Ou autrement dit : $Y = X \backslash \{z\}$. Encore une fois, puisque l'on avance le long du chemin, on connait à ce moment tous les enregistrements possibles de X, et on veut déterminer tous les enregistrements possibles de Y.

Pour chaque enregistrement $[\mathcal{S} \cup \{z\}, k_1, k_2, nbsol_1]$ qui contient z, il existe un enregistrement $[\mathcal{S}, k_1, k_2, nbsol_2]$ correspondant. Dans Y les solutions codées par ces deux enregistrements ne pourront plus être différenciées, nous les rassemblons donc dans un nouvel enregistrement $[\mathcal{S}, k_1, k_2, nbsol_1 + nbsol_2]$ que l'on ajoute à Y.

Lorsque l'on arrive au dernier sac, on a donc une série d'enregistrements avec un nombre de sommets rouges, un nombre de sommets bleus, et le nombre de solutions qui correspondent.

2.3 Première approche, analyse simple

La partie branchement. Le branchement sur un sommet v de degré au moins 5 permet de définir la récurrence suivante.

Si on choisit de prendre le sommet v dans l'ensemble stable, alors aucun de ses voisins ne peut y être. Supprimer v, et tout son voisinage, dans le graphe G, permet d'obtenir le graphe $G'_{select} = G \backslash N[v]$ sur lequel il faut chercher le reste. De plus, l'on fait décroître l'un des deux entiers en fonction de la couleur de v. Si v est rouge, il reste à chercher dans le sous-graphe résultant G'_{select} un ensemble stable avec $k_1 - 1$ sommets rouges et k_2 sommets bleus. À l'inverse, si v est bleu, alors il reste à chercher dans le graphe G'_{select} un ensemble stable avec k_1 sommets rouges et $k_2 - 1$ sommets bleus. Quoi qu'il en soit, et parce que le sommet v est toujours de degré supérieur ou égal à 5, la taille de l'instance définie par G'_{select} en incluant v dans l'ensemble stable est inférieure à la taille de l'instance définie par G d'au moins 6 sommets. On dira que $\Delta_{select} \geq 6$.

Si l'on choisit de ne pas prendre le sommet v dans l'ensemble stable, alors on supprime simplement v du graphe G, afin d'obtenir le graphe $G'_{discard} = G \backslash \{v\}$, et de continuer à chercher sur ce dernier un ensemble stable contenant k_1 sommets rouges et k_2 sommets bleus. On dira que $\Delta_{discard} = 1$.

Cela implique que le nombre de feuilles dans l'arbre de recherche est borné par la récurrence : $T(n) = T(n - \Delta_{select}) + T(n - \Delta_{discard})$ qui est aussi le pire des cas. Le vecteur de branchement est donc au moins $(1, 6)$ et la constante de branchement correspondante est environ 1.2852. Le temps d'exécution de la

partie branchement est donc $O(1.2852^n)$. On fera remarquer que le nombre de feuilles dans cet arbre de recherche, correspondant à un graphe G' qui aurait exactement n' sommets, est $O(1.2852^{n-n'})$.

La partie dynamique. Pour prendre en compte la partie de programmation dynamique, soit G' un graphe se trouvant dans l'une des feuilles de l'arbre de recherche de la partie algorithme de branchement. Clairement, le degré maximum de G' est $\Delta(G') \leq 4$. Notre algorithme applique une programmation dynamique sur la décomposition arborescente en chemin (jolie) du graphe G'.

Pour pouvoir analyser le temps d'exécution, nous avons besoin d'une borne supérieure pour la taille de la décomposition en chemin de G'. Nous utiliserons le lemme suivant de Fomin et al. [41].

Lemme 1. *Pour tout $\epsilon > 0$, il existe un n_ϵ tel que pour tout graphe G de maximum degré $\Delta(G) \leq 4$ et avec $n > n_\epsilon$, alors on a :*

$$pw(G) \leq \frac{n_3}{6} + \frac{n_4}{3} + \epsilon \cdot n$$

Où n_i est le nombre de sommets de degré i dans G, et où $pw(G)$ représente la taille d'une décomposition en chemin de G que l'on peut obtenir avec un temps polynomial.

Donc, comme dans notre cas $\Delta(G') \leq 4$, on obtient que l'on peut déterminer en temps polynomial une décomposition en chemin de G' dont la taille serait bornée par $pw(G') \leq \frac{n'}{3} + \epsilon \cdot n'$, ou $n' = V(G')$. On peut en conclure que le temps d'exécution de la partie de programmation dynamique est de $O(1.26^{|V(G')|})$ pour chaque graphe G' que l'on pourra trouver dans une feuille de l'arbre de recherche de la partie branchement.

Pour analyser l'algorithme de façon globale, considérons chaque graphe G' que l'on peut trouver dans les feuilles de l'arbre de recherche comme ayant h sommets. Bien entendu, tous les graphes G' que l'on peut trouver dans les feuilles n'ont pas exactement le même nombre de sommets, mais étant donné que l'on cherche une borne supérieure sur le temps nécessaire à atteindre les solutions, cela n'est pas gênant pour l'analyse.

Le nombre de feuilles de l'arbre de recherche est $O(\alpha^{n-h})$ (avec ici, $\alpha = 1.2852$) et la partie de programmation dynamique de chaque feuille s'exécute en β^h (avec ici $\beta = 1.26$).

Comme il est déjà remarqué dans [41], le temps d'exécution d'un algorithme ainsi décomposé en deux parties peut être considéré de la façon suivante :

$$\sum_{h=0}^{n} O(\alpha^{n-h} \cdot \beta^h) = O(\max(\alpha, \beta)^n)$$

Par conséquence, notre algorithme BIS a un temps d'exécution borné par $O(1.2852^n)$.

2.4 Deuxième approche, measure and conquer

Nous pouvons obtenir une meilleure borne sur le temps d'exécution au pire des cas en utilisant une méthode d'analyse plus sophistiquée sur les instances des sous-problèmes de l'algorithme de branchement. Le fait d'analyser un algorithme combinant une partie de branchement avec une autre partie en programmation dynamique sur une décomposition arborescente en chemin a été initié par Fomin et al. [44, 41]. D'autres applications peuvent aussi être trouvées dans [94]. L'analyse de temps d'exécution qui suit est basée sur ces travaux.

Rappelons que dans l'analyse simple, la mesure utilisée pour comparer la taille d'un graphe G était le nombre de ses sommets n. Maintenant, nous allons définir la mesure d'une graphe avec la formule suivante :

$$\mu = \mu(G) = \sum_{0 \leq i \leq n} n_i \cdot w_i$$

Dans cette formule, n_i représente le nombre de sommets de degré i dans le graphe G, et w_i représente le poids d'un sommet de degré i. Pour tout i, nous avons $0 \leq w_i \leq 1$. Le but de l'analyse que nous allons faire maintenant consiste en fait à déterminer quel est le meilleur choix pour les poids des sommets. Nous donnerons les valeurs correspondantes à la fin de cette section.

Afin de simplifier l'analyse, nous allons aussi définir une petite restriction sur les poids, à savoir $\forall i \geq 0, w_i \leq w_{i+1}$. Cette restriction simplifie l'analyse sans avoir d'impact sur la meilleure borne de temps d'exécution que l'on puisse analyser.

Toujours pour simplifier l'analyse, nous allons décider de fixer les poids pour tout sommet de degré supérieur ou égal à 6. Tout sommet de degré au moins 6 aura donc un poids égal à 1. De plus, nous définirons λ_i comme étant le poids total de l'ensemble des sommets de degré i. Ce qui implique que :

$$\lambda_i = n_i \cdot w_i$$

et, par conséquent :

$$\mu(G) = \sum_{0 \leq i \leq n} \lambda_i$$

Afin de simplifier la notation, nous définissons l'ensemble des sommets de degré 6 ou plus comme étant l'ensemble ≥ 6. Ainsi, $n_{\geq 6}$ représentera désormais le nombre de sommets dans le graphe G dont le degré est supérieur ou

égal à 6. Le poids d'un sommet de degré 6 ou plus sera représenté par $w_{\geq 6}$ et le poids total des sommets de degré supérieur ou égal à 6 sera noté $\lambda_{\geq 6}$. Comme nous avons fixé $w_{\geq 6} = 1$, $\lambda_{\geq 6} = n_{\geq 6}$ implique :

$$\mu(G) = \sum_{0 \leq i \leq 5} (n_i \cdot w_i) + n_{\geq 6} = \sum_{0 \leq i \leq 5} \lambda_i + n_{\geq 6}$$

La partie branchement. Dans un premier temps, nous allons nous occuper de la partie branchement. Supposons que l'algorithme effectue un branchement sur le graphe G, et choisit un sommet v afin de brancher dessus. Comme dans l'analyse simple, nous pouvons être sûr que $d(v) \geq 5$. Dans la partie branchement, toutes les récurrences sont de la forme suivante :

$$T(\mu(G)) = T(\mu(G) - \Delta_{select}) + T(\mu(G) - \Delta_{discard})$$

Dans cette formule, Δ_{select} et $\Delta_{discard}$ représentent respectivement des bornes inférieures sur la valeur dont la mesure décroît entre le graphe G et le graphe G'_{select} résultant si l'on choisit de prendre v dans la solution, ou le graphe $G'_{discard}$ résultant si l'on choisit que v ne fera pas partie de la solution.

Si nous choisissons de prendre v dans la solution, alors le sommet v et tous ses voisins seront supprimés dans le graphe résultant, nous aurons $G'_{select} = G \backslash N[v]$. Par conséquence, le poids décroît de la somme du poids de v et du poids de ses voisins. Cela implique :

$$\Delta_{select} = w_{d(v)} + \sum_{u \in N(v)} w_d(u)$$

Si nous décidons que le sommet v ne fera pas partie de la solution, alors nous supprimons v du graphe, et, par conséquence, tous les voisins de v vont voir leur degré diminuer de 1. Il en suivra que la mesure du graphe va diminuer du poids de v, et pour tous les sommets u dans le voisinage de v, leur poids va décroître de la différence entre le poids $w_{d(u)}$ et le poids pour le degré inférieur $w_{d(u)-1}$. Cela implique :

$$\Delta_{discard} = w_{d(v)} + \sum_{u \in N(v)} (w_{d(u)} - w_{d(u)-1})$$

Soit G un graphe de mesure $\mu = \mu(G)$, soit v le sommet de v sur lequel on branche et appelons $t = d(v) \geq 5$. Soit $u_1, u_2, \cdots, u_t$ les voisins du sommet v classés par ordre croissant de leurs degrés, c'est-à-dire que $\forall i \in \{1, 2, \cdots, t\}$, si $d_i = d(u_i)$ alors nous avons $d_1 \leq d_2 \leq \cdots \leq d_t$. Maintenant, tenant compte du fait que l'on branche toujours sur un sommet de degré maximum, nous

allons établir un système de récurrences pour notre algorithme de branchement. Pour tout $t, d_1, d_2, \cdots, d_t$ tels que $t \geq 5$ et $d_1 \leq d_2 \leq \cdots \leq d_t \leq t$:

$$T(\mu) = T(\mu - w_t - \sum_{i=1}^{t} w_{d_i}) + T(\mu - w_t - \sum_{i=1}^{t} (w_{d_i} - w_{(d_i-1)}))$$

Cette collection de récurrences linéaires est bien trop grande pour que l'on puisse travailler avec directement. En effet, si l'on considère que le degré d'un sommet n'est borné que par n, le nombre de sommets dans le graphe G, (ou plutôt $n-1$ puisqu'il ne peut être voisin de lui-même), alors le nombre de récurrences dans ce système est environ de $\sum_{r=5}^{n}(2 \cdot r^r)$. Et comme il n'y a pas de borne sur n, il est facile de voir que le nombre de récurrences est potentiellement infini. Fort heureusement, nous pouvons utiliser les restrictions que nous avons définies au début de cette section pour limiter le nombre de récurrences à prendre en considération. Pour voir cela, considérons n'importe quelle récurrence R avec $t \geq 7$ et $d_1 \leq d_2 \leq \cdots \leq d_t$ et une récurrence R' avec $t' = 6$ et $d'_1 \leq d'_2 \leq \cdots \leq d'_6 \leq t'$ avec $d'_i = min(d_i, 6)$.

Comme nous avons défini que $w_i = 1$ pour tous les $i \geq 6$, on obtient rapidement que $\Delta_{select}(R) \geq \Delta_{select}(R')$ et que $\Delta_{discard}(R) \geq \Delta_{discard}(R')$. Ce qui signifie que, pour toute récurrence R avec $t \geq 7$, il existe une récurrence R' correspondante avec $t' = 6$ tel que la valeur du branchement sur R' soit supérieure à la valeur du branchement sur R, et donc, un meilleur choix de borne supérieure.

De ce fait, nous pouvons donc en toute sécurité négliger les récurrences avec un $t \geq 7$, et, pour chercher les bonnes valeurs des différents w_i, nous pouvons nous contenter de faire les calculs sur les récurrences avec un $t \leq 6$.

La programmation dynamique. Nous devons maintenant passer à l'analyse de la programmation dynamique. Lorsque le degré maximum $\Delta(G')$ du graphe courant G' est au plus 4, alors nous faisons appel à notre algorithme $BIS - dynamique(G', k_1, k_2)$. Notons que cet algorithme a besoin d'une décomposition arborescente en chemin du graphe G'. Pour être précis, cet algorithme est appelé pour chaque feuille de l'arbre de branchement construit dans la première approche. L'analyse ici va demander un peu plus d'attention que lorsque nous avons fait l'analyse simple. Nous avons besoin d'une borne supérieure sur la taille d'une décomposition arborescente de G' qui sera en fonction, non plus du nombre de sommets dans le graphe, mais plutôt de la mesure $\mu(G')$ que l'on aura pour chaque G' d'une feuille de l'arbre de recherche.

De la même manière que pour l'analyse simple, nous allons utiliser le lemme 1. Puisque tous les graphes G' apparaissant dans des feuilles de la

partie branchement ont un degré maximum $\Delta(G') \leq 4$, alors pour tout $\epsilon > 0$, pour tout graphe G' ayant au moins n_ϵ sommets :

$$pw(G') \leq \frac{n_3}{6} + \frac{n_4}{3} + \epsilon \cdot n$$

Une décomposition arborescente en chemin d'une taille inférieure à $\frac{n_3}{6} + \frac{n_4}{3} + \epsilon \cdot n$ peut être trouvée en temps polynomial.

Mais nous avons besoin d'une borne supérieure de la taille de la décomposition arborescente en chemin qui soit en fonction du poids $\mu(G')$ du graphe G, et non plus du nombre de ses sommets. Pour faire cela, nous allons réexprimer la formule $\frac{n_3}{6} + \frac{n_4}{3} + \epsilon \cdot n$ en remplaçant les valeurs de n_3 et n_4 par leurs expressions équivalentes en fonction de la mesure.

Par construction, et par définition, de la mesure nous avons :

$$\mu = \sum \lambda_i = \sum (n_i \cdot w_i)$$

où μ est la mesure d'un graphe G' se trouvant dans une feuille de l'arbre, et pour tout $i \geq 0$, nous avons donc que $\lambda_i = n_i \cdot w_i$ est le poids total des sommets de degré i pour tout $i \geq 0$. Cela implique aussi que $n_i = \frac{\lambda_i}{w_i}$ et que nous pouvons alors substituer les n_i dans la formule pour établir la borne supérieure :

$$pw(G') \leq \frac{\lambda_3}{6 \cdot w_3} + \frac{\lambda_4}{3 \cdot w_4} + \epsilon \cdot n$$

Il reste à déterminer, pour toutes les possibilités de valeurs des poids w_3 et w_4, la valeur maximum correspondante pour la borne supérieure, et cela en fonction de toutes les valeurs possibles de λ_3 et λ_4. Pour faire cela, nous utiliserons le fait que $\lambda_3 + \lambda_4 \leq \mu(G') = \mu$.

Finalement, nous pouvons déterminer la meilleur borne supérieure possible pour le temps d'exécution de l'algorithme BIS. Pour chaque choix possible des poids $w_0 \leq w_1 \leq \cdots \leq w_5$, le temps d'exécution total de l'algorithme est :

$$O^*(\sum \alpha^{n-\mu} \cdot \beta^\mu) = O^*(\max(\alpha, \beta)^n)$$

où $O^*(\alpha^{n-\mu})$ est une borne supérieure au nombre de feuilles contenant un graphe de poids μ dans l'arbre de recherche de la partie branchement, et où $O^*(\beta^\mu)$ est le temps d'exécution de l'algorithme de programmation dynamique sur une feuille contenant un graphe G' de mesure μ. Nous avons alors besoin de trouver les bonnes valeurs pour les poids w_i afin de balancer α et β. En effet, on constatera que, à mesure que les poids permettent de faire diminuer la valeur d'α, celle de β, au contraire, a tendance à augmenter. Et inversement, faire diminuer la valeur de β a pour conséquence d'augmenter la valeur de α.

Lorsque nous utilisons les poids :

w_0	0
w_1	0.5386
w_2	0.8059
w_3	0.9233
w_4	0.9701
w_5	0.9999
$w_{\geq 6}$	1

, nous obtenons les résultats suivants.

Dans la partie branchement, quelle que soit la valeur de μ, le nombre de feuilles de l'arbre de recherche qui contiennent un graphe G' de mesure $\mu(G') = \mu$ est $O^*(1.2691^{n-\mu})$. Dans la partie de programmation dynamique, le temps d'exécution pour une feuille de l'arbre de recherche qui contiendrait un graphe G' de poids $\mu(G') = \mu$ est de $O^*(1.2691^{\mu})$. Cette valeur est atteinte si l'on se trouve dans le cas d'un graphe ne contenant que des sommets de degré 4, c'est-à-dire un graphe $4-$régulier.

Si le fait de trouver quelles étaient les valeurs optimales a demandé beaucoup d'efforts de calcul avec l'aide d'outils informatiques, la vérification des poids choisis et des valeurs correctes pour les bornes supérieures de temps d'exécution se font plus facilement. Il suffit de résoudre le système de récurrence décrit au cour de l'analyse en remplaçant les w_i par leur valeur correspondante. Comme on peut le voir dans [42], où les auteurs donnent le code nécessaire à la vérification des poids qu'ils ont choisi.

La combinaison de tout ce que nous venons de voir nous permet la conclusion suivante : l'algorithme BIS a un temps d'exécution en $O^*(1.2691^n)$.

Théorème 1. *L'algorithme BIS résout le problème de comptage* #Bicolored Independent Set *(#Ensemble stable bicoloré) ainsi que sa version de décision, et son temps d'exécution est de $O(1.2691^n)$.*

2.5 Bicliques biparties et non-induites

Dans cette section, je vais vous présenter deux réductions en temps polynomial, la première étant relativement simple, et la seconde, un peu plus complexe. Ces deux réductions sont des réductions de Turing, en les utilisant conjointement avec notre algorithme qui résout BIS en temps $O(1.2691^n)$, nous obtiendrons des algorithmes de comptage qui s'exécutent avec une borne sur le temps d'exécution inférieure à celle des meilleurs algorithmes de décision connus précédemment [9].

2.5.1 Biclique bipartie

Le problème d'ensemble stable bicoloré peut être vu comme une généralisation du problème de biclique bipartie grâce à la réduction suivante.

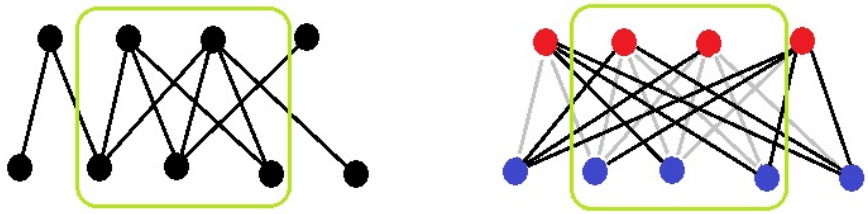
Théorème 2. *Il y a une réduction en temps polynomial directe entre le problème de* #Bicliques bipartie *et le problème de* #Ensemble stable bicoloré.

Démonstration. Soit (G, k_1, k_2) l'instance de n'importe quel problème de Bicliques bipartie. Dans ce cas, $G = (X, Y, E)$ est un graphe bipartie où X et Y représente les deux ensembles stables qui partitionnent l'ensemble des sommets de G, k_1 et k_2 sont deux entiers positifs.

Ce que nous voulons, c'est trouver dans le graphe G un sous-ensemble $\mathcal{X} \subseteq X$ tel que $|\mathcal{X}| \geq k_1$ et un sous ensemble $\mathcal{Y} \subseteq Y$ tel que $|\mathcal{Y}| \geq k_2$ et avec la propriété que, pour tout $x \in \mathcal{X}$ et $y \in \mathcal{Y}$, alors l'arête $xy \in E$.

Nous construisons une instance (G', k_1, k_2) d'Ensemble Stable Bicoloré de la manière qui suit : G' est le graphe complément bipartie de G, c'est-à-dire que $G' = (X, Y, E')$, avec $E' = \{xy | x \in X, y \in Y, xy \notin E\}$. Vous remarquerez que le graphe G' obtenu est un graphe bipartie.

Nous colorons toutes les arêtes de l'ensemble X en rouge, et toutes les arêtes de l'ensemble Y en bleu.



Une biclique dans un graphe bipartie, et l'ensemble stable bicoloré correspondant.

Nous pouvons voir facilement que, s'il existe une biclique dans le graphe G, avec k_1 sommets dans l'ensemble X et k_2 sommets dans l'ensemble Y, alors elle correspond à un ensemble stable bicoloré dans le graphe G', avec k_1 sommets rouges et k_2 sommets bleus. □

La réduction en temps polynomial du théorème 2 peut être combinée facilement avec l'algorithme #Ensemble Stable Bicoloré de la section précédente. Pour compter le nombre de bicliques biparties qui sont solution

de l'instance (G, k_1, k_2) dans le graphe G, il suffit simplement de générer le graphe G', complément bipartie de G, de donner sa couleur à chaque partition, rouge pour l'ensemble X et bleu pour l'ensemble Y, puis de lancer notre algorithme qui résout le problème #Ensemble stable bicoloré en temps $O(1.2691^n)$ sur l'instance (G', k_1, k_2). Par conséquence, nous obtenons le théorème suivant :

Théorème 3. *Le problème* #bipartie biclique *peut être résolu en temps* $O(1.2691^n)$.

Bien entendu, puisque nous pouvons résoudre la version comptage de ce problème, le problème de décision correspondant peut être résolu dans le même temps d'exécution.

2.5.2 Les bicliques non-induites

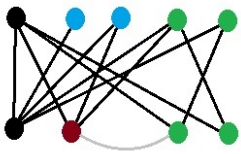
Nous allons montrer maintenant une réduction de Turing en temps polynomial pour réduire le problème des bicliques non-induites à celui des bicliques dans un graphe bipartie. Cette réduction est différente de celle utilisée dans [9]

Théorème 4. *Il y a une réduction de Turing en temps polynomial entre le problème des* Bicliques non-induites *et le problème des* Bicliques Biparties.

Démonstration. Soit $G = (V, E)$ un graphe, et soit $ab \in E$ une arête de ce graphe. Nous définissons un graphe bipartie $G_{ab} = (A, B, E_{ab})$ de la façon qui suit. Les deux partitions A et B de l'ensemble des sommets de G_{ab} sont des ensembles stables, définissant la bipartition du graphe G_{ab}. La partition A est composée d'un groupe de sommets qui représentent en bijection les sommets du voisinage $N_G(b)$ du sommet $b \in V(G)$, le sommet a étant compris. De même, la partition B est composée d'une groupe de sommets qui représentent en bijection les sommets du voisinage $N_G(a)$ du sommet $a \in V(G)$, le sommet b étant compris. S'il y a dans le graphe G des sommets z étant des voisins commun de a et de b, alors on remarque que ces sommets z seront représentés deux fois : une première fois dans $z_A \in A$ en tant que voisin de b, et la seconde fois dans $z_B \in B$ en tant que voisin de a. Pour un sommet $z \in V(G)$, nous utiliserons la notation z_A pour designer le sommet qui le représente dans la partie A, et z_B pour designer le sommet qui le représente dans B. Pour tout $x_A \in A$ et $y_B \in B$, il y a une arête $x_A y_B \in E_{ab}$ si, et seulement si, il y a une arête $xy \in E$. Notez que cela implique que $z_A z_B \notin E_{ab}$ pour tout $z \in N(a) \cap N(b)$. Enfin, nous remarquerons que, puisque chaque sommet de

G risque d'être représenté deux fois dans le graphe G_{ab}, alors ce dernier peut avoir au plus deux fois le nombre de sommets de G.

Nous affirmons qu'il y a dans G une biclique non-induite de taille (k_1, k_2), si, et seulement si, il existe dans G une arête ab telle qu'il existe une biclique bipartie de taille (k_1, k_2) dans le graphe G_{ab}. Pour démontrer cette affirmation, supposons qu'il y a dans le graphe $G = (V, E)$, une biclique non-induite (X, Y) telle que $|X| = k_1$ et $|Y| = k_2$. Notez que $X \cap Y = \emptyset$. Soient $x \in X$ et $y \in Y$, alors clairement, puisque (X, Y) est une biclique, $xy \in E$ est une arête dans le graphe G.



Une arête ab et le graphe G_{ab} résultant. On peut voir le voisinage de a, le voisinage de b, et deux représentations du voisinage commun de a et b.

Intéressons-nous maintenant au graphe $G_{xy} = (A, B, E_{xy})$. Nous avons par construction du graphe G_{xy} que $X \subseteq N(y)$ est représenté dans G_{xy} par un ensemble $X_A \subseteq A$ de même taille k_1, ainsi que $Y \subseteq N(x)$ qui est représenté dans G_{xy} par un ensemble $Y_B \subseteq B$ de même taille k_2. Comme toutes les arêtes étaient présentes entre X et Y, alors il y a aussi toutes les arêtes entre X_A et X_B. Il en découle directement que (X_A, Y_B) est une biclique bipartie de taille k_1, k_2 dans le graphe G_{xy}.

Pour compléter notre affirmation, supposons maintenant que (X_A, Y_B) est une biclique bipartie de taille k_1, k_2 dans un graphe $G_{ab} = (A, B, E_{ab})$ obtenu à partir d'une arête $ab \in E(G)$ comme nous l'avons décrit précédemment. Sans perte de généralité, nous pouvons supposer que le sommet $a_A \in A$ est inclus dans la biclique, et donc que $a_A \in X_A$. Nous supposons de la même manière que $b_B \in B$ est inclus dans la bilclique et donc que $b_B \in Y_B$. Puisque dans G_{ab}, le sommet a_A est voisin de tous les sommets de B et que le sommet b_B est voisin de tous les sommets de A.

Pour tout sommet $z \in E(G)$ qui serait représenté deux fois, avec $z_A \in A$ et $z_B \in B$, alors on a que $z_A, z_B \notin E_{ab}$, ce qui nous assure que l'ensemble $X \subseteq V(G)$ représenté dans G_{ab} par X_A et $Y \subseteq V(G)$ représenté dans G_{ab} par Y_B sont disjoints dans G.

Par conséquent, (X, Y) est une biclique non-induite dans G avec la taille k_1, k_2.

Cela complète la preuve de notre affirmation, ainsi que la preuve du théorème. □

La preuve du théorème 4 a la conséquence suivante :

Corollaire 1. *Soit $G = (V, E)$ un graphe et k_1, k_2 deux entiers positifs. Le nombre de bicliques non-induites de taille k_1, k_2 dans le graphe G est égal à $\frac{1}{k_1 \cdot k_2}$ fois la somme des nombres de bicliques de taille k_1, k_2 que l'on trouvera dans les graphes G_{xy} obtenus à partir de toutes les arêtes $xy \in E$ du graphe G.*

Démonstration. Cela est dû au fait que, pour chaque biclique non induite (X, Y) de taille k_1, k_2 dans le graphe G, elle est en correspondance bijective directe avec une biclique de taille k_1, k_2 dans chaque graphe G_{xy}, pour toutes les arêtes xy telles que $x \in X$ et $y \in Y$. □

Si l'on se base sur le corolaire 1, la réduction du théorème 4 peut être combinée avec l'algorithme pour résoudre respectivement les problèmes de Biclique Bipartie et #Biclique bipartie afin de résoudre respectivement les problèmes Biclique non-Induite et #Biclique non-Induite.

Pour une instance avec un graphe G, et deux entiers positifs k_1 et k_2, alors on lance l'algorithme permettant, respectivement, de trouver ou de compter les nombres de bicliques de taille k_1, k_2 dans le graphe G_{xy} pour toutes les arêtes xy du graphe G. En additionnant les résultats obtenus sur chaque arête indépendamment, il suffit de diviser par $k_1 \cdot k_2$ pour retrouver le nombre de bicliques de taille k_1, k_2 dans le graphe G.

Pour analyser le temps d'exécution de cet algorithme, remarquez que le nombre de sommets de chaque G_{xy} peut être au pire des cas le double du nombre de sommets dans G. Sachant que l'on exécutera cet algorithme autant de fois qu'il y a d'arêtes dans le graphe, et que le nombre d'arêtes est inférieur à n^2.

Le temps d'exécution de l'algorithme sera donc en $O(n^2 \cdot 1.2691^{2 \cdot n})$, et donc, nous avons là un algorithme capable de résoudre #Bicliques non-Induites avec un temps d'exécution en $O^*(1.6107^n)$.

Théorème 5. *Le problème consistant à compter le nombre de bicliques non-induites dans un graphe peut être résolu en temps $O^*(1.6107^n)$.*

2.6 Conclusion

Nous avons donné un algorithme pour résoudre le problème de comptage d'#Ensemble Stable Bicoloré en temps $O^*(1.2691^n)$, en combinant les

techniques de branchement et réduction et de programmation dynamique. Cela nous permet de résoudre les problèmes de #Bipartie Biclique en temps $O^*(1.2691^n)$ et le problème de #Biclique non-Induite en temps $O^*(1.6107^n)$.

Par la suite, K. Kutzkov [66] a amélioré nos résultats à propos des bicliques en proposant un algorithme résolvant le problème de Bipartie Biclique en temps $O^*(1.2491^n)$.

Mise à part la question évidente consistant à chercher s'il existe un moyen d'améliorer nos temps d'exécution, il pourrait être intéressant d'étudier des moyens d'énumérer les ensembles stables bicolorés, plutôt que de simplement les compter. Il est intéressant de noter que l'énumération des bicliques d'un graphe dans un ordre lexicographique peut être effectuée avec un délai polynomial [29].

Troisième partie

Problèmes d'énumération

Chapitre 3

Coloration et étiquetage de graphes cubiques

3.1 Introduction

Les colorations de graphe sont des problèmes classiques dans la théorie des graphes. La conjecture des quatre couleurs, disant qu'il n'est besoin que de quatre couleurs au plus pour colorer proprement un graphe planaire, fut à l'origine d'un regain d'intérêt pour la théorie des graphes au cours du dernier siècle. D'un point de vue algorithmique, beaucoup de problèmes de coloration, que l'on cherche à colorer les sommets, les arêtes, à faire une coloration totale ou des colorations plus complexes telles L(2,1)-Labelling. Le problème consistant à trouver une coloration avec un nombre fixé de couleur est NP-complet. Le problème reste d'ailleurs NP-complet si la question est juste de savoir s'il existe une coloration avec un nombre de couleurs prédéterminées, comme on peut le voir dans de nombreux papiers traitant de ce sujet [36, 46, 54, 89].

Les algorithmes exacts à temps d'exécution exponentiels sont une approche intéressante pour les problèmes de coloration de graphes, et de nombreuses recherches ont été faites à ce sujet. L'un des résultats les plus important que l'on puisse citer est l'algorithme permettant de résoudre le problème du nombre chromatique avec un temps d'exécution de $O^*(2^n)$, grâce à la méthode d'inclusion-exclusion, qui fut présenté à la conférence FOCS 2006 par Björklund, Husfeldt [10] et Koivisto [61]. Cette approche pouvant aussi être utilisée pour établir un algorithme en $O^*(2^n)$ pour compter le nombre de k-colorations dans un graphe.

Le fait de chercher des k-colorations pour des petites valeurs de k a aussi été le sujet de beaucoup d'attention. En ce qui concerne les colorations de

sommets, un algorithme rapide pour $k = 3$ a été proposé par Beigel et Eppstein [5], et un autre le fut par Fomin et al. pour $k = 4$ [40]. Ces derniers ont aussi établi des algorithmes de comptage pour le nombre de colorations possibles des sommets avec 3, ou 4 couleurs.

En ce qui concerne les colorations d'arêtes, le cas où le nombre de couleurs $k = 3$ a été étudié dans [5] et un algorithme s'exécutant en $O^*(1.344^n)$ a été donné par Kowalik dans [62].

Un algorithme exact pour résoudre le problème de L(2,1)-LABELLING de *span* k a été montré par Kratochvil et al. dans [63], permettant par exemple pour $k = 4$ d'avoir un temps d'exécution de $O^*(1.3161^n)$.

L'une des questions qui semblent naturelles lorsque l'on a un problème sur les graphes est celle de déterminer une borne combinatoire sur le nombre de solutions que l'on peut trouver au maximum dans un graphe G avec n sommets. L'une des méthodes permettant d'obtenir ce genre de bornes est de construire un algorithme qui énumérerait toutes les solutions existantes. Un exemple bien connu est l'algorithme de branchement qui permet l'énumération de tous les ensembles stables maximals par inclusion dans un graphe et qui permit d'établir que le nombre d'ensembles stables maximals par l'inclusion dans un graphe à n sommets est inférieur à $3^{\frac{n}{3}}$. À l'origine, Moon et Moser avaient démontré en 1965 qu'il y a au plus $3^{\frac{n}{3}}$ cliques maximales dans un graphe à n sommets [76].

Un autre résultat important est la borne en $O^*(1.7159^n)$ pour le nombre d'ensembles dominants minimaux par inclusion dans un graphe à n sommets, établie par Fomin et al. [43], mais nous y reviendrons plus longuement dans un autre chapitre.

Dans ce chapitre, je vais vous présenter des algorithmes de branchement permettant d'énumérer le nombre de colorations d'arêtes avec 3 couleurs, ainsi que la coloration particulière L(2,1)-LABELLING pour des couleurs de 0 à 5. Vous remarquerez que ces colorations ne sont possibles que si le degré maximum dans le graphe est $\Delta(G) = 3$. Pour cette raison, nous travaillerons essentiellement sur des graphes cubiques (ou 3-réguliers).

Grâce au théorème de Vising [95], nous savons que le nombre minimum de couleurs nécessaires pour colorer les arêtes d'un graphe est très proche de son degré maximum. Pour être précis, on a exactement que $\Delta(G) \leq \mathcal{X}'(G) \leq \Delta(G) + 1$. Nous savons aussi, grâce à Rosenfeld, que pour tout graphe avec degré maximum égal à 3, ou $\Delta(G) = 3$, alors il y a une coloration totale de G avec 5 couleurs.

Nous nous intéressons à pouvoir déterminer l'existence, le nombre existant, et à pouvoir faire une énumération exhaustive des colorations d'arêtes et $L(2,1) - Labelling$. Bien entendu, le fait de pouvoir énumérer les solu-

tions, nous permet de les compter, et pouvoir les compter nous permet d'en déterminer l'existence.

3.2 Algorithmes d'énumération pour graphes cubiques.

Nous allons travailler sur des graphes cubiques, c'est-à-dire des graphes où tous les sommets du graphe sont de degré 3.

3.2.1 Coloration d'arêtes avec 3 couleurs.

Je vais vous présenter ici un algorithme de branchement capable d'énumérer les colorations d'arêtes d'un graphe donné qui ne nécessitent que 3 couleurs.

L'algorithme que nous avons développé consiste en une procédure récursive (que l'on appellera **EnumCol**), aidée de deux petites routines **color** et **extend**.

La procédure **EnumCol** prend en entrée un graphe cubique connexe G, ainsi qu'un sous-ensemble $\mathcal{S} \subseteq E(G)$ des arêtes de G déjà colorées tel que, pour toute arête $e \in \mathcal{S}$, alors la couleur $c(e) \in \{1, 2, 3\}$ est connue. La procédure **EnumCol** énumère toutes les colorations d'arêtes de G avec 3 couleurs qui sont des extensions de la coloration d'arêtes donnée par $\mathcal{S}$.

Pour énumérer toutes les 3-colorations du graphe G, nous commençons par choisir une arête $e \in E(G)$ du graphe et par appeler consécutivement la procédure **EnumCol** pour $\mathcal{S} = \{e\}$ et $c(e) = 1$,$c(e) = 2$ et $c(e) = 3$.

Théorème 6. *Notre algorithme énumère toutes les 3-colorations d'arêtes d'un graphe cubique connexe avec un temps d'exécution en* $O^*(2^{\frac{5n}{8}})$.

Démonstration. Afin de démontrer notre théorème, intéressons-nous à la procédure **EnumCol**. Dans la première étape, nous essayons d'étendre la coloration des arêtes du graphe sans utiliser de branchement, en utilisant la simple constatation que, s'il existe dans le graphe un sommet dont deux des arêtes sont déjà colorées, alors il n'existe qu'une solution possible pour colorer la troisième.

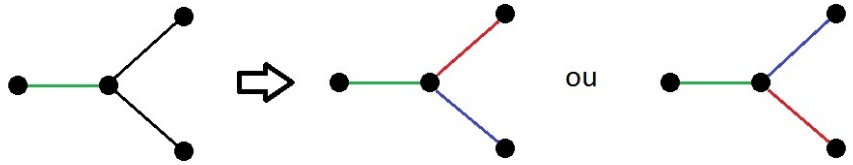
Dans la deuxième étape, nous choisissons un sommet v du graphe adjacent à exactement une arête e_1 déjà colorée. Nous effectuons alors un branchement sur e_2, l'une des deux arêtes non colorées adjacentes au sommet v. Clairement, il y a pour cette arête deux couleurs possibles au plus. On peut aussi facilement deviner que cette étape sera suivie d'une étape 1 pour colorer e_3,

Algorithme `EnumCol`$(G = (V, E), \mathcal{S}, c)$**.**
Entrée : Un graphe cubique $G = (V, E)$,
Un ensemble $\mathcal{S} \subseteq E$ des arêtes déjà colorées,
Une fonction $c : \mathcal{S} \to \{1, 2, 3\}$ qui donne la couleur de chaque arête.
Résultat : un ensemble de fonctions $c : E \to \{1, 2, 3\}$ représentant toutes les colorations possibles du graphe G avec 3 couleurs.

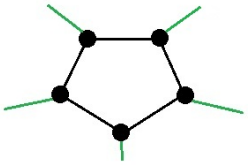
1 : Tant que, il existe un sommet $x \in V$ tel que $|E(x) \cap \mathcal{S}| = 2$ alors :
Soit $xy = E(x) \backslash \mathcal{S}$ l'arête adjacente à x pas encore colorée,
Soient $\{\alpha, \beta\} = \{c(e) | e \in \mathcal{S} \cap E(x)\}$ et $\gamma = \{1, 2, 3\} \backslash \{\alpha, \beta\}$,
Si $\gamma \in \{c(e) | e \in \mathcal{S} \cap E(y)\}$, alors STOP ;
Sinon, $c(xy) \leftarrow \gamma$;
2 : Si le graphe $H = G \backslash \mathcal{S}$ contient un composant F qui n'est pas un cycle, alors :
Choisir un sommet $x \in V(F)$ tel que $d_F(z) < 3$,
Choisir une arête $xy \in E(x) \backslash \mathcal{S}$,
Soit $\{\alpha, \beta\} = \{1, 2, 3\} \backslash \{c(e) | e \in E(x) \cap \mathcal{S}\}$,
Soit $res = \emptyset$ un ensemble de fonctions de $E(G) \to \{1, 2, 3\}$,
Si $\alpha \notin \{c(e) | e \in E(y) \cap \mathcal{S}\}$, alors :
$res \leftarrow res \cup EnumCol(G, \mathcal{S} \cup \{xy\}, c \cup \{c(xy) = \alpha\})$.
Si $\beta \notin \{c(e) | e \in E(y) \cap \mathcal{S}\}$, alors :
$res \leftarrow res \cup EnumCol(G, \mathcal{S} \cup \{xy\}, c \cup \{c(xy) = \beta\})$.
Retourner res comme résultat ;
3 : Si, il y a dans $H = G \backslash \mathcal{S}$ un composant F tel que :
F est un cycle de taille impaire et toutes les arêtes adjacentes à F ont la même couleur, alors STOP.
4 : Si, il existe dans $H = G \backslash \mathcal{S}$ une arête uv telle que $c(E(u) \cap \mathcal{S}) \neq c(E(v) \cap \mathcal{S})$, alors :
Soit $\alpha \in \{1, 2, 3\} \backslash \{c(e) | e \in (E(u) \cup E(v)) \cap \mathcal{S}\}$
Retourner $EnumCol(G, \mathcal{S} \cup \{uv\}, c \cup \{c(uv) = \alpha\})$;
5 : Si, il y a dans $H = G \backslash \mathcal{S}$ un composant F qui est un cycle, et tel que, pour tout $x \in V(F)$ alors $c(E(x) \cap \mathcal{S}) = \alpha$, alors :
Soit $\{\beta, \gamma\} = \{1, 2, 3\} \backslash \alpha$, et soit $e \in E(F)$,
Soit $res = \emptyset$ un ensemble de fonctions de $E(G) \to \{1, 2, 3\}$,
$res \leftarrow res \cup EnumCol(G, \mathcal{S} \cup \{e\}, c \cup \{c(e) = \beta\})$;
$res \leftarrow res \cup EnumCol(G, \mathcal{S} \cup \{e\}, c \cup \{c(e) = \gamma\})$;
Retourner res comme résultat ;
6 : Si $\mathcal{S} = E(G)$ alors **Retourner** $\{c\}$;

Algorithme `EnumCol` pour EDGE 3-COLORING

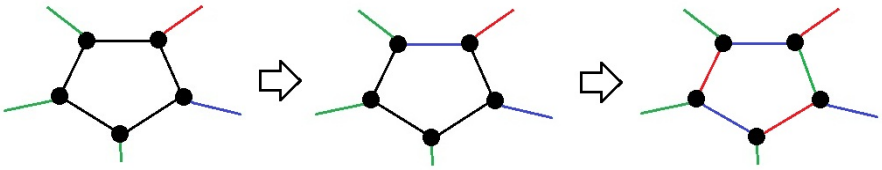
la deuxième arête non colorée de v. Notez que dans cette deuxième étape, le sommet v fait partie d'une composante $F \subseteq H = G \backslash \mathcal{S}$ qui n'est pas un cycle. Après un certain nombre d'utilisations des règles 1 et 2, nous obtenons donc une instance du problème telle que $H = G \backslash \mathcal{S}$ est une collection de cycles.



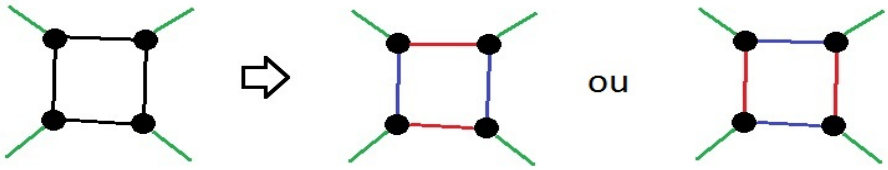
Dans la troisième étape, supposons que nous ayons un cycle de longueur impaire tel que toutes les arêtes qui entourent ce cycle soient déjà colorées avec la même couleur. Alors, dans ce cas, il est impossible de colorer ce cycle avec les couleurs prédéfinies dans $\mathcal{S}$, et on stoppe cette branche de l'exécution de l'algorithme.



Pour la quatrième étape, nous remarquons que si l'on a un cycle dont deux sommets consécutifs x et y ont leur troisième arête colorée avec deux couleurs différentes (disons sans perte de généralité que ces couleurs sont 1 et 2), alors, dans ce cas, il n'y a qu'une solution pour colorer l'arête $xy \in E$ et c'est de lui donner la couleur 3. Par suite, tout le cycle pourra être coloré sans avoir besoin de branchements.



Enfin, dans la dernière étape, il nous reste des cycles de longueur paire dont toutes les arêtes adjacentes sont déjà colorées par une même couleur. Pour chacun des cycles, il existe exactement deux solutions pour le colorer.



Intéressons-nous maintenant au temps d'exécution de cet algorithme. Remarquez que, pour chaque exécution successives des étapes 1 et 2, nous ajoutons au moins 2 sommets au graphe $G(\mathcal{S})$. Nous pouvons en déduire que la profondeur de l'arbre de recherche, jusqu'à obtenir que $H = G \backslash \mathcal{S}$ soit composé uniquement de cycles, est au plus de $\frac{n}{2}$.

Les branchements, lorsque nous n'avons plus que des cycles dans H, ne se font qu'une fois sur chaque cycle de longueur paire. Reste pour nous à estimer le nombre de ces cycles. G est un graphe connexe (et cubique) et, par la construction faite de $\mathcal{S}$, nous savons que $G(\mathcal{S})$ est un graphe connexe avec comme ensemble de sommets, $V(G(\mathcal{S})) \subseteq V(G)$ un sous-ensemble des sommets de G. Notez que $G(\mathcal{S})$ a uniquement des sommets de degré 1 et des sommets de degré 3. Nous appelons n_1 les sommets de degré 1 dans $G(\mathcal{S})$ et n_3 ses sommets de degré 3.

Il est clair que le graphe $G(\mathcal{S})$ possède $\frac{1}{2} \cdot (n_1 + 3 \cdot n_3)$ arêtes. Comme l'on sait que $G(\mathcal{S})$ est connexe, alors on peut en déduire que $\frac{1}{2} \cdot (n_1 + 3 \cdot n_3) \geq n_1 + n_3 - 1$, car on a besoin au minimum de $n - 1$ arêtes pour connecter n sommets. Sachant cela, et puisque dans $G(\mathcal{S})$ on a que $n = n_1 + n_3$, alors on peut voir que $\frac{1}{2} \cdot n + n_3 \geq n_1 + n_3 - 1$ et donc $n_1 - 1 \leq \frac{n}{2}$.

Les cycles de longueur paire que nous cherchons ne peuvent être constitués que de sommets qui, dans le graphe $G(\mathcal{S})$, sont les sommets de degré 1, et comme les longueurs de nos cycles sont paires, alors il nous faut au moins 4 arêtes par cycle. Il en advient que le nombre maximum de cycles que l'on peut avoir est de $\frac{n}{8}$.

Finalement, la profondeur de l'arbre de recherche avant de trouver les colorations en 3 couleurs du graphe G est au plus de $\frac{n}{2} + \frac{n}{8}$, et comme pour chaque branchement, nous avons deux solutions au plus, alors on peut dire que notre algorithme s'exécute avec un temps d'exécution de $O^*(2^{\frac{5n}{8}})$. □

Si l'on considère le fait que toutes les colorations possibles du graphe G avec trois couleurs se trouvent dans au moins une feuille de l'arbre de recherche de notre algorithme, alors nous pouvons conclure le corolaire suivant :

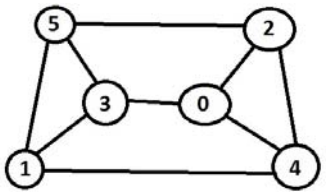
Corollaire 2. *Soit G un graphe cubique de n sommets, alors il y a dans G au plus $O^*(2^{\frac{5n}{8}})$ colorations des arêtes avec 3 couleurs.*

3.2.2 Étiquetage d'un graphe

Le problème sur lequel nous allons travailler maintenant s'appelle L(2,1)-Labelling, (Labelling signifie étiquetage en anglais).

L(2,1)-Labelling L'entrée du problème est un graphe $G = (V, E)$. Le problème consiste à déterminer une fonction $L : V(G) \to \mathbb{N}$ satisfaisant les propriétés suivantes :

- $\forall x \in V, \forall y \in V, xy \in E \Rightarrow |L(x) - L(y)| \geq 2$.
- $\forall x \in V, \forall y \in V$, Si $\exists z \in V$ tel que $xz \in E$ et $yz \in E$ alors $L(x) \neq L(y)$.



L(2,1)-Labelling sur un graphe cubique

Le problème L(2,1)-Labelling peut être vue comme un cas particulier d'un problème plus général, le problème de L(S)-Labelling.

L(S)-Labelling L'entrée du problème est un graphe $G = (V, E)$, et une suite d'entiers $S = \{l_1, \cdots, l_t\}$. Le problème consiste à déterminer une fonction $L : V(G) \to \mathbb{N}$ tel que :

- $\forall x \in V, \forall y \in V$ et pour tout $i \leq t$. Si les sommets x et y sont à distance i dans le graphe G, alors $|L(x) - L(y)| \geq l_i$.

Avec cette définition, on voit que le problème classique de coloration propre consistant à colorer les sommets d'un graphe de façon à ce que deux sommets adjacents soient de couleurs différentes peut être définit comme étant un L(1)-Labelling.

Dans notre cas, le graphe en entrée sera un graphe cubique, et nous ne chercherons que les étiquetages qui utilisent six couleurs (de 0 à 5). Notre algorithme, qui énumère tous les L(2,1)-Labelling, fonctionne comme un algorithme de branchement classique. Notre première observation sera que les couleurs n'ont pas une influence symétrique. Utiliser les couleurs 0 ou 5 sur un sommet ne bloque que deux couleurs pour le voisinage, alors que les couleurs $1, 2, 3$ et 4 bloquent chacune trois couleurs. Cela motivera la première règle de branchement de notre algorithme.

Notre algorithme de branchement se déroule principalement de la manière qui suit : Nous cherchons un sommet déjà étiqueté dont certains des voisins ne sont pas encore étiquetés. La partie des sommets déjà étiquetés dans le graphe au cours de l'exécution de l'algorithme forme donc une composante connexe. En considérant toutes les possibilités d'étiqueter ses voisins, l'algorithme va alors brancher sur plusieurs sous-problèmes correspondants. Ce que nous allons faire, c'est donc surtout de choisir de la meilleure façon possible, et à partir de quel sommet, nous souhaitons étendre l'étiquetage, et de quelle manière. Ces choix sont importants pour nous permettre d'établir la meilleure borne possible sur le nombre d'étiquetages possibles de notre graphe. Au fur et à mesure de la description des règles de branchement de l'algorithme, nous donnerons les équations linéaires correspondantes nécessaires au calcul du temps d'exécution.

Pour analyser le temps d'exécution de notre algorithme, nous allons utiliser la méthode *measure and conquer*, mais d'une façon différente de ce que l'on a fait dans le chapitre 2. Cette fois, nous n'utiliserons qu'un seul poids ϵ que nous attribuerons, au cours de l'exécution, à des sommets qui dans le déroulement de l'algorithme, acquerront une propriété particulière. Pour toute instance G de la procédure récursive, le graphe G étant déjà partiellement étiqueté, nous considèrerons la mesure suivante :

$$\mu(G) = n - \epsilon \cdot min(2, |\tilde{\mathrm{M}}|)$$

avec dans cette formule :

- n est le nombre de sommets dans le graphe G partiellement étiqueté qui ne sont pas encore étiquetés.
- Soit X l'ensemble des sommets étiquetés avec $\{1, 2, 3, 4\}$, et soit Y l'ensemble des sommets non-étiquetés. (X, Y) forme la partition d'un graphe bipartie $\tilde{\mathrm{G}}$, et $\tilde{\mathrm{M}}$ représente un couplage induit (en anglais, induced-Matching) de taille maximum.
- ϵ est une constante réelle choisie dans l'intervalle $\epsilon \in [0, 1]$.

Le couplage $\tilde{\mathrm{M}}$ nous permet d'estimer le nombre de branchements effectués avec la règle $[B1]$, qui est bien meilleure et arrive beaucoup plus souvent que les autres. L'exécution de notre algorithme peut être représentée par un arbre de recherche, dans lequel le temps pour construire les fils d'un nœud donné est polynomial. Le temps d'exécution de l'algorithme complet est donc un polynôme, multiplié par le nombre de nœuds dans l'arbre de recherche. Notre but est de donner une borne supérieure au nombre de feuilles que peut avoir l'arbre de recherche, à l'aide de la mesure $\mu = \mu(G)$.

Je vais maintenant vous décrire en détail les règles de branchement de l'algorithme, étape par étape. Les règles de branchement s'appliquent dans

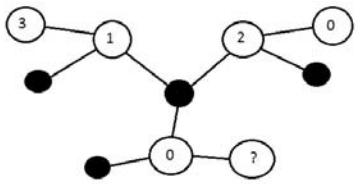
l'ordre de la description, cela signifie par exemple que, si j'applique la règle $[B2]$, c'est parce que la règle $[B1]$ ne pouvait pas être utilisée, et ainsi de suite. Si l'instance définie par le graphe G n'a aucuns sommets déjà étiquetés, alors nous choisissons deux sommets adjacents, et nous leurs attribuons les vingt combinaisons possibles d'étiquetage, amorçant ainsi notre algorithme avec vingt sous-problèmes. Nous étiquetterons ensuite les sommets en étendant, à partir de la partie étiqueté, l'ensemble des sommets déjà étiquetés d'un graphe formant un sous-graphe connexe. Soit $c : V(G) \to \{0, \cdots, 5\}$ la fonction d'étiquetage des sommets de G. Nous procédons comme suit :

B0 : Si, il existe un sommet v dont les trois voisins u_1, u_2, u_3 sont déjà étiquetés, alors :

B0a : L'étiquetage du sommet v est forcé, et on doit juste lui donner sa couleur. La règle de branchement correspondante est :

$$T(\mu) = T(\mu - 1 + \epsilon)$$

B0b : Dans un certain cas bien particulier, il peut y avoir deux possibilités pour étiqueter le sommet v. Supposons par exemple que $c(u_1) = 1$, $c(u_2) = 2$ et $c(u_3) = 0$ (respectivement $c(u_1) = 4$, $c(u_2) = 3$ et $c(u_3) = 5$), il existe dans ce cas deux possibilités pour étiqueter v. Seulement, nous savons, puisque l'ensemble des sommets étiquetés forme un graphe connexe, que les voisins de v ont chacun au moins 1 sommet déjà étiqueté (en fait, nous savons même que le voisin de u_1 est étiqueté 3, et que le voisin de u_2 est étiqueté 0, sinon l'étiquette de v est forcée).



Le fait de choisir une étiquette au sommet v aura pour conséquence de forcer la couleur des voisins pas encore étiquetés des sommets u_1 et u_2 (nous expliquerons pourquoi dans la règle **B1a**). La règle de récurrence correspondante est :

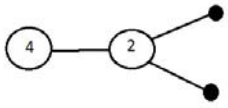
$$T(\mu) = 2 \cdot T(\mu - 3 + \epsilon)$$

B1 : Si, il y a dans le graphe, un sommet pas encore étiqueté dont l'un des voisins a une étiquette $1, 2, 3$ ou 4, alors nous avons trois possibilités :

B1a : Supposons qu'il y ait un sommet v non-encore étiqueté avec un voisin u étiqueté $1, 2, 3$ ou 4, et supposons que les deux autres voisins de u aient déjà une étiquette. Dans ce cas, il n'existe qu'une seule possibilité pour l'étiquette du sommet v. On peut donc étiqueter le sommet v par une règle de réduction, sans nécessiter de branchement. Cette règle fait décroître le nombre de sommets non-étiquetés de 1 et peut faire décroître la taille de $\tilde{\mathrm{M}}$ de 1. La règle de réduction correspondante est :

$$T(\mu) = T(\mu - 1 + \epsilon)$$

B1b : Soit $u'u \in E$ une arête du graphe G telle que $c(u') = 4$ (respectivement, $c(u') = 1$), et $c(u) = 2$ (respectivement, $c(u) = 3$). Soient x, y les deux voisins de u qui n'ont pas encore d'étiquettes.



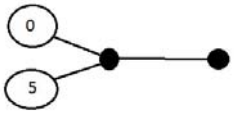
Il y a deux possibilités pour étiqueter x et y. Dans ces deux cas, le nombre de sommets sans étiquettes décroît de 2, et la taille de $\tilde{\mathrm{M}}$ peut décroître de 1. La règle de récurrence correspondante est :

$$T(\mu) = 2 \cdot T(\mu - 2 + \epsilon)$$

B1c : Soit un sommet u étiqueté $1, 2, 3$ ou 4 avec deux voisins non-encore étiquetés x et y, et tel que l'étiquette de son voisin étiqueté u' ne correspond pas à l'un des cas décrits dans l'étape **B1b**. Dans ce cas, il existe deux solutions pour étiqueter les sommets x et y. Chacune de ces deux solutions nécessitera d'avoir l'un des deux sommets, x ou y, qui devra prendre une étiquette $1, 2, 3$ ou 4. Ce qui fait que la taille de $\tilde{\mathrm{M}}$ ne décroîtra pas. La règle de récurrence correspondante est :

$$T(\mu) = 2 \cdot T(\mu - 2)$$

B2 : *À partir de cette règle, tous les sommets qui n'ont pas encore d'étiquettes ne sont voisins qu'avec des sommets portant les étiquettes* 0 *ou* 5*, et le couplage $\tilde{M}$ est vide.* Soit un sommet x sans étiquettes, adjacent à deux sommets u et v étiquetés respectivement 0 et 5. Soit y le voisin non-encore étiqueté de x.
Il y a deux possibilités pour étiqueter x et y et ces deux possibilités nécessitent de donner à y une étiquette parmi $1, 2, 3$ ou 4, augmentant

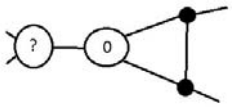


la taille de $\tilde{\mathrm{M}}$ de 1. Dans le pire des cas, y avait déjà ses deux autres voisins étiquetés, et dans ce cas, la taille de $\tilde{\mathrm{M}}$ ne change pas. La règle de récurrence correspondante est :

$$T(\mu) = 2 \cdot T(\mu - 2)$$

B3 : *Maintenant, tous les sommets non-encore étiquetés sont adjacents à au plus un sommet étiqueté.*

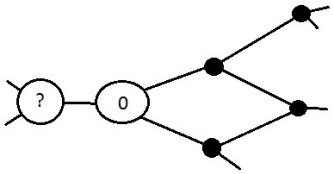
B3a : Supposons qu'il existe deux sommets adjacents x et y avec un voisin commun v étiqueté 0 ou 5.



Il y a alors au plus 4 possibilités pour étiqueter X et Y, et dans chacune de ces possibilités, x ou y (ou les deux) doi(ven)t être étiqueté(s) parmi $1, 2, 3$ ou 4. La taille de $\tilde{\mathrm{M}}$ augmente de 1 pour chacune des quatre possibilités. La règle de récurrence correspondante est :

$$T(\mu) = 4 \cdot T(\mu - 2 - \epsilon)$$

B3b : Supposons que nous ayons deux sommets x et y non-adjacents et non étiquetés avec un voisin commun v étiqueté 0 ou 5. Supposons aussi que ces deux sommets x et y ont un autre voisin commun a qui n'est pas encore étiqueté non plus.

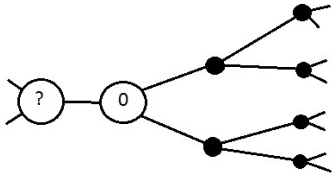


Étant données les règles précédentes, et le fait que le graphe est cubique, nous pouvons aussi dire que x a un autre sommet non étiqueté b, et que y a un autre sommet non étiqueté b'. Il existe 6 possibilités pour étiqueter x et y. Dans ces 6 possibilités, au moins 4 vont forcer

la valeur de a, ou déterminer que l'étiquetage ne peut pas être étendu. Dans ces 4 possibilités, il y aura toujours au moins b ou b' qui sera forcé à son tour. Finalement, la règle de récurrence correspondante est :

$$T(\mu) = 4 \cdot T(\mu - 4) + 2 \cdot T(\mu - 2 - \epsilon)$$

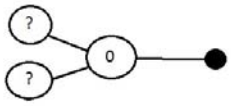
B3c : Soient x et y deux sommets non-encore étiquetés, avec un voisin commun u étiqueté 0 ou 5.



Dû aux règles précédentes, nous pouvons affirmer que les sommets x et y ont chacun deux autres voisins non-étiquetés, que nous appellerons a et b pour x, et a' et b' pour y. Il existe 6 possibilités pour étiqueter x et y, dont au moins 4 qui donneront à x et y une étiquette parmi $1, 2, 3$ ou 4. La taille de $\tilde{\text{M}}$ dans ces cas-là augmentera de 2. La récurrence correspondante est alors :

$$T(\mu) = 4 \cdot T(\mu - 2 - 2 \cdot \epsilon) + 2 \cdot T(\mu - 2 - \epsilon)$$

B4 : *Nous avons maintenant que chaque sommet étiqueté est voisin d'au plus un sommet non étiqueté.* Soit v un sommet non étiqueté, adjacent à un sommet u étiqueté 0 ou 5 dont les deux autres voisins sont étiquetés.



Si l'étiquette de v n'est pas forcée, alors il n'existe que deux possibilités pour étiqueter v, et au moins une d'entre elles est avec une étiquette parmi $1, 2, 3$ ou 4. La règle de récurrence correspondante est alors :

$$T(\mu) = T(\mu - 1) + T(\mu - 1 - \epsilon)$$

Notez que le fait que l'algorithme soit correct est lié au fait qu'il s'agit là d'une étude exhaustive des cas et que l'on branche sur toutes les possibilités existantes.

Lorsque l'on résout les récurrences en utilisant $\epsilon = 0.819531$, alors on obtient que $T(\mu) \leq O^*(1.7790^{\mu})$. En observant que, avec notre mesure, nous avons toujours que $\mu \leq n$, nous pouvons alors affirmer le théorème suivant :

Théorème 7. *Notre algorithme de branchement énumère tous les étiquetages L(2,1) possible d'un graphe cubique donné avec un temps d'exécution de* $O^*(1.7790^n)$.

Puisque nous avons énuméré tous les étiquetages avec étiquettes jusqu'à 5 dans notre algorithme de branchement, et que nous pouvons affirmer que chaque solution possible se trouve dans au moins une feuille de l'arbre de recherche, alors nous avons aussi le corollaire suivant :

Corollaire 3. *Le nombre d'étiquetages L(2,1) dans un graphe cubique de n sommets est inférieur à* $O^*(1.7790^n)$.

3.3 Bornes inférieures

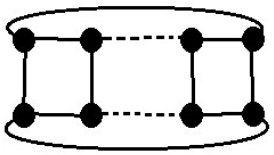
Les algorithmes que nous venons de décrire nous permettent d'obtenir une borne supérieure sur le nombre de solutions à nos problèmes. Mais à quel point cette borne supérieure est-elle surestimée ? Pour essayer de s'en faire une idée, nous allons donner maintenant une borne inférieure pour ces même problèmes.

3.3.1 Coloration d'arêtes

Vous remarquerez que notre algorithme qui énumère les colorations d'arêtes sur graphes cubiques s'appuie fortement sur le fait que le graphe est connexe, et que tous les sommets du graphe sont de degré 3. À aucun moment nous ne prenons en compte l'éventualité d'un sommet de degré 1 ou 2. Particulièrement, notre algorithme ne fonctionne pas si le graphe est juste de degré maximum $\Delta(G) \leq 3$. Par exemple, pour un chemin de longueur n, le nombre de colorations d'arêtes possibles avec trois couleurs est $3 \cdot 2^{n-2}$, et pour une union disjointe de $k_{3,3}$, il existe $12^{\frac{n}{6}}$ solutions. Nous allons présenter maintenant une borne inférieure pour le nombre de colorations d'arêtes à 3 couleurs pour les graphes cubiques connexe.

Théorème 8. *Il peut y avoir au moins* $2^{\frac{n}{2}}$ *colorations d'arêtes différentes dans un graphe cubique connexe.*

Démonstration. Soit $n = 2 \cdot r$, et soit G un graphe composé de deux cycles de longueur r reliés entre eux par un couplage entre les sommets, de façon à ce que le couplage respecte l'ordre d'apparition des sommets dans leurs cycles respectifs. Pour imager, le graphe G prend alors l'apparence d'une *roue de hamster*.



Si l'on cherche à colorer les arêtes du premiers cycle c_r, alors nous avons 3 solutions pour colorer la première arête, puis à chaque fois 2 solutions pour colorer l'arête suivante. Sachant que la toute dernière arête du cycle c_r peut se retrouver forcée, le cycle a $3 \cdot 2^{r-2} = \frac{3 \cdot 2^r}{4}$ colorations d'arête à trois couleurs possibles.

Pour toutes ces colorations du premier cycle, toutes les arêtes du couplage entre les deux cycles c_r et c'_r sont forcées, et les couleurs des arêtes du cycle c'_r sont forcées à leur tour. (En fait, il y a exactement six cas pour lesquels le cycle c'_r a deux possibilités de colorations, cela correspond aux cas où toutes les arêtes du couplage entre les deux cycles sont forcées à une même couleur, et donc aux cas où les cycles eux-même ne sont colorés qu'avec 2 couleurs.)

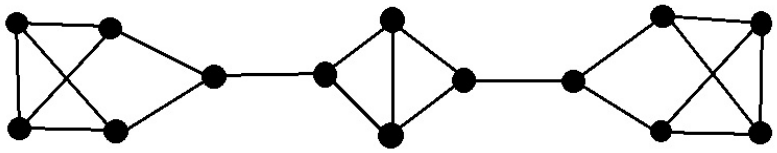
Le graphe G a donc au moins $2^{\frac{n}{2}}$ colorations d'arêtes avec 3 couleurs. □

3.3.2 Étiquetage L(2,1) avec étiquettes de 0 à 5

Nous allons essayer maintenant de construire un graphe G cubique et connexe dont le nombre d'étiquetages possibles est au moins de $2^{\frac{2 \cdot n}{7}}$.

Théorème 9. *Il peut y avoir au moins $2^{\frac{2 \cdot n}{7}}$ étiquetages dans un graphe cubique connexe.*

Démonstration. Soit F un graphe obtenu à partir d'un K_4 dont les sommets sont l'ensemble $\{x_1, x_2, x_3, x_4\}$, en subdivisant l'arête x_1x_2 pour y ajouter un nouveau sommet x_0. Considérons que nous avons $F_1, F_2, \cdots, F_{2t}$ qui sont $2 \cdot t$ copies disjointes du graphe F. Construisons un cycle C_{4t} avec $4 \cdot t$ sommets dans l'ensemble $\{v_1, v_2, \cdots, v_{4t}\}$ dans lequel on ajoutera les arêtes $v_{4i}v_{4i-2}$. Enfin, pour finir la construction de notre graphe, nous relierons chaque sommets v_{2i-1} du cycle C_{4t} à un sommet x_0 du sous-graphe F_i correspondant.



Observons maintenant que nous avons la possibilité d'étiqueter le cycle C_{4t} de la manière qui suit : pour tout $i = 1, 2, \cdots, t$ alors $c(v_{4i-3}) = 2$, $c(v_{4i-2}) = 0$, $c(v_{4i-1}) = 3$ et $c(v_{4i}) = 5$. Ce cycle d'étiquettes nous permet sans difficulté d'étiqueter tout graphe de taille $4 \cdot i$. Remarquez que l'étiquetage reste valide si l'on inverse les 0 et les 5, ou si l'on inverse les 2 et les 3.

Intéressons nous maintenant aux étiquettes que peuvent prendre les sous-graphes F_i. Si le voisin j de x_0 est étiqueté à 2, alors les voisins de j étant étiquetés 0 et 5, l'étiquette de x_0 se retrouve forcée à 4. Les sommets x_1 et x_2 peuvent alors être étiquetés de deux manière possibles, l'un prenant l'étiquette 1 et l'autre l'étiquette 0. Les sommets x_3 et x_4 à leurs tours peuvent être étiquetés de deux manières possibles, l'un prenant l'étiquette 3 et l'autre l'étiquette 5. Dans l'ensemble, on voit que si j a comme étiquette 2, alors il y a 4 façon différentes d'étiqueter le sous-graphe F.

De même, si le voisin j de x_0 est étiqueté 3, alors x_0 sera forcé à 1. Dans ce cas, les valeurs que pourront prendre x_1 et x_2 seront 4 et 5, et les valeurs pour x_3 et x_4 seront 0 et 2.

Pour tous les F_i, il y a quatre possibilités de les étiqueter. Comme nous avons dans notre graphe $2 \cdot t$ sous-graphes F, et que les étiquetages de ces sous-graphes sont indépendants, nous pouvons conclure qu'il existe au moins 4^{2t} possibilités d'étiqueter le graphe. Comme $2t = \frac{n}{14}$, alors nous avons que le nombre d'étiquettes dans un graphe cubique connexe peut être au moins $2^{\frac{2n}{7}}$. □

3.4 Conclusion

Nous avons présenté, dans ce chapitre, un algorithme donnant toutes les 3-colorations d'arêtes d'un graphe cubique connexe en temps $O^*(2^{\frac{5n}{8}})$, et montré qu'il existe des graphes cubiques connexes dont le nombre de 3-colorations d'arêtes différentes est au moins $2^{\frac{n}{2}}$. Nous présentons aussi un algorithme permettant d'énumérer les étiquetages $L(2,1)$ dans un graphe cubique en temps $O^*(1.7790^n)$, et montrons qu'il existe des graphes cubiques avec au moins $2^{\frac{2 \cdot n}{7}}$ étiquetages $L(2,1)$ différents.

Par la suite, S. Bessy et F. Havet [8] ont amélioré nos résultats, donnant notamment un algorithme pour énumérer les 3-colorations d'arêtes dans un graphe cubique qui s'exécute en temps $O^*(2^{\frac{n}{2}})$. Ces résultats sont actuellement en cours de publication dans *Journal of Combinatorial Optimization*.

Nos résultats concernant essentiellement les graphes cubiques, il serait intéressant de chercher comment étendre nos résultats à d'autres classes de graphe, comme par exemple, dans un premier temps, les graphes 4-réguliers, ou même les graphes d-réguliers pour un $d \geq 4$ fixé. Il pourrait être intéressant

aussi de chercher à généraliser les résultats sur le problème d'ETIQUETAGE $L(2,1)$ aux problèmes des ETIQUETAGES $L(p,q)$.

Chapitre 4

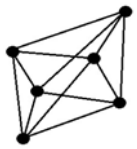
Ensembles dominants minimaux par inclusion

4.1 Introduction

Les questions combinatoires du type *« quel est le nombre maximum de sous-ensembles de sommets que je peux trouver avec une certaine propriété ? »* sont très intéressantes en informatique, et spécialement dans la branche des algorithmes exacts à temps d'exécution exponentiels au rapport de l'entrée [38]. La question a été récemment étudiée pour les ensembles coupe cycles (FEEDBACK VERTEX SET), les séparateurs minimaux, les cliques maximales, et les coupe cycles dans les tournois [39, 45, 48]. L'exemple le plus célèbre, devenu même un classique dans le domaine, est l'algorithme de Moon et Moser [76] qui permet d'établir que le nombre maximum de cliques dans un graphe de n sommets est de l'ordre de $3^{\frac{n}{3}}$, induisant directement un résultat équivalent pour le nombre d'ensembles stables maximals par inclusion. Bien que dans le document original [76] la preuve de ce théorème soit faite par induction, il n'est pas difficile de transformer cette preuve en un algorithme de branchement permettant d'énumérer tous les ensembles stables maximals par inclusion dans le graphe avec un temps d'exécution de $O^*(3^{\frac{n}{3}})$. Ce résultat fut d'ailleurs utilisé par Lawler [68] afin de construire un algorithme de coloration de graphes qui, pendant une vingtaine d'années, fut le meilleur algorithme connu pour résoudre ce problème. Un algorithme permettant de résoudre le problème de coloration de graphes plus rapidement fut obtenu par Eppstein [33] qui améliora la borne supérieure du nombre d'ensembles stables maximals par inclusion de petites tailles.

Le nombre de documents parlant de domination d'un graphe approche plusieurs milliers, et de nombreux livres et études bien connues sont dédiés

à ce sujet. On citera notamment le livre de T.W. Haynes, S.T. Hedetniemi et P.J. Slater [51] sur les notions fondamentales de domination d'un graphe. Il est surprenant qu'une réponse à la question du même genre que la question de Moon et Moser, à savoir *« quel est le nombre maximum d'ensembles dominants minimaux par inclusion dans un graphe ? »* n'ait été établie que très récemment. Fomin, Grandoni, Pyatkin et Stepanov [43] ont montré que le nombre maximum d'ensembles dominants minimaux par inclusion dans un graphe de n sommets est au plus 1.7159^n. Dans le même document, ils utilisent ce résultat afin d'obtenir un algorithme avec un temps d'exécution en $O(2.8718^n)$ pour le problème du NOMBRE DOMATIQUE (DOMATIC NUMBER en anglais). De plus, ils donnent aussi un exemple de graphe dont le nombre d'ensembles dominants minimaux par inclusion est de l'ordre de 1.5704^n, démontrant qu'il existe une famille de graphes dont le nombre d'ensembles dominants minimaux par inclusion peut être de 1.5704^n au moins, et établissant ainsi une borne inférieure.



Un graphe de 6 sommets avec 15 ensembles dominants minimaux par inclusion

Notre intérêt pour cette question combinatoire est renforcé par cet écart important entre la meilleure borne supérieure connue et la meilleure borne inférieure connue pour les graphes en général, et ce que cela implique pour les algorithmes exacts à temps d'exécution exponentiels qui serviraient à résoudre ce problème. Nous fournirons, dans ce chapitre, des bornes supérieures et inférieures pour le nombre d'ensembles dominants minimaux par inclusion pour plusieurs classes de graphe particulières. Les bornes supérieures que nous fournirons seront fortement liées aux propriétés structurelles des classes de graphe correspondantes. Typiquement, lorsque nous obtenons des bornes ajustées, c'est-à-dire des bornes telles que la borne inférieure et la borne supérieure se rejoignent, elles seront généralement démontrées par des arguments combinatoires. Sinon, nous aurons des bornes asymptotiques dont les valeurs seront démontrées par le temps d'exécution d'algorithmes de branchement. Ci-après un tableau de nos différents résultats.

En 2011, Kanté et al. [57] ont montré que sur les graphes *splits*, il y a un algorithme qui permet d'énumérer tous les ensembles dominants minimaux

Classes de graphes	Bornes inférieures	Bornes supérieures
Cas général [43]	1.5704^n	1.7159^n
Graphes cordaux	1.4422^n	1.6181^n
Graphes co-biparties	1.3195^n	1.5875^n
Graphes splits	1.4422^n	1.4656^n
Graphes d'intervalles propres	1.4422^n	1.4656^n
Co-graphes*	1.5704^n	1.5705^n
Graphes trivialement parfaits*	1.4422^n	1.4423^n

*Bornes inférieures et supérieures sur le nombre d'ensembles dominants minimaux par inclusion pour les classes de graphe. les classes de graphes avec des bornes ajustées sont signalé par **

par inclusion avec un temps d'exécution polynomial au rapport du nombre de ces ensembles. Autrement dit, leur algorithme s'exécute en temps polynomial au rapport de la sortie de l'algorithme. Dans leur document, ils n'ont pas cherché à étudier le nombre de ces ensembles.

Pour toutes les classes de graphe étudiées, nous avons obtenu un algorithme permettant d'énumérer les ensembles dominants minimaux par inclusion. Toutes nos preuves de bornes supérieures sur le nombre d'ensembles sont constructives et les algorithmes qui en découlent sont en fait des conséquences faciles. Il faut juste, dans certains cas, vérifier que les objets générés ont bien les propriétés souhaitées.

Lorsque la borne supérieure sur le nombre d'éléments est de la forme $O(c^n)$, alors cela implique un algorithme dont le temps d'exécution sera en $O^*(c^n)$.

4.2 Préliminaires

4.2.1 Définitions utiles

Un ensemble de sommets $D \subseteq V(G)$ est un ensemble dominant dans le graphe G si, pour tout sommet $v \in V(G)$, alors soit $v \in D$, soit il existe $x \in (N(v) \cap D)$.

Un ensemble de sommets $D \subseteq V(G)$ est un ensemble dominant minimal par inclusion si D est un ensemble dominant et que aucun sous-ensemble de D n'est un ensemble dominant. Autrement dit, pour tout sommet $v \in D$, l'ensemble $D \backslash \{v\}$ n'est pas un ensemble dominant dans G. Il n'est pas difficile de conclure que si D est un ensemble dominant, alors pour tout $v \in D$, il

existe un sommet $x \in V(G)$ tel que v est le seul sommet à dominer x. Ou encore $N(x) \cap D = \{v\}$. Dans ce cas, on appellera x le voisin privé de v. Notez que rien n'empêche que $x = v$, un sommet peut être son propre voisin privé.

Le nombre d'ensembles dominants minimaux par inclusion sera noté $\mu(G)$.

Un ensemble dominant minimal par inclusion dans un graphe composé de plusieurs composantes connexes est une union des ensembles dominants minimaux par inclusion de chaque composante connexe. Cela implique directement l'observation suivante :

Lemme 2. *Soit G un graphe composé de différentes composantes connexes $G_1, G_2, \cdots, G_t$, alors le nombre d'ensembles dominants minimaux par inclusion dans G est égal au produit des nombres d'ensembles dominants minimaux par inclusion de chaque composante connexe. On a donc $\mu(G) = \prod_{i=1}^{t} \mu(G_i)$.*

Pour chaque classe de graphe étudiée, nous redonnerons la définition et les caractéristiques principales au début de la section qui leurs correspondra. Toutes les classes de graphe que nous étudions dans ce chapitre peuvent être reconnues en temps linéaire, et sont assez proches les unes des autres si l'on considère les sous-graphes induits [15, 50].

Nous allons maintenant définir deux familles de graphes qui nous seront utiles pour donner des bornes inférieures sur le nombre d'ensembles dominants minimaux par inclusion dans certaines classes de graphe.

H_n désignera une famille de graphes contenant $3 \cdot k$ sommets et constituée de k triangles disjoints.

S_n désignera une famille de graphes contenant $3 \cdot k$ sommets et constituée d'une clique de $2 \cdot k$ sommets et d'un ensemble stable de k sommets tel que chaque sommet de l'ensemble stable a exactement deux voisins dans la clique, et que chaque sommet de la clique n'est adjacent qu'à un et un seul sommet dans l'ensemble stable.

On peut vérifier assez facilement que pour ces deux familles de graphes, $\mu(H_n) = \mu(S_n) = 3^{\frac{n}{3}} \approx 1.4422^n$. Pour plusieurs des classes de graphe que

nous étudierons dans ce chapitre, cette borne est la meilleure borne inférieure connue pour le nombre maximum d'ensembles dominants minimaux par inclusion.

4.2.2 Les algorithmes de branchement

Dans ce chapitre, nous utilisons des algorithmes de branchement afin de générer tous les sous-ensembles de sommets pouvant correspondre à des ensembles dominants minimaux par inclusion. L'exécution de ces algorithmes peut être vue et représentée comme un arbre de recherche dont les feuilles seraient toutes les résultats que donnera l'algorithme. Comme nous savons que nos algorithmes énumèrent de façon exhaustive les ensembles dominants minimaux par inclusion dans les classes de graphe correspondantes, nous pouvons sans risque affirmer que, pour tout ensemble dominant minimal par inclusion, il y a une feuille correspondante dans l'arbre de recherche de l'algorithme associé. Notez que toutes les feuilles ne contiennent pas forcément un ensemble dominant minimal par inclusion. Le nombre de feuilles dans l'arbre de recherche associé à nos algorithmes constitue donc une borne supérieure au nombre d'ensembles dominants minimaux par inclusion $\mu(G)$ dans les classes de graphes associées. La correction de nos algorithmes est liée au fait que, pour toutes feuilles de l'arbre de recherche, il est facile de vérifier, en temps polynomial, si le sous-ensemble de sommets correspondant est bien un ensemble dominant minimal par inclusion ou pas. Le temps d'exécution de nos algorithmes étant donc de l'ordre d'un polynôme associé au nombre de feuilles dans l'arbre de recherche associé.

Typiquement, tous les appels récursifs sont de la forme (G', D) où G' représente un sous-graphe induit du graphe d'origine constitué des sommets dont on n'a pas encore choisi s'ils doivent ou pas faire parte de l'ensemble dominant, et $D \subseteq V(G) \backslash V(G')$ représente l'ensemble des sommets qui ont été choisis pour faire partie de l'ensemble dominant dans tous les sous-problèmes correspondant à cet appel récursif. Notez que les sommets inclus dans D sont en dehors du graphe G'. Le tout premier appel ne contenant aucuns sommets déjà choisis dans l'ensemble dominant correspondra à $(G, \emptyset)$. À chaque étape des algorithmes, nous faisons des choix qui, pour les appels récursifs suivant, font décroître la taille de G', et, peut-être, accroître la taille de D. Nos algorithmes avancent de façon à ce que jamais un sommet de G' soit indispensable à dominer un sommet en dehors de G', ou autrement dit, lorsque nous ajoutons un sommet du graphe G' courant à l'ensemble D, alors son voisin privé sera toujours un des sommets du graphe G' courant.

4.3 Les graphes cordaux

Une corde est une arête qui relie deux sommets non consécutifs d'un cycle. Un graphe est *cordal* si tous les cycles de longueur au moins 4 présents dans ce graphe possèdent une corde. Autrement dit, le plus grand cycle induit présent dans ce graphe est un triangle, tout cycle plus grand que cela sera coupé par des arêtes. Un sommet v est appelé sommet *simplicial* si le voisinage de ce sommet $N(v)$ est une clique. On peut dire aussi que, pour tout sommet $x \in N(v)$, alors $N(v) \subseteq N(x)$. Tous les graphes cordaux ont au moins un sommet simplicial [50].

Remarquez que les familles de graphes H_n et S_n que nous avons définies dans la partie 4.2.1 sont des graphes cordaux, nous donnant ainsi un exemple de familles de graphes cordaux dont le nombre d'ensembles dominants minimaux par inclusion est au moins de $3^{\frac{n}{3}} \approx 1.4422^n$.

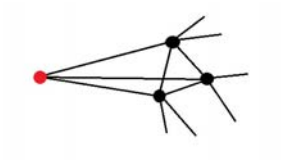
Théorème 10. *Un graphe cordal contient au plus $O(1.6181^n)$ ensembles dominants minimaux par inclusion.*

Démonstration. Soit une instance (G', D), nous appellerons v un sommet *déjà dominé* s'il existe un sommet $u \in N_G(v) \cap D$. Notre algorithme va choisir un sommet simplicial x dans le graphe G', un tel sommet existe toujours puisque le graphe G' est un graphe cordal. Si le sommet x est un sommet isolé, alors aucun branchement n'est nécessaire. Il faut et il suffit de vérifier si x est déjà dominé ou non, et de, respectivement, ne pas le prendre ou le prendre dans l'ensemble dominant afin de s'assurer qu'il sera dominé au final. Puis on supprime simplement x du graphe G' et on continue avec un autre sommet simplicial.

Nous considérerons donc à partir de cet instant que les sommets x simplicials dans le graphe G' que nous avons choisis possèdent au moins un voisin dans le graphe G'. Les actions que nous ferons ensuite vont dépendre à la fois du fait que x soit ou non déjà dominé, et du nombre de voisins qu'il a dans G'. Notez que, parmi les cas qui vont suivre, il n'y aura qu'un seul cas à la fois qui pourra correspondre à une situation donnée, et que c'est ce cas là qui sera exécuté par notre algorithme.

Cas 1 : le sommet x est déjà dominé. Nous allons alors brancher selon la possibilité que l'on a d'ajouter x à l'ensemble D des sommets qui feront partie de l'ensemble dominant minimal par inclusion, ou sur le fait de décider que le sommet x ne fera définitivement pas partie de l'ensemble dominant minimal par inclusion.

$x \in D$: Si l'on choisit d'inclure le sommet x dans l'ensemble dominant minimal par inclusion, alors comme le sommet x est déjà dominé, il



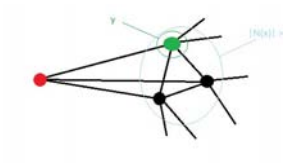
Le sommet x

a besoin d'avoir un voisin privé dans son voisinage $N(G')$ du graphe G'. Puisque le sommet x est simplicial, cela signifie que son voisinage est une clique, et donc, qu'aucun sommet dans $N(x)$, le voisinage de x, ne peut faire partie de l'ensemble dominant D à l'avenir. Sinon, il dominerait tous les sommets qui peuvent être les voisins privés de x, ce dernier se retrouvant donc dépourvu de voisin privé et l'ensemble D n'étant donc plus minimal par inclusion. Par conséquent, nous pouvons supprimer le sommet x et tout son voisinage $N(x)$, et appeler la récurrence $(G' \backslash N(x), D \cup \{x\})$. La taille de G' décroît alors d'au moins 2 sommets.

$x \notin D$: Si l'on choisit que le sommet x ne fera jamais partie de l'ensemble dominant D, alors, comme nous savons que le sommet x est déjà dominé, nous pouvons tout simplement nous contenter de le supprimer du graphe G'. Nous appellerons donc la récurrence $(G' \backslash \{x\}, D)$ et la taille de G' décroît donc de 1 sommet.

Nous avons donc deux sous-problèmes, avec des tailles réduites respectivement de 2 et de 1 sommet(s). Le vecteur de branchement qui correspondra à cette règle sera le vecteur $(2, 1)$.

Cas 2 : x n'est pas dominé, et $|N'_G[x]| \geq 2$. Soit y l'un des voisins de x dans le graphe G'. Nous allons brancher selon la possibilité d'ajouter y à l'ensemble dominant D ou de choisir définitivement qu'il n'en fera jamais partie.



Le sommet x et un voisin y

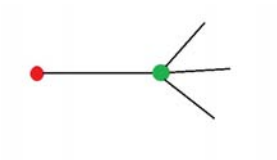
$y \in D$: Si jamais nous décidons d'ajouter le sommet y à l'ensemble dominant D, alors il dominera x et dominera en même temps tous les

autres voisins de x. En effet, comme x est un sommet simplicial, on a $N(x) \subseteq N(y)$ et cela signifie que tous les sommets dans le voisinage de x sont donc dominés. Sachant que tous les sommets de $V(G) \backslash V(G')$ sont déjà tous dominés, il n'existe plus aucune possibilité pour x d'avoir un voisin privé, et il ne pourra donc jamais faire partie de l'ensemble dominant D sans remettre en cause la minimalité de D. Cela nous permet donc de supprimer sans risques le sommet x, et nous pouvons appeler la récurrence $(G' \backslash \{x, y\}, D \cup \{y\})$. La taille de G' décroît alors de 2 sommets.

$y \notin D$: Si nous décidons que y ne fera pas partie de l'ensemble dominant D, la seule question qu'il nous reste à résoudre pour pouvoir supprimer y est de savoir comment il sera dominé dans la solution. Or, on sait que le sommet x, sommet simplicial dont le voisinage inclus le sommet y, n'est pas encore dominé lui non plus. Observons que toutes les possibilités qu'il nous restent pour dominer x ont la particularité de dominer y en même temps. Lorsque x sera dominé dans les prochaines étapes de l'algorithme, alors y le sera automatiquement. C'est avec cette certitude que le sommet x finira par être dominé au cours de l'exécution de l'algorithme, et que cela entraînera systématiquement la domination du sommet y, que nous pouvons décider de supprimer y du graphe G'. La récurrence que l'on appelle alors sera $(G' \backslash \{y\}, D)$. La taille du graphe G' décroît alors de 1.

Cette fois encore, nous avons deux sous-problèmes dont les tailles se réduisent respectivement de 2 et de 1 sommet(s). Le vecteur de branchement correspondant à cette règle sera le vecteur $(2, 1)$.

Cas 3 : x n'est pas dominé et n'a qu'un seul voisin y dans G'. Si x n'est toujours pas dominé, et que le seul voisin qui lui reste est le sommet y, alors il n'y a plus que deux solutions pour dominer x. Nous branchons alors sur ces deux solutions.



Le sommet x et son unique voisin y

$x \in D$: Si nous décidons que le sommet x doit se dominer lui-même, alors, dans le même temps, il domine son voisin y. Dans les étapes sui-

vantes de l'algorithme, y ne pourra jamais faire partie de l'ensemble dominant D sinon il recouvrirait tous les voisins privés de x et remettrait donc en cause la minimalité de l'ensemble dominant D. Les deux sommets x et y se retrouvant dominés, et leur appartenance à l'ensemble dominant D étant fixé, nous pouvons les supprimer tous les deux dans la suite du déroulement de l'algorithme. La récurrence que nous appellerons sera alors la récurrence $(G'\backslash\{x, y\}, D \cup \{x\})$, et la taille du graphe G' décroît alors de 2.

$y \in D$: Si nous décidons que le sommet x ne se domine pas lui-même, alors il doit être dominé par son unique voisin y. Le sommet y domine alors le sommet x et lui-même, et x devient alors l'un de ses voisins privés. Là encore, nous pouvons sans risque supprimer les deux sommets x et y du graphe G'. La récurrence qui sera alors appelée sera la récurrence $(G'\backslash\{x, y\}, D \cup \{y\})$, et là aussi, la taille du graphe G' décroît de 2.

Dans les deux sous-problèmes que nous avons cette fois, la taille du graphe se réduit de 2. Le vecteur de branchement correspondant à ce troisième cas sera donc le vecteur $(2, 2)$.

Avec ces trois cas, nous avons traité la totalité des éventualités qui peuvent survenir autour d'un sommet simplicial. Comme le graphe G est un graphe cordal, et que les graphes G' générés pour chaque sous-problème sont des sous-graphes induits du graphe G et sont cordaux eux-aussi, alors un sommet simplicial pourra toujours être trouvé au cours de l'exécution de l'algorithme. Les vecteurs de branchement correspondant aux cas 1,2 et 3 sont respectivement les vecteurs $(2, 1)$, $(2, 1)$ et $(2, 2)$. C'est le vecteur $(2, 1)$ qui nous donne la plus grande valeur, à savoir 1.6181. Il en résulte que le nombre d'ensembles dominants minimaux par inclusion dans un graphe est borné par $O(1.6181^n)$. □

4.4 Les graphes splits

La classe des graphes splits est une sous-classe de la classe des graphes cordaux. Tous les graphes splits sont donc aussi des graphes cordaux et les bornes que nous avons définies dans la section précédente sont valables pour les graphes de cette section. Mais nous allons montrer que, pour cette sous-classe particulière, nous sommes à même de donner de meilleures bornes que celles que nous avons déjà.

Un graphe $G = (V, E)$ est un graphe split si l'ensemble des sommets V peut être partitionné en deux ensembles distincts C et I. L'ensemble C forme alors une clique et l'ensemble I forme un ensemble stable, le couple

(C, I) sera appelé la *partition split* du graphe G. Une partition split peut être trouvée en temps linéaire [50], et n'est pas forcément unique. En effet, si l'on considère le cas d'un sommet x qui serait voisin de tous les sommets de la clique C, et dont le voisinage ne contiendrait aucuns des sommets de l'ensemble stable I, alors ce sommet x peut indifféremment être considéré comme un sommet de la clique et comme un sommet de l'ensemble stable, donnant ainsi deux partitions splits possibles pour le graphe correspondant.

Pour notre algorithme, nous considérerons les cas où l'ensemble stable I est maximal, c'est-à-dire que si nous avons le cas d'un sommet x qui peut indépendamment faire partie de l'ensemble stable ou de la clique, nous considérerons qu'il fait toujours partie de la clique. Ce choix a pour effet que, pour tout sommet v dans la clique C, nous pouvons supposer qu'il a au moins un voisin u dans l'ensemble stable I.

Notez qu'il n'est pas possible d'avoir en même temps dans un ensemble dominant minimal par inclusion un sommet u de l'ensemble stable I et un de ses voisins v de la clique c, car le voisin v de la clique C dominera forcément tous les sommets pouvant servir de voisin privé au sommet u de l'ensemble stable. On pourrait dire aussi que tous les sommets de l'ensemble stable dans le graphe split sont des sommets simplicials.

On remarquera enfin que la famille de graphes S_n que nous avons définie 4.2.1 est une famille de graphes splits, et qu'il y a donc au moins une famille de graphes dont le nombre d'ensembles dominants minimaux par inclusion est au moins $3^{\frac{n}{3}} \approx 1.4422^n$.

Théorème 11. *Un graphe split peut contenir au plus $O(1.4656^n)$ ensembles dominants minimaux par inclusion.*

Démonstration. Nous allons à nouveaux utiliser un algorithme de branchement afin de démontrer notre borne supérieure. Mais cette fois, l'algorithme que nous utiliserons sera constitué de deux phases distinctes, utilisant tout de même la technique de branchement toutes les deux.

L'entrée initiale de notre algorithme est un graphe split $G = (V, E) = (C, I, E)$ avec (C, I), une partition des sommets du graphe G telle que C est une clique et I est un ensemble stable. Il est nécessaire que l'ensemble stable I soit maximal, de façon à ce que tous les sommets de la clique C ont au moins un voisin dans l'ensemble stable I.

À chaque étape de l'algorithme, nous définirons des sous-problèmes de la forme (G', C', I', D) où G' est un sous-graphe induit de G et avec (C', I') une partition split de G' telle que $C' \subseteq C$ et $I' \subseteq I$. L'ensemble D représentant les sommets que l'on a déjà ajoutés dans l'ensemble dominant.

Au cours de l'exécution de notre algorithme, les sommets que nous placerons dans les I' seront toujours des sommets qui ne sont pas encore dominés,

ni par un sommet de la clique C, ni par eux même. Par conséquent, si un sommet de I' s'avérait être un sommet isolé, alors il faut l'ajouter à l'ensemble dominant D, cela se fait sans nécessiter de branchement.

Phase 1 : branchement sur la clique. Après avoir éliminé tous les sommets isolés de I', nous choisissons un sommet $c \in C'$ de façon à ce que c ait le plus de voisins possibles dans I'. Cette phase s'arrête si le nombre maximum de voisins possibles dans I' d'un sommet de la clique C' est inférieur à 2. Tant que l'on peut trouver dans C un sommet c tel que son nombre de voisins dans l'ensemble stable I' est supérieur ou égal à 2, alors nous branchons récursivement sur deux sous-problèmes.

- $c \in D$ Si l'on décide que le sommet c doit être ajouté à l'ensemble dominant D, alors il domine tous ses voisins dans I' et nous pouvons supprimer c avec tous ses voisins de $N(c) \cap I'$. Comme l'on sait que le sommet c avait au moins 2 voisins dans I', alors nous supprimons 3 sommets.
- $c \notin D$ Si l'on décide que le sommet c ne fait pas partie de l'ensemble dominant D, alors ses voisins de l'ensemble I' devront être dominés d'une autre manière. Tous les voisins de c perdant un degré, certains peuvent devenir des sommets isolés et doivent alors se dominer eux-même, les autres seront dominés dans les étapes suivantes de l'algorithme. Dans tous les cas, le sommet c lui-même n'a pas besoin d'être dominé, car faisant partie de la clique, il suffit que n'importe lequel des autres sommets de la clique soit ajouté à l'ensemble dominant pour que c soit dominé. Supposons maintenant que, lorsque l'on arrive à la fin de l'algorithme, aucuns des sommets de la clique n'ait été ajouté à l'ensemble dominant, cela signifie que tous les sommets de l'ensemble stable I' se dominent eux-même, et le sommet c est alors dominé par ses voisins dans I'. Nous pouvons donc simplement supprimer le sommet c du graphe G'.

Dans le premier cas, nous avons supprimé 3 sommets, et dans le deuxième cas, nous n'en supprimons que 1. Le vecteur de branchement correspondant est le vecteur $(3, 1)$ et la valeur correspondante est de 1.4656. Cela complète la description de la première phase de l'algorithme.

Phase 2 : branchement sur l'ensemble stable. Considérons les feuilles de l'arbre de recherche constitué par l'algorithme de la phase 1, et intéressons-nous aux instances du problème qui se trouve dans les-dites feuilles. Nous avons arrêté l'exécution de la phase 1 lorsque, pour chaque sommet de la clique C', il restait au plus 1 voisin dans l'ensemble stable I'. Mais à ce stade, l'ensemble $V(G')$ des sommets de G' n'est pas vide, et à cette feuille

peut encore correspondre bien plus d'un ensemble dominant. Nous pensons que le nombre d'ensembles dominants présents dans une des feuilles de l'arbre de recherche de la phase 1 ne peut excéder $3^{\frac{n}{3}}$, où n représente le nombre de sommets de G', et nous allons le démontrer par la description de la seconde phase de notre algorithme.

Tout d'abord, s'il existe dans la clique C' un sommet c qui n'ait plus aucun voisin dans l'ensemble dominant I', alors nous savons que ce sommet est déjà dominé. En effet, puisque tous les sommets de la clique du problème de départ avaient au moins 1 voisin dans l'ensemble stable, le fait que le sommet c n'ait pas de voisin dans I' signifie que tous les voisins qu'il y avait au départ sont déjà dominés et ont été éliminés. Or, pour dominer l'un des voisins de c, il faut soit prendre un autre sommet de la clique, qui dans ce cas, dominera c en même temps, soit prendre le voisin dans I' de c pour qu'il se domine lui-même, et qui, dans ce cas, dominera c en même temps. Dans tous les cas, si c n'a pas de voisin dans I', c'est qu'il est déjà dominé, et comme il n'a plus de voisins privés possibles, alors il ne sera jamais dans l'ensemble dominant et on le supprime. Cela se fait sans branchement.

De même, comme pour l'étape 1, nous pouvons supprimer, s'ils existent, tous les sommets isolés dans l'ensemble I', et les ajouter à l'ensemble dominant.

Une fois toutes les réductions effectuées, nous allons choisir un sommet $u \in I'$ et brancher sur $|N_{G'}[u]|$ sous-problèmes. Pour tous les sous-problèmes générés, nous choisissons un sommet dans $N_{G'}[u]$ afin de dominer u, et dans chacun de ces sous-problèmes, nous supprimons tous les sommets de $N_{G'}[u]$ car ils sont alors tous dominés et n'ont plus de possibilité de voisins privés.

Afin de donner la valeur de branchement de cette deuxième phase de notre algorithme, supposons que le sommet u, sur lequel nous allons brancher, possède $j-1$ voisins dans C'. Dans ce cas, il y aura j sous-problèmes générés qui, à chaque fois, verrons le nombre de sommets de G' décroître de j. Il est évident que $j \geq 1$. Une analyse simple nous donnera que les récurrences correspondant à cette règle de branchement sont de la forme :

$$T(n') = j \cdot T(n' - j)$$

Ce type de récurrence est bien connue, et le pire des cas correspond à lorsque la valeur $j = 3$, donnant la solution $T(n) \leq 3^{\frac{n'}{3}}$ (voir [38]). Le nombre de feuilles dans l'arbre de recherche de la phase 2 est donc de $3^{\frac{n'}{3}}$ et il ne reste cette fois qu'un seul ensemble dominant de G par feuille de l'arbre de recherche.

Phase 1 et 2 combinées. Pour établir la borne supérieure $\mu(G)$ du nombre d'ensembles dominants minimaux par inclusion dans le graphe G, notons que le nombre de feuilles de l'arbre de recherche de la première phase correspondant à une instance de n' sommets est au plus $O(1.4656^{n-n'})$, toutes ces feuilles correspondant à $n-n'$ sommets supprimés au graphe G d'origine jusqu'à atteindre le sommet G'.

Comme nous l'avons déjà vue dans le chapitre sur les ENSEMBLES STABLES BICOLORÉ [2.3], le temps d'exécution d'un algorithme décomposé en deux parties peut être considéré de la façon suivante :

$$\sum_{h=0}^{n} O(\alpha^{n-h} \cdot \beta^h) = O(\max(\alpha, \beta)^n)$$

Le nombre de feuilles de l'arbre de recherche de la première phase est donc borné par au plus $\sum_{n'=0}^{n} O(1.4656^{n-n'})$. Puisque nous savons que, pour chacune de ces feuilles, le nombre d'ensembles dominants est au plus $3^{\frac{n'}{3}} = 1.4423^{n'}$, nous pouvons en conclure que le nombre maximum d'ensembles dominants maximals par inclusion dans un graphe split G est au plus

$$\sum_{n'=0}^{n} O(1.4656^{n-n'} \cdot 1.4423^{n'}) \leq O(1.4656^n)$$

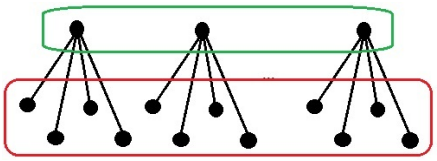
□

4.5 Les graphes co-biparties

Cette classe de graphe n'a pas de lien avec la classe des graphes cordaux, bien que l'on puisse considérer qu'elle peut avoir un lien avec la classe des graphes splits, étant donné que l'ensemble de ses sommets est partitionné en deux cliques.

Un graphe $G = (V, E) = (X, Y, E)$ est un graphe co-bipartie s'il est le graphe complémentaire d'un graphe bipartie. On peut aussi assurer que le graphe G est complémentaire d'un graphe bipartie si, et seulement si, l'ensemble de ses sommets peut être partitionné en deux cliques.

Afin d'obtenir une borne inférieure du nombre d'ensemble dominant minimal par inclusion que l'on peut trouver dans un graphe co-bipartie, nous allons définir la famille de graphes $\mathcal{B}_n$ de la façon qui suit. Pour $n = 5 \cdot k$, le nombre de sommets dans un graphe de la famille $\mathcal{B}_n$, nous formons deux cliques disjointes X est Y telles que $|X| = k$ et $|y| = 4k$. Puis nous plaçons



La clique X et la clique Y

des arêtes entre X et Y de façon à ce que chaque sommet dans X soit adjacent à exactement quatre sommets dans Y, et que chaque sommet dans Y ne soit adjacent qu'à un seul sommet dans X.

Pour cette famille de graphes, nous avons alors trois possibilités pour trouver un ensemble dominant minimal par inclusion.

$D \subseteq X$: Si l'on souhaite dominer le graphe avec uniquement des sommets de la clique X, alors nous sommes obligé de prendre tous les sommets de X dans l'ensemble dominant. Cela ne définit qu'une seule solution.

$D \subseteq X \cup Y$: Si nous souhaitons avoir dans notre ensemble dominant des sommets qui proviennent à la fois de la clique X et de la clique Y, alors nous n'avons en fait besoin que de deux sommets, à savoir un dans chaque clique qui dominera la clique dont il est issue. Le nombre de solutions correspondant est de $k \cdot 4 \cdot k$, soit $\frac{n}{5} \cdot \frac{4 \cdot n}{5}$. Nous pouvons considérer que c'est un nombre polynomial en $O(n^2)$.

$D \subseteq Y$: Si l'on décide de dominer le graphe en utilisant uniquement des sommets de la clique Y, alors pour chaque voisinage d'un sommet dans X, nous devons prendre un et un seul sommet. Le nombre de solutions correspondant est de $4^{\frac{n}{5}} \approx 1.3195^n$.

Globalement, nous pouvons dire que pour la classe des graphes co-biparties, il existe une famille de graphes avec au moins 1.3195^n ensembles dominants minimaux par inclusion, définissant ainsi une borne inférieure.

Théorème 12. *Un graphe $G = (V, E) = (X, Y, E)$ co-bipartie peut contenir au plus $O(1.5875^n)$ ensembles dominants minimaux par inclusion.*

Démonstration. Soit $G = (V, E) = (X, Y, E)$ un graphe co-bipartie de n sommets, avec X et Y les deux cliques qui forment la co-bipartition de G. Disons, sans perte de généralité, que la clique X est la plus grande des deux. Soit $|X| = \alpha \cdot n$ avec une constante $0.5 \leq \alpha \leq 1$, ce qui implique bien sûr que $|Y| = (1 - \alpha) \cdot n$.

Soit D un ensemble dominant minimal par inclusion dans le graphe G.

$|D| = 1$: Il est possible dans la classe des graphes co-biparties de trouver des sommets universels, c'est-à -dire des sommets voisins avec tous

les autres sommets du graphe. De tels sommets sont à eux-seuls des ensembles dominants minimaux par inclusion, définissant des ensembles D avec $|D| = 1$.

$D \cap X \neq \emptyset, D \cap Y \neq \emptyset$: Si l'on choisit de prendre un sommet dans chaque clique, alors, de même que lorsque nous avons défini la borne inférieure, il ne faut prendre qu'un seul sommet dans chaque clique afin de dominer la clique dont il est issu. Le pire des cas survient alors si les tailles des deux cliques sont équivalentes, et il y a au plus $\frac{n^2}{4}$ ensembles dominants minimaux par inclusion de ce type.

$D \subseteq Y$: Si l'on souhaite que l'ensemble dominant D soit un sous-ensemble de Y, la plus petite des deux cliques, alors, clairement, il y a moins de $2^{|Y|} \leq 2^{\frac{n}{2}}$ ensembles dominants minimaux par inclusion de ce type.

$D \subseteq X$: Reste enfin à énumérer les ensembles dominants minimaux par inclusion qui sont des sous-ensembles de X, la plus grande des deux cliques. Comme les ensembles dominants que nous cherchons doivent être minimaux par inclusion, cela signifie que chaque sommet dans D doit avoir un voisin privé dans le graphe G. Seuls les sommets dans Y peuvent servir de voisins privés, du moment que nous ne considérons plus les sommets universels et que $|D| \geq 2$. Mais cela implique directement que le nombre de sommets de $D \subseteq X$ doit être inférieur au nombre de sommets dans Y, afin d'avoir suffisamment de voisins privés. Soit $\beta \cdot n$ le nombre de sommets de l'ensemble D. Nous pouvons ainsi affirmer que $\beta \leq |Y|$, ou encore $\beta \leq (1 - \alpha)$. Nous pouvons alors énumérer le nombre d'ensembles de taille $\beta \cdot n$ contenus dans X. Le nombre de ces ensembles est de $\binom{\alpha \cdot n}{\beta \cdot n}$.
Pour tous α fixé, la valeur est maximisée si $\beta = \frac{\alpha}{2}$. Notez que $\beta = \frac{\alpha}{2}$ implique que $\alpha = \frac{2}{3}$. Donc, le nombre d'ensembles dominants minimaux par inclusion qui sont des sous-ensembles de la clique X est au plus $\binom{\frac{2 \cdot n}{3}}{\frac{n}{3}}$, ce qui est inférieur ou égal à $2^{\frac{2 \cdot n}{3}}$.

Finalement, le nombre d'ensembles dominants minimaux par inclusion dans un graphe co-bipartie peut être borné par $n + \frac{n^2}{4} + 2^{\frac{n}{2}} + 2^{\frac{2 \cdot n}{3}}$. Cela est équivalent à $O(2^{\frac{2 \cdot n}{3}})$ soit $O(1.5875^n)$. Il y a donc au plus $O(1.5875^n)$ ensembles dominants minimaux par inclusion dans un graphe co-bipartie. □

4.6 Les co-graphes

Les co-graphes sont une classe de graphe pour laquelle le nombre maximum d'ensembles dominants minimaux par inclusion est particulièrement intéressant à étudier, car la borne inférieure pour le cas général nous est donnée par un co-graphe. En effet, le seul graphe connu qui nous permette d'obtenir

une borne inférieure de 1.5704^n est un co-graphe. Nous allons voir que ce nombre définit aussi, pour cette classe de graphe, une borne supérieure.

Rappelons tout d'abord que nous avons définit en introduction [1.3.2], qu'un graphe $G = (V, E)$ est un co-graphe s'il peut être construit à partir de sommets isolés en utilisant uniquement des unions disjointes et des jonctions. Une autre façon de définir les co-graphes est de préciser que ce sont des graphes sans P_4, c'est-à-dire que l'on ne peut pas trouver de chemin de 4 sommets comme sous-graphe induit.

Le graphe que nous avons donné en exemple pour la borne inférieure dans le cas général est un co-graphe de six sommets dans lequel chaque paire de sommets est un ensemble dominant minimal par inclusion. Il y a donc 15 ensembles dominants minimaux par inclusion dans ce graphe de 6 sommets. Une union disjointe permet de définir une famille de co-graphes (non connexes) dont le nombre de sommets est $n = 6k$, avec 15 ensembles dominants minimaux par inclusion par composante connexe. Nous avons donc une famille de co-graphes avec $15^{\frac{n}{6}} \approx 1.5704^n$ ensembles dominants minimaux par inclusion.

Théorème 13. *Le nombre d'ensembles dominants minimaux par inclusion dans un co-graphe est au plus* $15^{\frac{n}{6}}$*. Il existe une famille de co-graphes dont le nombre d'ensembles dominants minimaux par inclusion est de l'ordre de* $15^{\frac{n}{6}}$*. Cette borne est donc la meilleure borne possible.*

Démonstration. Contrairement aux autres parties de ce chapitre, nous n'allons pas ici utiliser d'algorithmes pour démontrer notre théorème. Nous allons préférer utiliser une démonstration par induction.

Avant toutes choses, remarquons que nous avons défini, pour la borne inférieure, une famille de co-graphes dont le nombre d'ensembles dominants minimaux par inclusion est de $15^{\frac{n}{6}}$. Il est possible de vérifier, de façon exhaustive, que cette valeur est bien la plus grande valeur possible à atteindre pour tout co-graphe dont la taille n'excède pas six sommets, en énumérant manuellement, ou avec l'aide d'un programme informatique, tous les graphes de six sommets possibles.

Supposons maintenant que nous soyons dans le cas d'un co-graphe $G = (V, E)$ dont le nombre de sommets $|V| \geq 7$. Nous allons démontrer notre théorème par induction sur le nombre de sommets. Par définition de ce qu'est un co-graphe, il existe deux sous-graphes distincts G_1 et G_2 dans G tels que, soit nous avons $G = G_1 \uplus G_2$, soit nous avons $G = G_1 \bowtie G_2$. J'appelle $n_i = |V(G_i)|$ pour $i = \{1, 2\}$ le nombre de sommets des graphes G_i.

$G = G_1 \uplus G_2$: et dans ce cas, le graphe G est un graphe non connexe dont les deux composantes sont les deux graphes G_1 et G_2. (Remarquez que G_1 et G_2 eux-même ne sont peut être pas connexes.) Quoi qu'il en soit, pour avoir un ensemble dominant minimal par inclusion dans G,

il faut indépendamment un ensemble dominant minimal par inclusion pour dominer G_1 et un autre pour dominer G_2. Nous avons donc que le nombre d'ensembles dominants minimaux par inclusion est :

$$\mu(G) = \mu(G_1) \cdot \mu(G_2) \leq 15^{\frac{n_1}{6}} \cdot 15^{\frac{n_2}{6}} \leq 15^{\frac{n}{6}}$$

$G = G_1 \bowtie G_2$: et dans ce cas, le graphe G peut être partitionné en deux sous-graphes G_1 et G_2 tels que toutes les arêtes possibles existent entre G_1 et G_2. Pour avoir un ensemble dominant minimal par inclusion dans le graphe G, il y a alors trois possibilités :

$D \subseteq V(G_1)$: et comme toutes les arêtes existent entre G_1 et G_2, alors avoir pris au moins un sommet de G_1 suffit à dominer l'ensemble du graphe G_2.

$D \subseteq V(G_2)$: et inversement, comme toutes les arêtes sont présentes entre G_1 et G_2, avoir pris un seul sommet de G_2 suffit à dominer G_1.

$D \cap V(G_1) \neq \emptyset, D \cap V(G_2) \neq \emptyset$: et dans ce cas, tous les ensembles dominants minimaux par inclusion seront constitués d'un sommet de G_1 associé à un sommet de G_2. Il y a au plus $n_1 \cdot n_2 \leq n^2$ possibilités.

Nous avons donc que le nombre d'ensembles dominants minimaux par inclusion est :

$$\mu(G) \leq \mu(G_1) + \mu(G_2) + n_1 \cdot n_2 \leq 15^{\frac{n_1}{6}} + 15^{\frac{n_2}{6}} + n_1 \cdot n_2$$

Sachant que nous pouvons supposer que $n = n_1 + n_2$, que G_1 et G_2 ne sont pas vides, c'est-à-dire $1 \leq n_1 \leq n-1$ et $1 \leq n_2 \leq n-1$, et que $n \geq 7$, alors la fonction $15^{\frac{n_1}{6}} + 15^{\frac{n_2}{6}} + n_1 \cdot n_2$ est maximale pour $n_1 \in \{1, n-1\}$ (respectivement $n_2 \in \{n-1, 1\}$). Dans tous les cas, le résultat obtenu permet d'affirmer que $\mu(G) \leq 15^{\frac{n-1}{6}} + 15^{\frac{1}{6}} + n - 1 \leq 15^{\frac{n}{6}}$, et cela pour tout $n \geq 7$.

□

4.7 Les graphes d'intervalles propres

Un graphe est dit d'*intervalles propres*, ou encore d'*intervalles unitaires* si, et seulement si, il peut être représenté par un ensemble d'intervalles sur une droite réelle, tel que chaque sommet correspond à un intervalle, une intersection entre deux intervalles signifie l'existence d'une arête entre les deux sommets correspondant, et aucun intervalle ne peut en contenir un autre [88]. Un *ordre d'intervalles propres* $\{v_1, v_2, \cdots, v_n\}$ sur un graphe G est un ordre donné aux sommets qui satisfait la propriété suivante : si $v_i v_j \in E$, avec $i < j$, alors les sommets $v_i, v_i + 1, \cdots, v_j$ constituent une clique dans le

graphe G. Un graphe G est un graphe d'intervalles propres si, et seulement si, il dispose d'un ordre d'intervalles propres [70].

On remarquera que le graphe H_n défini dans la partie 4.2.1 est un graphe d'intervalles propres. La borne inférieure pour le nombre d'ensembles dominants minimaux par inclusion dans un graphe d'intervalles propres est donc $3^{\frac{n}{3}}$.

Théorème 14. *Un graphe d'intervalles propres dispose au plus de* $O(1.4656^n)$ *ensembles dominants minimaux par inclusion.*

Démonstration. Pour notre démonstration, nous allons utiliser un algorithme de branchement que nous analyserons avec la technique mesurer et conquérir [38] en donnant des poids aux sommets. Soit (G', D) une instance d'un sous-problème. Nous allons définir trois types de sommet :

- Un sommet dont l'un des voisins au moins à été ajouté à l'ensemble dominant est un sommet *dominé*.
- Un sommet que l'on a déterminé comme ne pouvant pas faire partie de l'ensemble dominant est un sommet *interdit*.
- Un sommet qui n'est ni dominé ni interdit est un sommet *libre*.

Nous décidons, pour la mesure, d'attribuer aux sommets dominés et aux sommets libres le poids 1, et aux sommets interdits le poids 0. La mesure d'une instance (G', D) est la somme de tous les poids des sommets de G'. Initialement, tous les sommets dans un graphe G sont libres, et ont donc le poids 1. La mesure de l'instance d'entrée $(G, \emptyset)$ est donc n. Remarquez que si un sommet est à la fois dominé et interdit, alors on peut le retirer du graphe sans risque, ce sommet étant dominé et ne pouvant en aucun cas être ajouté à l'ensemble dominant. Cela nous définit une règle de réduction, et les sommets à la fois dominés et interdits ne seront donc jamais pris en considération dans nos récurrences.

Soit $\sigma = \{v_1, v_2, \cdots, v_n\}$ un ordre d'intervalles propres sur le graphe d'entrée G. Dans chaque instance (G', D), nous choisissons le sommet x dont l'index est le plus petit dans σ afin d'effectuer un branchement. Notez que σ définit un ordre d'intervalles propres sur tous les sous-graphes de G, et en particulier sur le graphe G'. Le sommet x de plus petit index sur σ est donc un sommet simplicial.

Durant le déroulement de notre algorithme, les deux affirmations suivantes seront toujours vraies.

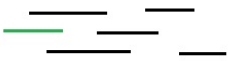
1 : Si un sommet v_j dans G' est un sommet interdit, alors tout sommet v_i dans G' avec $i < j$ est lui aussi un sommet interdit.

2 : Si un sommet v_j dans G' est un sommet dominé, alors tout sommet v_i dans G' avec $i < j$ est lui aussi un sommet dominé.

Lorsque l'on créera de nouveaux sous-problèmes, nous montrerons que si l'on suppose que ces deux affirmations sont vraies pour l'instance en cours, alors elles le restent sur les instances des sous-problèmes créés. Dans l'instance initiale, tous les sommets sont libres et les deux affirmations sont donc trivialement vraies.

Nous allons maintenant décrire les différentes phases de l'algorithme. Soit x le sommet de G' avec le plus petit index dans σ. Comme nous l'avons déjà précisé, x est simplicial. Nous distinguerons deux cas principaux, selon que le sommet x est un sommet interdit ou pas. Pour chaque cas, on peut distinguer plusieurs sous-cas selon le nombre de voisins de x dans le graphe G'. Pour chaque instance, un seul cas peut être appliqué.

Cas 1a : x n'est pas interdit, est a au moins deux voisins dans le graphe G'. Notez que si x n'est pas interdit, alors de par l'affirmation 1, et x étant le sommet de plus petit index dans σ, aucuns des voisins de x n'est un sommet interdit.



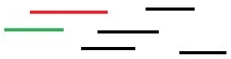
Le sommet x dans la représentation en intervalles du graphe

Nous branchons selon la possibilité d'ajouter x à l'ensemble dominant D, ou d'éliminer x des sommets pouvant être ajoutés à D.

- $x \in D$: le sommet x domine $N_{G'}(x)$ et a besoin d'un voisin privé dans $N_{G'}(x)$. $N_{G'}(x)$ étant une clique, cela signifie qu'aucuns des sommets de $N_{G'}(x)$ ne peut apparaître dans un ensemble dominant minimal par inclusion qui contiendrait D. Par conséquence, on peut supprimer $N_{G'}(x)$ sans risques, créant ainsi l'instance $(G' \backslash N_{G'}(x), D \cup \{x\})$. Puisque $N_{G'}(x)$ contient au moins deux sommets de poids 1, le sommet x lui-même étant de poids 1, alors, lors de ce branchement, la mesure va décroître de au moins 3.
- $x \notin D$: Dans ce cas, nous allons simplement interdire le sommet x. S'il était déjà dominé, alors on peut le supprimer. Si non, alors l'instance reste la même, sauf que le poids de x est passé de 1 à 0. Dans tous les cas, la mesure décroît de 1.

Il est clair que pour les deux branchements, les affirmations 1 et 2 sont préservées.

Cas 1b : x est un sommet libre (pas interdit et pas dominé) est a un seul voisin dans le graphe G'. Soit y le voisin de x dans le graphe G'. Grâce aux affirmations 1 et 2, nous savons que si x est un sommet libre, alors y l'est aussi.



Les sommets x et y dans la représentation en intervalles du graphe

Puisque x n'est pas dominé et que son seul voisin est y, alors l'un des deux sommets est requis dans l'ensemble dominant pour pouvoir dominer le sommet x. Nous branchons selon ces deux possibilités.

- $x \in D$: Dans ce cas, y devient dominé, et aucuns ensembles dominants minimaux par inclusion ne peuvent plus contenir y sans retirer à x ses voisins privés. Nous pouvons donc supprimer sans risques les deux sommets x et y, obtenant l'instance $(G' \backslash \{x, y\}, D \cup \{x\})$ et faisant ainsi décroître la mesure de 2.
- $y \in D$: Dans ce cas, x devient dominé, et plus aucun sommet dans G' ne peut plus être voisin privé pour x. Nous pouvons donc supprimer les sommets x et y sans risques. L'instance créée cette fois est $(G' \backslash \{x, y\}, D \cup \{y\})$, et la mesure décroît de 2.

Dans les deux cas, il est clair que les deux affirmations 1 et 2 sont préservées.

Cas 1c : x n'est pas interdit mais est dominé, et a exactement 1 voisin dans G'. Soit y le voisin de x dans G'.

- $x \in D$ Si l'on choisit de prendre le sommet x dans l'ensemble dominant D, cela signifie que le sommet y est son voisin privé. Par conséquent, aucun autre voisin de y ne peut plus faire partie de l'ensemble dominant D. On supprime alors le sommet x, le sommet y et au moins un voisin de y devient interdit.

Si y est voisin privé de x, les autres voisins de y deviennent interdits

La mesure décroît alors de 3. Le cas où y n'aurait pas d'autres voisins est négligeable, car cela signifie que l'exécution de l'algorithme est alors terminée sur cette composante connexe du graphe.

$x \notin D$ Si l'on décide que le sommet x ne fera pas partie de l'ensemble dominant D, alors il devient un sommet à la fois dominé et interdit. On peut donc sans risques le supprimer du graphe G'. La mesure décroît alors de 1.

Une fois encore, dans les deux cas, les affirmations 1 et 2 sont conservées.

Cas 1d : x n'est pas interdit et est un sommet isolé dans le graphe G'. Ce cas correspond à une règle de réduction. Si x est dominé, alors on peut simplement le supprimer, si il n'est pas dominé, il faut l'ajouter à l'ensemble dominant D. Dans tous les cas, l'exécution de notre algorithme est terminée sur cette composante connexe du graphe.

Cas 2a : x est interdit, et il y a au moins deux voisins de x qui ne sont pas interdits dans le graphe G'. Puisque x est interdit, mais n'est pas dominé, il est nécessaire que au moins un de ses voisins fasse partie de l'ensemble dominant D. Rappelons nous que le sommet x est un sommet simplicial, et que son voisinage $N(x)$ forme une clique. Soit $\{y_1, y_2, \cdots, y_t\}$ les voisins de x qui ne sont pas interdits, ordonnés selon leur ordre dans σ. Puisque nous avons que σ est un ordre d'intervalles propres, nous pouvons conclure que $N[x] \subseteq N[y_1] \subseteq N[y_2] \subseteq \cdots \subseteq N[y_t]$. Cela nous permet notamment de conclure que deux sommets différents dans $N(x)$ ne peuvent faire partie de l'ensemble dominant D en même temps. Le branchement à effectuer consiste alors à choisir quel sommet y_i, pour $1 \leq i \leq t$, doit être ajouté à l'ensemble dominant D. En ajoutant y_i à l'ensemble dominant D, alors $N[x]$ sera dominé et, de part l'argument précédent, nous pouvons sans risques supprimer $N[x]$ du graphe G'. De plus, le voisinage $N(y_i)$ peut être marqué comme dominé. On remarque que les affirmations 1 et 2 sont alors préservées.

Ce branchement crée alors $t \geq 2$ sous-instances différentes, la mesure décroissant de t dans chacune de ces sous-instances. Le vecteur de branchement comporte alors t valeurs, et est de la forme $(t, t, \cdots, t)$. Le pire des branchements dans ce cas est si $t = 3$. (En particulier, les branchements $(2, 2)$ et $(4, 4, 4, 4)$ ont pour nombre de branchement 1.4143, et $(5, 5, 5, 5, 5)$ a pour nombre de branchement 1.3797) Le vecteur de branchement dans le pire des cas sera donc $(3, 3, 3)$, ce qui correspond à un nombre de branchement égal à 1.4423.

Cas 2b : x est interdit, et n'a qu'un seul voisin non interdit dans le graphe G'. Soit y l'unique voisin non-interdit de x dans le graphe G'. Dans ce cas, la seule possibilité pour que le sommet x soit dominé est de prendre le sommet y dans l'ensemble dominant D. Il s'agit là d'une règle de réduction, ne

nécessitant donc aucuns branchements. Les voisins de y dans le graphe G' deviennent alors des sommets dominés, et on peut, sans risques, supprimer $N[x]$. Remarquez que les affirmations 1 et 2 restent conservées.

Cas 2c : x est interdit et n'a aucun voisin non-interdit dans le graphe G. Dans ce cas, il est impossible d'étendre l'ensemble dominant D, nous sommes dans une feuille négative de l'arbre de recherche de notre algorithme. Nous pouvons stopper cette récurrence et éliminer l'ensemble dominant D.

Finalement, les différents vecteurs de branchement que l'on obtient pour les récurrences de notre algorithme sont les vecteurs $(2, 2)$, $(3, 1)$ et $(3, 3, 3)$. C'est le vecteur de branchement $(3, 1)$ qui nous donne le plus grand nombre de branchement, 1.4656, nous donnant ainsi une borne supérieure pour le temps d'exécution de notre algorithme en $O(1.4656^n)$.

□

4.8 Les graphes trivialement parfaits

Les graphes trivialement parfaits (en anglais, *trivially perfect graphs*) sont une sous-classe des co-graphes, et peuvent être caractérisés de plusieurs façons différentes [15, 50]. Un graphe G est trivialement parfait si, et seulement si, chaque sous-graphe induit de G contient un sommet universel [99].

Théorème 15. *Un graphe trivialement parfait dispose d'au plus* $3^{\frac{n}{3}}$ *ensembles dominants minimaux par inclusion, et il existe une famille de graphes trivialement parfaits dont le nombre d'ensembles dominants minimaux par inclusion est de* $3^{\frac{n}{3}}$.

Démonstration. Observez que le graphe H_n défini dans la partie 4.2.1 est un graphe trivialement parfait. Nous avons donc un exemple de famille de graphes trivialement parfaits dont le nombre d'ensembles dominants minimaux par inclusion est de $3^{\frac{n}{3}}$. Nous allons maintenant faire une démonstration par induction sur le nombre de sommets dans un graphe trivialement parfait en utilisant le graphe H_6 comme cas de base. Pour tous les graphes trivialement parfaits de moins de 6 sommets, il est facile de vérifier que le théorème est correct.

Supposons que le théorème soit correct pour tous les graphes trivialement parfaits jusqu'à $n - 1$ sommets, et soit G un graphe trivialement parfait de n sommets.

- Si G n'est pas connexe, avec $n_1, n_2, \cdots, n_t$ la taille des t composantes connexes du graphe G et tels que $n_1 + n_2 + \cdots + n_t = n$ et $n_i \leq n$

pour $1 \leq i \leq t$, alors dans ce cas, toutes les composantes connexes de G sont des graphes trivialement parfaits dont le nombre de sommets est inférieur ou égal à $n-1$. Par hypothèse de récurrence, nous avons donc que $\mu(G) \leq 3^{\frac{n_1}{3}} \cdot 3^{\frac{n_2}{3}} \cdot \cdots \cdot 3^{\frac{n_t}{3}}$.

– Supposons que G est connexe. Dans ce cas, il existe un sommet x dans le graphe G tel que x est un sommet universel. Le seul et unique ensemble dominant minimal par inclusion qui contient le sommet x est l'ensemble $\{x\}$ car, à lui seul, il domine la totalité du graphe G. De ce fait, le nombre d'ensembles dominants minimaux par inclusion dans le graphe G qui ne contiennent pas x est exactement le nombre d'ensembles dominants minimaux par inclusion dans le graphe $G\backslash\{x\}$, puisque le sommet x peut être dominé par absolument n'importe quel sommet dans le graphe. Par conséquent, le nombre d'ensembles dominants minimaux par inclusion dans le graphe G est :

$$\mu(G) = \mu(G\backslash\{x\}) + 1 \leq 3^{\frac{n-1}{3}} + 1$$

Puisque $3^{\frac{n-1}{3}} + 1 \leq 3^{\frac{n}{3}}$ pour tous les $n \geq 3$, nous pouvons conclure que $\mu(G) \leq 3^{\frac{n}{3}}$.

Il en découle que, pour un graphe trivialement parfait, le nombre d'ensembles dominants minimaux par inclusion est toujours inférieur à $3^{\frac{n}{3}}$. □

4.9 Conclusion

Nous avons présenté des bornes supérieures et des bornes inférieures pour le problème d'énumération d'Ensembles Dominants Minimaux par inclusion sur les graphes cordaux, sur les graphes co-biparties, sur les graphes splits, sur les graphes d'intervalles propres, sur les co-graphes et sur les graphes trivialement parfaits. Pour les co-graphes et les graphes trivialement parfaits, les bornes inférieures et supérieures que nous présentons sont ajustées.

On peut se demander si les algorithmes d'énumération que nous avons développés peuvent servir à établir des algorithmes exact et exponentiels plus rapides pour les problèmes de nombre Domatique ou pour Ensemble Dominant Connexe sur les graphes splits ou sur les graphes cordaux. Pour tous les problèmes pour lesquels les bornes que nous fournissons ne sont pas ajustées, on peut se demander s'il existe une borne inférieure plus importante, ou une meilleure borne supérieure. Peut-on fournir une borne supérieure pour les graphes biparties meilleure que 1.7159^n ? Finalement, nous faisons une conjecture sur le nombre maximum d'ensembles dominants

minimaux par inclusion dans un graphe trivialement parfait, ou peut-être même pour un graphe cordal, qui serait au plus de $3^{\frac{n}{3}}$.

Nous signalerons que le fait de trouver s'il existe un algorithme polynomial en la taille du résultat pour le problème d'énumération des TRANSVERSAUX MINIMAUX dans un hypergraphe est un grand problème ouvert actuellement, et que le problème d'ENSEMBLES DOMINANTS MINIMAUX par inclusion est un cas particulier de ce problème [57].

Chapitre 5

Ensembles coupe cycles

Dans ce chapitre, nous allons quelques peu quitter le domaine de l'algorithmique pour nous approcher de la théorie des graphes. Ainsi, il n'y aura pas, à proprement parler, d'algorithmes dans ce chapitre, mais plutôt des démonstrations par induction, ou par récurrences, sur la structure des graphes étudiés. Cela sort un peu de la thématique générale de cette thèse, mais reste un sujet somme toute très intéressant. De plus, les résultats obtenus de façon combinatoire sur le nombre d'ensembles coupe cycles minimaux par inclusion sur un type de graphe donné implique indirectement un algorithme dont le temps d'exécution est de l'ordre de la borne combinatoire donnée grâce à l'algorithme d'énumération des ensembles coupe cycles en délai polynomial de Schwikowski et Speckenmeyer [90].

5.1 Introduction

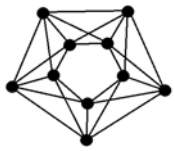
Un *ensemble coupe cycle* $F \subseteq V$, (en anglais, *feedback vertex set*), est un sous-ensemble de sommets d'un graphe $G = (V, E)$ tel que si l'on retire tous les sommets contenus dans F à l'ensemble V, on obtient un sous-graphe induit $G(V \backslash F)$ qui est sans cycles.

Le problème qui consiste à trouver un ensemble coupe cycle de taille ou de poids minimum est l'un des problèmes NP-difficiles les plus étudiés de ces dernières années en algorithmique des graphes. De nombreux documents ont d'ailleurs été publiés sur le sujet.

L'un de ces résultats les plus importants est la démonstration en 2002 par Schwikowski et Speckenmeyer que tous les graphes coupe cycles minimaux par inclusion d'un graphe donné peuvent être énumérés avec un délai polynomial [90]. Cela signifie que pour tout graphe, le temps nécessaire à énumérer tous les ensembles coupe cycles minimaux par inclusion pourra être calculer

en fonction du nombre de ces ensembles, ou autrement dit, sera polynomial au rapport du résultat.

En 2008, Fomin et al. ont montré que le nombre maximum d'ensembles coupe cycles dans un graphe de n sommets était au plus de 1.864^n [39]. Grâce à l'algorithme d'énumération en délai polynomial que nous avons évoqué juste avant, cela nous donne immédiatement un algorithme en $O^*(1.864^n)$ pour donner la liste de tous les ensembles coupe cycles du graphe, mais aussi pour les compter, ou encore donner celui de poids ou de cardinalité minimum. Dans ce document, Fomin et al. nous donne aussi un graphe à dix sommets dont le nombre d'ensembles coupe cycles est de 105. Ce graphe, et tout graphe composé d'unions disjointes de copies de ce graphe, forme une famille dont le nombre d'ensembles coupe cycles minimaux par inclusion est actuellement le plus élevé, nous donnant ainsi une borne inférieure de 1.593^n.



Un graphe de 10 sommets avec 105 ensembles coupe cycles

Les résultats obtenus par Fomin et al., et l'écart important entre les bornes inférieures et supérieures que nous pouvons constater, nous ont poussé à tourner notre attention vers les classes de graphes particulières, avec pour objectif, si possible, de trouver des bornes inférieures et supérieures les plus proches possibles. De ce fait, nous allons montrer dans ce chapitre que, pour les graphes cordaux et les co-graphes, le nombre d'ensembles coupe cycles minimaux par inclusion et d'au plus $10^{\frac{n}{5}} \approx 1.585^n$, et que cette valeur est aussi la borne inférieure. Ce résultat est obtenu avec des outils purement combinatoires, et des demonstrations par induction. Bien que la plupart des résultats obtenus dans [39] soient fait grâce à des outils algorithmiques, ce ne sera pas ici le cas. Cependant, nous avons toujours la possibilité d'utiliser l'algorithme de Schwikowski et Speckenmeyer [90] grâce auquel nos résultats peuvent être traduit par le fait que leur algorithme s'exécute en $O^*(1.585^n)$ sur les graphes cordaux ou les co-graphes.

Nos résultats impliquent aussi que, si l'on souhaite trouver une meilleure borne inférieure dans le cas général, alors les familles de graphes appartenant aux graphes cordaux et aux co-graphes doivent être écartés.

Les classes de graphes ont suscité beaucoup d'intérêt pour l'étude de problèmes d'optimisation, mais l'intérêt commence à se porter aussi sur les

avantages qu'elles nous apportent lorsque l'on souhaite faire du comptage, ou de l'énumération [24, 57, 80, 81].

Les graphes cordaux et les co-graphes sont, sans doutes, les sous-classes de graphes des graphes parfaits qui sont les plus connus et les plus étudiés, avec plusieurs applications possibles dans des problèmes réels [15, 50, 91].

Beaucoup de problèmes qui sont NP-complets dans le cas général peuvent être résolus en temps polynomial sur ces classes de graphe, et c'est notamment le cas du problème consistant à trouver un ensemble coupe cycle de poids minimum [20, 91].

De ce fait, notre motivation pour ces résultats n'a pas de lien avec la recherche d'un algorithme efficace permettant de trouver un ensemble coupe cycle de poids ou de taille minimum sur ces classes de graphe. Mais le fait de connaître les bornes combinatoires pourrait avoir des implications sur d'autres problèmes d'optimisation, comme quand le fait que Moon et Moser [76] aient borné le nombre maximum d'ensembles stables maximals par inclusion a permis à Lawler [68] d'en obtenir un algorithme pour les colorations de graphes, qui pendant plusieurs décennies fut le meilleur connu. L'algorithme obtenu par Eppstein, s'exécutant encore plus rapidement [33], utilise d'ailleurs une autre borne combinatoire, meilleure, pour les ensembles stables de petites tailles.

5.2 Préliminaires

Nous allons faire quelques petits rappels utiles à ce chapitre.

5.2.1 Les classes de graphe

Une corde, dans un cycle ou un chemin, est une arête qui relie deux sommets non consécutifs du cycle ou du chemin. Un graphe est cordal si, pour tous cycles de taille au moins quatre, il y a une corde. Une décomposition arborescente en arbre de cliques est un arbre T, tel que chaque nœud de l'arbre représente une clique dans le graphe G. Pour chaque sommet v du graphe G, l'ensemble des sommets de T correspondant à une clique maximale dans laquelle le sommet v apparait forme un sous-arbre connexe. Un graphe a une décomposition arborescente en arbre de cliques si, et seulement si, le graphe est cordal, et l'on peut décider si un graphe est cordal, ainsi que donner une décomposition arborescente en arbre de cliques, en temps polynomial [12].

L'union disjointe de graphes est une opération prenant en entrée une collection de graphes, et qui renvoie un graphe dont les différentes composantes

connexes sont les graphes de l'entrée, aucunes arêtes n'ayant été ajoutées. La jonction de graphes est une opération prenant en entrée une collection de graphes, et qui renvoie un graphe composé des graphes de l'entrée entre lesquels on aurait ajouté toutes les arêtes possibles. Un co-graphe est un graphe qui peut être obtenu à partir de sommets isolés en appliquant uniquement des opérations d'unions disjointes et de jonctions successives. Les co-graphes peuvent aussi être définis comme la classe des graphes n'ayant aucuns chemins de taille 4 comme sous-graphe induit. Ils peuvent être reconnus en temps linéaire [21]. Vous pouvez retrouver cette définition dans le chapitre **??**.

Un graphe d'intervalles circulaires (*circular arc* en anglais) et un graphe d'intersections entre des arcs de cercles. Un tel graphe a, au plus, n cliques, et dispose d'une représentation en cycle de cliques, analogue à la décomposition arborescente en arbre de cliques des graphe cordaux. Cette classe de graphe ne fait pas partie des sous-classes de graphes parfaits, et ne peut pas être comparée non plus aux graphes cordaux ni aux co-graphes. Elle peut être reconnue en temps linéaire [73].

Pour toutes ces classes de graphe, des informations plus complètes peuvent être trouvées dans les livres de Brandstädt et al. [15] ou de Golumbic [50].

5.2.2 Le problème des ensembles coupe cycles

Un ensemble $S \subseteq V$ est un ensemble coupe cycle (en anglais, *Feedback vertex set*) si $G[V \backslash S]$ est une forêt, c'est-à-dire un ensemble d'arbres. Un ensemble coupe cycle S est minimal si aucun sous-ensemble de S n'est un ensemble coupe cycles. Dans ce cas, $G[V \backslash S]$ est une forêt induite maximale.

Il y a une bijection forte entre les ensembles coupe cycles minimaux par inclusion et les forêts induites maximales, et leurs nombres sont égaux.

Pour un graphe non connexe, le nombre d'ensembles coupe cycles est le produit du nombre d'ensembles coupe cycles de chaque composante connexe. Nous utiliserons cette observation à de nombreuses reprises.

En particulier, intéressons-nous au graphe K_5, un graphe complet de cinq sommets. Clairement, ce graphe est cordal, car le cycle le plus grand est un triangle, et c'est aussi un co-graphe, car on peut l'obtenir à partir de cinq sommets isolés en effectuant uniquement des jonctions.

Dans le K_5, chaque arête est une forêt induite maximale, et il n'existe aucune autre forêt induite. Nous avons donc un graphe de cinq sommets avec dix forêts induites maximales (ou dix ensembles coupe cycles).

Considérons maintenant la famille de graphes définie par tous les graphes possibles à obtenir composés de K_5 disjoints. Clairement, tous les graphes de

cette famille sont cordaux, puisqu'aucun nouveau cycle n'est créé, et ils sont aussi des co-graphes, puisque créés par l'union disjointe de co-graphes.

Nous avons donc une famille infinie de graphes dont le nombre de sommets est $5 \cdot k$, et le nombre d'ensembles coupe cycles est de $10 \cdot k$. L'entier k pouvant prendre n'importe quelle valeur. Pour cette famille, le nombre d'ensembles coupe cycles est donc de $10^k = 10^{\frac{n}{5}} \approx 1.585^n$. Cela nous donne une famille infinie de graphes qui constitue pour nous la meilleure borne inférieure possible pour les graphes cordaux et les co-graphes. Nous allons maintenant démontrer que cette borne est aussi égale à la borne supérieure pour ces deux classes de graphe.

5.3 Une borne ajustée pour les graphes cordaux

Dans cette partie, nous allons démontrer que la borne supérieure pour le nombre d'ensembles coupe cycles dans un graphe cordal est la même que la borne inférieure que nous avons donnée dans la partie précédente. Pour les arguments que nous allons utiliser, il est plus simple de parler des forêts induites maximales, plutôt que des ensembles coupe cycles, mais cela ne change rien au résultat.

La question que nous allons chercher à résoudre est, en fait, plus forte que le problème de base. Pour un graphe $G = (V, E)$ et un ensemble $F \subseteq V$, quel est le nombre maximum de forêts induites maximales par inclusion que l'on peut trouver dans G, et qui contiennent tous les sommets de F ? Clairement, si $G[F]$ n'est pas une forêt, alors la réponse est 0, et si l'ensemble F est vide, alors la réponse est exactement le nombre de forêts maximales par inclusion contenues dans le graphe G.

Théorème 16. *Soit $G = (V, E)$ un graphe cordal, et soit $F \subseteq V$ un sous-ensemble des sommets de V, tel que $G[F]$ est une forêt. Alors le graphe G contient au plus $10^{\frac{n-|F|}{5}}$ forêts induites maximales par inclusion qui contiennent tous les sommets inclus dans F.*

Démonstration. Pour commencer, supposons que le graphe G est connexe. Dans le cas d'un graphe non connexe, alors le fait de pouvoir démontrer la borne supérieure sur chaque composante connexe nous donnera immédiatement pour le graphe au complet une borne équivalente.

Soit T un arbre représentant la décomposition arborescente en arbre de cliques du graphe G, et soit k le nombre de sacs (ou de nœuds) sur cet arbre. Nous allons faire une démonstration par induction sur la valeur de k. Rappelons que $n = |V|$, nous définissons que $n' = n - |F|$ et $V' = V \backslash F$?

$k = 1$ **:** Pour le cas de base, notre graphe G est un graphe complet de $n \geq 1$ sommets. Puisque toutes les arêtes possibles existent dans G, alors la totalité des forêts induites de taille maximum dans G sont en fait des arêtes, et ne contiennent que 2 sommets si $n \geq 2$.

Si $n = 1$, alors il n'y a qu'une unique forêt de taille maximum, et elle ne contient qu'un seul sommet. Comme on a que $1 \leq 10^0 \leq 10^{1-|F|}$, alors la proposition fonctionne dans ce cas.

Si jamais $n \geq 2$, alors, puisque les forêts de taille maximum ne peuvent avoir que 2 sommets, l'ensemble F ne peut contenir que, au plus, 2 sommets (il peut aussi n'en contenir qu'un ou être vide). Soit $i = 2 - |F|$ le nombre de sommets que l'on doit prendre dans $V' = V \backslash F$ pour obtenir une forêt induite maximale qui contient F. On peut vérifier facilement que $\binom{n'}{i} \leq 10^{\frac{n'}{5}}$, et cela pour tout $n' \geq 0$ et $i \leq 2$.

Il y a donc, au maximum, $10^{\frac{n-|F|}{5}}$ forêts induites maximales par inclusion dans le graphe complet G, et qui contiennent F. Le théorème est donc correct pour le cas de base.

$k \geq 2$ **:** Maintenant, le nombre de sacs de la décomposition arborescente en arbre de cliques est supérieur à 1, et l'on suppose, par hypothèse de récurrence, que le théorème est vrai pour tous les graphes qui ont une représentation en arbre de cliques dont le nombre de sacs est $k - 1$.

Soit T notre arbre de cliques, avec $k \geq 2$. Soit X_l une clique correspondant à une feuille de l'arbre T, et X_p la clique correspondant au sac parent de X_l dans T. Soit $L = X_l \backslash X_p$ et $C = X_l \cap X_p$. Nous remarquerons au passage que $C = N_G(L)$, et que $X_l = L \cup C$. Par construction, les ensembles L et C sont non-vides, et donc $|L| \geq 1$ et $|C| \geq 1$.

En fait, les sommets de l'ensemble L apparaissent uniquement dans la clique X_l et dans aucune autre clique de G. En particulier, supposons que nous supprimions les sommets de L dans le graphe G, alors on obtient un graphe $G[V \backslash L]$ cordal, et avec une clique maximale en moins, c'est-à-dire avec une représentation en arbre de cliques de $k - 1$ sacs.

Puisque C est une clique, alors chaque forêt induite maximale dans le graphe G correspond à l'un de ces trois cas :

- $|A \cap C| = 2$: Dans ce cas, il n'y a aucuns sommets de la forêt induite A dans l'ensemble L, c'est-à-dire $|A \cap L| = 0$, car sinon, cela créerait un cycle de taille 3.
- $|A \cap C| = 1$: Dans ce cas, et puisque la forêt induite A est maximale, le même argument que précédemment nous permet d'affirmer qu'il y a exactement 1 sommet de l'arbre A dans l'ensemble L, c'est-à-dire $|A \cap L| = 1$.

$|A \cap C| = 0$: Enfin, dans ce cas, et toujours avec les mêmes arguments, nous pouvons affirmer que si $|L| \geq 2$, alors $|A \cap L| = 2$. Attention cependant, car si $|L| = 1$, alors $|A \cap L| = 1$.

Rappelons-nous que, dans le théorème, nous cherchons toutes les forêts induites qui contiennent un certain ensemble F. Soit $\mathcal{A}_G$ l'ensemble de toutes les forêts induites maximales dans G qui contiennent l'ensemble F. Soit $F' = F \backslash X_l = F \backslash (L \cup C)$.

Avant de chercher la borne supérieure pour $|\mathcal{A}_G|$, nous allons essayer de compter toutes les forêts induites maximales qui contiennent F'. Le nombre de forêts que nous obtiendrons sera clairement au moins aussi grand que si nous avions cherché les forêts induites maximales dans G qui contiennent F, avec possiblement plusieurs résultats supplémentaires, où la forêt trouvée ne contiendrait pas les sommets de $F \backslash F'$. Nous verrons plus tard comment prendre en compte ces résultats supplémentaires.

Les trois cas que nous avons décrit pour le nombre de sommets dans $A \cap C$ nous donnent, comme conséquence directe, la borne suivante sur le nombre de forêts induites maximales qui contiennent F', et donc, indirectement, sur $\mathcal{A}_G$:

$$|\mathcal{A}_G| \leq \binom{|C|}{2} \cdot |\mathcal{A}_{G_2}| + |L| \cdot |C| \cdot |\mathcal{A}_{G_1}| + max\left\{1, \binom{|L|}{2}\right\} \cdot |\mathcal{A}_{G_0}|$$

Dans cette formule, $\mathcal{A}_{G_0}$ représente l'ensemble des forêts induites maximales dans un graphe $G_0 = G[V \backslash X_l]$ et qui contiennent F'. $\mathcal{A}_{G_1}$ représente l'ensemble des forêts induites maximales dans un graphe $G_1 = G[V \backslash (X_l \backslash \{u\})]$ contenant $F' \cup \{u\}$, avec u un sommet quelconque de la clique C. Enfin, $\mathcal{A}_{G_2}$ est l'ensemble des forêts induites maximales d'un graphe $G_2 = G[V \backslash (X_l \backslash \{u, v\})]$ contenant $F' \cup \{u, v\}$, où u et v sont deux sommets de la clique C.

Parmi les trois graphes G_0, G_1 et G_2, aucun ne contient de sommets venant de la clique L. Cela signifie que ces trois graphes ont une représentation en arbre de cliques composée de seulement $k-1$ sacs. Si l'on se réfère à notre hypothèse d'induction, il en suit que :

- $|\mathcal{A}_{G_0}| \leq 10^{\frac{n-|X_l|-|F'|}{5}}$
- $|\mathcal{A}_{G_1}| \leq 10^{\frac{n+1-|X_l|-(|F'|+1)}{5}}$
- $|\mathcal{A}_{G_2}| \leq 10^{\frac{n+2-|X_l|-(|F'|+2)}{5}}$

En d'autres mots, on voit que ces trois ensembles contiennent au plus $10^{\frac{n-|X_l|-|F'|}{5}}$ forêts induites maximales par inclusion.

La formule $\binom{|C|}{2} + |L| \cdot |C| + \binom{|L|}{2}$ nous donne le nombre de possibilités de choisir soit deux sommets dans C, soit un sommet dans C et un dans L, ou

encore deux sommets dans L. En clair, cette formule nous donne le nombre de possibilités de choisir deux sommets dans $L \cup C$, c'est-à-dire dans X_l.

$$\binom{|C|}{2} + |C| \cdot |L| + \binom{|L|}{2} = \binom{|L \cup C|}{2} = \binom{|X_l|}{2}$$

Avec la formule précédente, nous pouvons maintenant dire :

$$\mathcal{A}_G \leq 10^{\frac{n-|X_l|-|F'|}{5}} \cdot \binom{|X_l|}{2}$$

Nous allons pouvoir à présent utiliser cette dernière formule pour trouver une borne supérieure au nombre de forêts induites maximales contenant F, et non plus F'. Considérons l'ensemble $F \cap X_l$, puisque $G[F]$ est une forêt et que X_l est une clique, nous savons que $i = |F \cap X_l| \in \{0, 1, 2\}$.

Cela signifie que seul i sommets de la clique X_l peuvent être pré-sélectionnés dans F pour faire partie de la forêt induite aux cotés des sommets de F'. Comme nous avons que $\binom{|X_l|}{2-i} \leq 10^{\frac{|X_l|-i}{5}}$ et que $|F'| + i = |F|$, alors nous pouvons utiliser les formules précédentes pour obtenir :

$$|\mathcal{A}_G| \leq 10^{\frac{n-|X_l|-|F'|}{5}} \cdot \binom{|X_l|}{2-i} \leq 10^{\frac{n-|X_l|-|F|+i+|X_l|-i}{5}} \leq 10^{\frac{n-|F|}{5}}$$

Cela prouve la validité du théorème. □

Corollaire 4. *Tous les graphes cordaux ont au plus* $10^{\frac{n}{5}}$ *ensembles coupe cycles minimaux par inclusion.*

Même si notre but était, en fait, de démontrer ce corolaire 4, remarquez que, dans la pratique, le théorème 16 démontre quelque chose de plus fort, et finalement de plus utile. Si dans l'avenir, nous souhaitons chercher une borne supérieure sur le nombre d'ensembles coupe cycles minimaux par inclusion sur une classe de graphe plus large que la classe des graphes cordaux, et que suite à une série d'opérations, comme des branchements, nous obtenons un graphe cordal avec un certain nombre de sommets déjà éliminés, alors nous pourrons directement utiliser le théorème pour résoudre ce sous-problème, et obtenir ainsi une solution.

5.4 Une borne ajustée pour les co-graphes

Nous allons maintenant démontrer que, sur les co-graphes aussi, la borne supérieure pour le nombre d'ensembles coupe cycles minimaux par inclusion est ajusté à la borne inférieure avec pour les deux valeurs $10^{\frac{n}{5}}$. De la même

manière que dans la section précédente, nous allons travailler avec les forêts induites maximales par inclusion. Bien que la borne supérieure ait la même valeur pour les graphes cordaux et les co-graphes, les deux preuves sont très différentes. En particulier, nous allons avoir besoin, dans notre preuve pour les co-graphes, de la borne supérieure sur le nombre d'ensembles stables maximals par inclusion dans le graphe.

Nous allons commencer par définir une fonction f qui nous aidera à clarifier nos notations dans les démonstrations qui suivront :

$$f(\alpha, \beta, i, j) = \alpha^i + \alpha^j + i \cdot \beta^j + j \cdot \beta^i$$

Nous allons avoir besoin de certaines propriétés de cette fonction f, et surtout, du lemme qui suit :

Lemme 3. *Soit* $\mathcal{B} = \{(1, 10), (2, 8), (3, 7), (4, 5), (5, 4), (7, 3), (8, 2), (10, 1)\}$. *Pour* $\alpha = 10^{\frac{1}{5}}$, $\beta = 3^{\frac{1}{3}}$, $i \geq i' \geq 1$ *et* $j \geq j' \geq 1$, *de telle façon que* (i', j') *soit un couple d'entiers tel que* $(i', j') \in \mathcal{B}$, *alors la fonction est bornée asymptotiquement par* α^{i+j}.

$$f(\alpha, \beta, i, j) \leq \alpha^{i+j}$$

Démonstration. La preuve que nous allons faire est par induction sur le couple (i, j). Remarquons qu'il y a une symétrie dans notre équation, et travailler sur un couple (i, j) ou (j, i) donnera le même résultat. En effet, il est incontestable que :

$$f(\alpha, \beta, i, j) = \alpha^i + \alpha^j + i \cdot \beta^j + j \cdot \beta^i = \alpha^j + \alpha^i + j \cdot \beta^i + i \cdot \beta^j = f(\alpha, \beta, j, i)$$

Le cas de base de notre induction est lorsque le couple $(i, j) \in \mathcal{B}$, et nous allons faire une vérification pour toutes les possibilités. Ce qu'il nous faut, c'est que :

$$\alpha^{-j} \cdot \left(1 + j \cdot \left(\frac{\beta}{\alpha}\right)^i\right) + \alpha^{-i} \cdot \left(1 + i \cdot \left(\frac{\beta}{\alpha}\right)^j\right) \leq 1$$

En résolvant la partie gauche de cette formule, avec toutes les valeurs de $(i, j) \in \mathcal{B}$ possibles, nous obtenons les valeurs : $\{0.978, 0.965, 0.891, 0.997, 0.891, 0.965, 0.978\}$. Le lemme est donc vérifié en ce qui concerne le cas de base.

Supposons par hypothèse de récurrence, que le lemme soit vérifié pour tous les couples $(i - 1, j)$, et montrons que cela est vrai pour (i, j), et cela

pour tous les couples (i, j) tels que $i > i'$ et $j \geq j'$ avec $(i', j') \in \mathcal{B}$. Nous allons devoir distinguer deux cas distincts : si $i \geq 3$ ou si $i \leq 2$.

Supposons d'abord que $i \geq 3$, par hypothèse de récurrence, nous avons que cela fonctionne pour $(i-1, j)$, et donc que :

$$\alpha^{i-1} + \alpha^j + (i-1) \cdot \beta^j + j \cdot \beta^{i-1} \leq \alpha^{i+j-1}$$

Afin de démontrer que $f(\alpha, \beta, i, j) \leq \alpha^{i+j}$, il nous suffit désormais de démontrer que :

$$\alpha^i + \alpha^j + i \cdot \beta^j + j \cdot \beta^i \leq \alpha \cdot (\alpha^{i-1} + \alpha^j + (i-1) \cdot \beta^j + j \cdot \beta^{i-1})$$

Nous pouvons réécrire cette formule de la façon qui suit :

$$\alpha^j \cdot (1-\alpha) + i \cdot \beta^j \cdot \left(1 - \frac{i-1}{i} \cdot \alpha\right) + j \cdot \beta^i \cdot \left(1 - \frac{\alpha}{\beta}\right) \leq 0$$

Nous allons désormais montrer que le lemme est correct en utilisant comme argument que, dans cette dernière formule, toutes les expressions se trouvant entre parenthèses donnent en fait des résultats négatifs. Ce qui nous permettra d'affirmer que la formule est toujours vraie puisque l'addition de trois nombres négatifs sera toujours inférieure à 0.

$(1-\alpha)$: il est clair que tant que $\alpha > 1$, alors $(1-\alpha) < 0$.

$\left(1 - \frac{\alpha}{\beta}\right)$: dans notre fonction, et nos définitions, nous avons que $\alpha > \beta$, et cela implique évidemment que $1 < \frac{\alpha}{\beta}$. Par conséquent, $\left(1 - \frac{\alpha}{\beta}\right) < 0$.

$\left(1 - \frac{i-1}{i} \cdot \alpha\right)$: il nous reste la dernière expression. Nous pouvons la réécrire sous la forme suivante : $\left(1 - \alpha + \frac{\alpha}{i}\right)$. Rappelons que nous avons $\alpha = 10^{\frac{1}{5}} \approx 1.585$, et que nous avons supposé que $i \geq 3$. Dans ce cas, nous avons toujours que $\left(1 - \frac{i-1}{i} \cdot \alpha\right) < 0$.

Il nous reste à montrer que le lemme est correct pour $i \leq 2$. Dans ce cas, le seul couple d'entiers qui peut convenir dans $\mathcal{B}$ est le couple $(1, 10)$ car nous avons précisé en hypothèse de récurrence que $i > i'$ et $j \leq j'$ pour $(i', j') \in \mathcal{B}$. Puisque $j' \leq 10$, nous avons alors que si nous sommes dans le cas où $i \leq 2$, alors nous avons forcément que $j \geq 10$.

Revenons à la formule :

$$\alpha^j \cdot (1-\alpha) + i \cdot \beta^j \cdot \left(1 - \frac{i-1}{i} \cdot \alpha\right) + j \cdot \beta^i \cdot \left(1 - \frac{\alpha}{\beta}\right) \leq 0$$

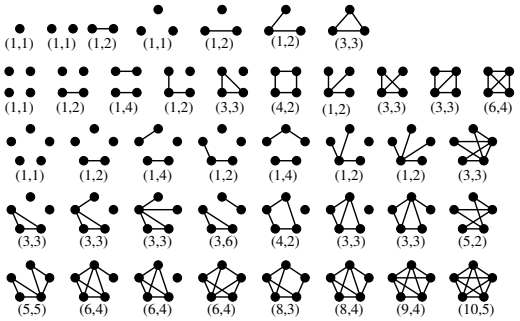
Si l'on calcule la valeur de la partie gauche avec $(i, j) = (2, 10)$, on obtient une valeur inférieure à 0.81. Il nous reste alors à prouver que cela fonctionne si $j > 10$.

Dans ce cas, nous allons simplement utiliser la symétrie de notre formule f. En effet, rappelons que $f(\alpha, \beta, i, j) = f(\alpha, \beta, j, i)$. Nous avons déjà démontré que pour $i > 10 \geq 3$, s'il y a un couple $(i', j') \in \mathcal{B}$, alors le lemme est correct. Si $i > 10$, alors le couple $(10, 1) \in \mathcal{B}$ pourra toujours convenir. □

Cela termine la démonstration de notre lemme. Nous sommes désormais prêts à démontrer notre résultat sur les co-graphes.

Théorème 17. *Le nombre de forêts induites maximales dans un co-graphe de n sommets est au plus de* $10^{\frac{n}{5}}$.

Démonstration. Nous allons maintenant démontrer ce théorème par induction sur le nombre de sommets d'un co-graphe. Comme cas de base, nous considérerons le cas de tous les co-graphes composés d'au plus cinq sommets. Tous les co-graphes d'au plus 5 sommets sont connus et sont listés dans la figure suivante. On peut aussi retrouver cette liste sur internet : http ://mathworld.wolfram.com/Cograph.html. En examinant tous les graphes de cette figure, on peut facilement vérifier qu'aucun d'entre-eux ne possède plus de $10^{\frac{n}{5}}$ forêts induites maximales par inclusion.



Liste des co-graphes de moins de cinq sommets. Pour chaque graphe, on donne un couple d'entiers (x, y) *: x est le nombre de forêts induites maximales par inclusion, y est le nombre d'ensembles stables maximals par inclusion.*

Considérons maintenant que, comme hypothèse de récurrence, le théorème soit vrai pour tous les co-graphes dont le nombre de sommets est au plus $n - 1$, et soit $G = (V, E)$ un co-graphe constitué de $n \leq 6$ sommets. D'après la définition de la construction d'un co-graphe, que nous avons faaite dans l'introduction [1.3.2], nous savons que le graphe G peut être partitionné en deux sous-graphes G_1 et G_2 tels que $G_1 \uplus G_2$ ou que $G_1 \bowtie G_2$.

Si le graphe G n'est pas connexe, les deux sous-graphes G_1 et G_2 sont alors des co-graphes composés de moins que $n - 1$ sommets. Par hypothèse

de récurrence, nous pouvons donc affirmer que G_1 et G_2 n'ont pas plus de, respectivement, $10^{\frac{n_1}{5}}$ et $10^{\frac{n_2}{5}}$ ensembles coupe cycles minimaux par inclusion, avec $n_1 = |V(G_1)|$ et $n_2 = |V(G_2|$. Il est évident que $n_1 + n_2 = n$.

Le nombre de forêts induites maximales dans le graphe G étant, dans le cas d'un graphe non-connexe, le nombre de forêts induites maximales de G_1 multiplié par le nombre de forêts induites maximales de G_2, nous obtenons $10^{\frac{n_1}{5}} \cdot 10^{\frac{n_2}{5}} \leq 10^{\frac{n_1+n_2}{5}} \leq 10^{\frac{n}{5}}$.

Désormais, nous pouvons être sur que le graphe G est connexe. De la même manière, les deux sous-graphes G_i (pour $i \in \{1, 2\}$) possèdent chacun au plus $10^{\frac{n_i}{5}}$ forêts induites maximales par inclusion. Puisque le graphe G est une jonction entre le graphe G_1 et le graphe G_2, nous savons aussi que toutes les arêtes possibles sont présentes entre les sommets du sous-graphe G_1 et les sommets du sous-graphe G_2. En particulier, nous savons que tous les sommets de $V(G_1)$ sont adjacents à tous les sommets de $V(G_2)$.

Appelons A un sous-ensemble de sommets de G tel que $G[A]$ soit une forêt induite maximale par inclusion dans le graphe G. Soient $i, j \in \{1, 2\}$ et $i \neq j$, nous pouvons observer les propriété suivantes :

- $|A \cap V(G_i)| \geq 2$: Si l'ensemble A possède plus de deux sommets dans un des sous-graphes G_i, alors on a que $|A \cap V(G_j)| \leq 1$, sinon on aurait un cycle de longueur 4, ce qui contredit le fait que l'ensemble A permette de définir une forêt.
- $|A \cap V(G_i)| = 1$: Si l'ensemble A possède un et un seul sommet dans le sous-graphe G_i, alors il ne peut y avoir deux sommets adjacents du sous-graphe G_j qui fassent partie de A, sinon cela créerait un cycle de taille 3. Cela implique donc que $A \cap V(G_j)$ est un ensemble stable maximal par inclusion.
- $A \cap V(G_i) = \emptyset$: Si l'un des deux sous-graphes ne contient aucuns sommets de l'ensemble A, alors $A \subseteq V(G_j)$ et dans ce cas, $G[A]$ est une forêt induite maximale par inclusion dans le sous-graphe G_j.

Appelons $\mathcal{I}_n$ le nombre maximum d'ensembles stables maximals par inclusion dans un graphe de n sommets. Les trois précédentes observations nous montrent qu'une forêt induite maximale par inclusion dans le co-graphe G est soit une forêt induite maximale dans l'un des deux sous-graphes G_i, soit un sommet isolé de $V(G_i)$ associé à un ensemble stable maximal par inclusion dans le sous-graphe G_j, avec $i, j \in \{1, 2\}$ et $i \neq j$. Si nous appelons $\mathcal{A}_g$ l'ensemble des forêts induites maximales par inclusion dans le graphe G, cela nous donne la formule suivante :

$$|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + |\mathcal{A}_{G_2}| + n_1 \cdot \mathcal{I}_{G_2} + n_2 \cdot \mathcal{I}_{G_1}$$

Grâce à Moon et Moser [76], nous savons que $\mathcal{I}_n \leq 3^{\frac{n}{3}}$. En utilisant cette borne sur le nombre d'ensembles stables maximals par inclusion dans un graphe, et en utilisant notre hypothèse du récurrence sur le nombre de forêts induites maximales par inclusion dans un co-graphe, nous pouvons en déduire la formule suivante :

$$|\mathcal{A}_G| \leq 10^{\frac{n_1}{5}} + 10^{\frac{n_2}{5}} + n_1 \cdot 3^{\frac{n_2}{3}} + n_2 \cdot 3^{\frac{n_1}{3}}$$

Cette dernière formule correspond au lemme 3 que nous avons démontré un peu plus tôt dans ce chapitre. Il nous suffit maintenant d'appliquer le lemme 3 à notre formule, et cela nous donnera immédiatement que $|\mathcal{A}_G| \leq 10^{\frac{n}{5}}$ pour une large possibilité de valeurs possibles de $n = n_1 + n_2$. Cependant, avant de pouvoir faire cela, nous devons démontrer que, pour toutes les valeurs qui sont inférieures à celles définies dans le cas de base du lemme, nous avons bien là aussi $|\mathcal{A}_G| \leq 10^{\frac{n_1+n_2}{5}} = 10^{\frac{n}{5}}$.

n_0 \ n_1	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	≤ 5	≤ 5	≤ 5	≤ 5	$C1$	$R1$	$R2$	$R3$	$R4$	OK
2	≤ 5	≤ 5	≤ 5	$C2$	$C4$	$C6$	$C7$	OK	OK	OK
3	≤ 5	≤ 5	$C3$	$C5$	$C8$	$C9$	OK	OK	OK	OK
4	≤ 5	$C2$	$C5$	$C10$	OK	OK	OK	OK	OK	OK
5	$C1$	$C4$	$C8$	OK	OK	OK	OK	OK	OK	OK
6	$R1$	$C6$	$C9$	OK	OK	OK	OK	OK	OK	OK
7	$R2$	$C7$	OK	OK	OK	OK	OK	OK	OK	OK
8	$R3$	OK	OK	OK	OK	OK	OK	OK	OK	OK
9	$R4$	OK	OK	OK	OK	OK	OK	OK	OK	OK
10	OK	OK	OK	OK	OK	OK	OK	OK	OK	OK

Ce tableau de vérification des cas permet de s'assurer que toutes les valeurs de n_1 et n_2 sont vérifiées.

Dans le but de nous aider à nous assurer que tous les cas sont vérifiés, nous utilisons le tableau de vérification des cas ci-dessus.

Pour toutes les valeurs de n_1 et n_2 telles que $n_1 + n_2 \leq 5$, nous savons déjà que, grâce au cas de base de notre induction et à la liste des co-graphes de moins de 5 sommets, que nous avons bien $|\mathcal{A}_G| \leq 10^{\frac{n}{5}}$. Ces cas correspondent à toutes les cases marquées "≤ 5" dans le tableau de vérification des cas.

Seulement, il nous reste à examiner un certain nombre de cas se trouvant entre les cas où $n_1 + n_2 \leq 5$ et les cas où $n_1 \leq i$ et $n_2 \leq j$ avec $(i, j) \in \mathcal{B} = \{(1, 10), (2, 8), (3, 7), (4, 5), (5, 4), (7, 3), (8, 2), (10, 1)\}$.

Pour toutes les valeurs de $\mathcal{B}$, ainsi que pour toutes les valeurs supérieures, nous pouvons appliquer le lemme 3. Toutes les paires de valeurs (n_1, n_2) pouvant être vérifiées par le lemme 3 sont marquées "OK" dans le tableau de vérification des cas. Il est évident que, pour toutes les valeurs en dehors du tableau, le lemme fonctionne aussi.

Commençons par le cas où $n_1 = 1$ et $n_2 = 5$. Le sous-graphe G_1 ne possède qu'un seul sommet, appelons ce sommet x. Rappelons que l'ensemble A représente un sous-ensemble de sommets de G tel que $G[A]$ est une forêt induite maximale par inclusion.

- $x \in A$: Si le sommet unique du sous-graphe G_1 est inclus dans une forêt induite maximale dans G, alors on sait que l'on ne peut avoir deux sommets adjacents de G_2 dans A, puisque x est, en fait, un sommet universel. Le nombre de forêts induites maximales dans G correspond alors au nombre d'ensembles stables maximals par inclusion dans G_2.
- $x \notin A$: Si le sommet unique de G_1 n'est pas inclus dans une forêt induite maximale de G, alors le nombre de forêts induites maximales dans G correspond au nombre de forêts induites maximales dans G_2.

En utilisant la liste des co-graphes de moins de 5 sommets, nous pouvons voir que le nombre de forêts induites maximals par inclusion, plus le nombre d'ensembles stables maximals par inclusion ne dépasse jamais 15 dans un graphe de 5 sommets, nous permettant de conclure que $|\mathcal{A}_G| \leq |\mathcal{A}_{G_2}| + \mathcal{I}_{G_2}$. Puisque $15 \leq 10^{\frac{6}{5}}$, alors le théorème est vrai pour ce cas. Ce cas particulier correspond aux cases notés $C1$ dans le tableau de vérification des cas.

Considérons maintenant le cas où $n_1 = 2$ et $n_2 = 4$. Dans un premier temps, observez que, si les deux sommets de G_1 sont adjacents, alors ces deux sommets sont aussi universels. Il est alors possible de modifier la partition G_1, G_2 en faisant passer l'un des sommets de G_1 dans le graphe G_2. Ce cas est donc similaire au cas où $n_1 = 1$ et $n_2 = 5$, qui est le cas $C1$ que nous avons déjà traité précédemment.

Supposons donc que les deux sommets de G_1 ne sont pas adjacents.

- $|A \cap V(G_1)| = 0$: si la forêt induite maximale ne contient aucun des deux sommets de G_1, alors il s'agit aussi d'une forêt induite maximale dans G_2.
- $|A \cap V(G_1)| = 1$: si la forêt induite maximale contient un seul sommet de G_1, alors les sommets $A \cap V(G_2)$ de G_2 inclus dans la forêt forment un ensemble stable maximal par inclusion.
- $|A \cap V(G_1)| = 2$: si la forêt induite maximale contient les deux sommets de G_1, alors on ne peut avoir qu'un seul sommet dans G_2.

Cela nous donne la formule :

$$|\mathcal{A}_G| \leq |\mathcal{A}_{G_2}| + 2 \cdot \mathcal{I}_{G_2} + n_2$$

Si l'on regarde les graphes dans la liste des co-graphes de moins de 5 sommets, qui ont au plus 4 sommets, on voit que, pour la plupart, le nombre de forêts induites maximals par inclusion additionné au nombre d'ensembles stables maximales par inclusion ne dépasse pas 11. Dans ce cas, on a $11 + 4 = 15 \leq 10^{\frac{6}{5}}$. Il y a cependant une exception, lorsque le graphe de 4 sommets est un K_4. Si le graphe G_2 est un K_4, on peut observer qu'aucune forêt induite maximale de G ne peut être obtenue avec un seul sommet de G_1. On peut alors ignorer le terme $2 \cdot \mathcal{I}_{G_2}$, et voir que dans ce cas :

$$|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + n_1 \leq 6 + 4 \leq 10$$

Cela complète le cas $n_1 = 2$ et $n_2 = 4$ qui est noté "$C2$" dans le tableau de vérification des cas.

Le cas suivant que nous avons à traiter est le cas où $n_1 = n_2 = 3$. Nous pouvons utiliser le même argument que précédemment pour mettre de coté les cas où il y aurait dans le graphe un sommet universel. Du fait de la symétrie, un sommet universel pourrait très bien passer d'un sous-graphe à l'autre, et nous amener dans les cas précédemment traités.

Supposons donc qu'aucuns des deux sous-graphes G_1 et G_2 ne disposent de sommet universel. Nous pouvons voir, dans la liste des co-graphes de moins de 5 sommets, que les co-graphes de trois sommets qui n'ont pas de sommet universel disposent d'au plus une forêt induite maximale par inclusion, et d'au plus deux ensembles stables maximals par inclusion. Cela nous donne donc que :

$$|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + |\mathcal{A}_{G_2}| + n_1 \cdot \mathcal{I}_{G_2} + n_2 \cdot \mathcal{I}_{G_1} \leq 1 + 1 + 3 \cdot 2 + 3 \cdot 2 = 14 \leq 10^{\frac{6}{5}}$$

Ce cas est dénoté par l'étiquette "$C3$" dans le tableau de vérification des cas.

Nous allons maintenant nous intéresser à un argument plus général pour les cas pour lesquels l'un des deux sous-graphes (G_1 ou G_2) ne contient qu'un seul sommet, ce sommet étant, en fait, un sommet universel. Disons donc que nous traitons les cas avec $n_1 = 1$ et $n_2 = n - 1 \geq 6$. Comme nous avons pu le voir en traitant le cas "$C1$", nous avons dans ce cas $|\mathcal{A}_G| \leq \mathcal{A}_{G_2} + \mathcal{I}_{G_2}$.

En utilisant l'hypothèse de récurrence, nous pouvons réécrire que $|\mathcal{A}_{G_1}| \leq 10^{\frac{n_1}{5}} \leq 10^{\frac{n-1}{5}}$, ce qui nous permettra d'obtenir la borne souhaitée si nous pouvons démontrer que $10^{\frac{n-1}{5}} + 3^{\frac{n-1}{3}} \leq 10^{\frac{n}{5}}$. Cette équation peut aussi s'écrire $\left(\frac{3^{\frac{1}{3}}}{10^{\frac{1}{5}}}\right)^{n-1} \leq 10^{\frac{1}{5}} - 1$, ce qui est vrai pour tout $n \geq 7$.

De ce fait, cela peut être utilisé pour démontrer tous les n supérieurs à 7, mais seulement à condition que la valeur du théorème soit déjà validée pour toutes les valeurs possibles du couple (n'_1, n'_2) telles que $n'_1 + n'_2 \leq n - 1$. Pour

l'instant, nous avons, dans notre tableau de vérification des cas, validé tous les couples tels que $n'_1 + n'_2 \leq 6$, et nous pouvons donc, grâce à cet argument, valider les cas $(6, 1)$ et $(1, 6)$, que nous marquons "$R1$" dans le tableau. Par contre, il nous reste encore à valider un certain nombre de couples tels que $7 \leq n'_1 + n'_2 \leq 10$, mais à mesure que cela sera fait, alors notre argument permettra de valider les cases que nous avons successivement notées "$R2$", "$R3$" et "$R4$".

Le cas suivant que nous avons à traiter est lorsque $n_1 = 2$ et $n_2 = 5$. Nous pouvons réutiliser les mêmes arguments que dans le cas "$C2$" pour supposer que les deux sommets du graphe G_1 ne sont pas reliés par une arête. De plus, nous pouvons reprendre dans le cas "$C2$" la formule $|\mathcal{A}_G| \leq |\mathcal{A}_{G_2}| + 2 \cdot \mathcal{I}_{G_2} + n_2$. Si l'on regarde les graphes de 5 sommets dans la liste des co-graphes de moins de 5 sommets, on constate que le seul à avoir un nombre d'ensembles stables maximals par inclusion supérieur à 5 est le cas où le graphe est une union disjointe entre un triangle et une arête. Si le graphe G_2 est constitué de cette façon, alors on aura :

$$|\mathcal{A}_G| \leq 3 + 2 \cdot 6 + 5 = 20 < 10^{\frac{7}{5}}$$

Pour toutes les autres possibilités, avec un graphe de cinq sommets au plus, le nombre d'ensembles stables maximals par inclusion ne dépasse pas 5, et le plus grand nombre d'ensembles coupe cycles est de 10. Nous avons alors :

$$|\mathcal{A}_G| \leq 10 + 2 \cdot 5 + 5 = 25 < 10^{\frac{7}{5}}$$

Le théorème est donc valide aussi pour ces cas que nous marquons "$C4$" dans le tableau de vérification des cas.

Le prochain cas que nous allons traiter est le cas $n_1 = 3$ et $n_2 = 4$. Rappelons que, pour les mêmes arguments que précédemment, il n'y a pas, dans le sous-graphe G_1, de sommet universel. Sinon nous pourrions changer la partition de notre graphe afin de le faire correspondre aux cas précédents.

Si l'on observe la liste des co-graphes de moins de 5 sommets, on constate que le sous-graphe G_1 dispose d'au plus une forêt induite maximale par inclusion, et d'au plus deux ensembles stables maximals par inclusion.

Nous allons utiliser la formule :

$$|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + |\mathcal{A}_{G_2}| + n_1 \cdot \mathcal{I}_{G_2} + n_2 \cdot \mathcal{I}_{G_1}$$

Si le graphe G_2 est un K_4, alors il n'y a aucune forêt induite maximale qui ne soit constituée d'un sommet unique dans le sous-graphe G_1 associé à

un ensemble stable maximal dans le sous-graphe G_2. La composante $n_1 \cdot \mathcal{I}_{G_2}$ peut être ignorée. On a alors que :

$$|\mathcal{A}_G| \leq 1 + 6 + 4 \cdot 2 = 15 < 10^{\frac{7}{5}}$$

Dans tous les autres cas, le nombre maximum de forêts induites maximales est de 4, et le nombre maximum d'ensembles stables maximals est de 4 aussi. La formule devient alors :

$$|\mathcal{A}_G| \leq 1 + 4 + 3 \cdot 2 + 3 \cdot 4 = 25 < 10^{\frac{7}{5}}$$

Cela valide, à leurs tours, ces cas que nous marquons "$C5$" dans le tableau de vérification des cas.

Remarquez qu'à cette étape, toutes les possibilités de couples (n_1, n_2) tels que $n_1 + n_2 \leq 7$ sont validés. Grâce à l'argument que nous avions décrit à propos des cas où le sous-graphe G_1 n'aurait qu'un seul sommet, nous pouvons valider directement les cases marquées "$R2$" dans le tableau.

Regardons maintenant les deux cas correspondant à $n_1 = 2$ et $n_2 \in \{6, 7\}$. Le nombre de forêts induites maximales par inclusion, et le nombre d'ensembles stables maximals par inclusion sont, bien évidement, des entiers. En utilisant l'hypothèse de récurrence, nous pouvons obtenir la formule pour $n_1 = 2$ que nous avons déjà utilisée dans le cas "$C2$" :

$$|\mathcal{A}_G| \leq |\mathcal{A}_{G_2}| + 2 \cdot \mathcal{I}_{G_2} + n_2 \leq \lfloor 10^{\frac{n_2}{5}} \rfloor + 2 \cdot \lfloor 3^{\frac{n_2}{3}} \rfloor + n_1$$

Ce que nous souhaitons, c'est démontrer que cela est inférieur à $10^{\frac{n_2+2}{5}}$.
$n_2 = 6$: s'il y a 6 sommets dans le graphe G_2, alors la formule devient :

$$\lfloor 10^{\frac{6}{5}} \rfloor + 2 \cdot \lfloor 3^{\frac{6}{3}} \rfloor + 6 \leq 15 + 2 \cdot 9 + 6 \leq 39 < 10^{\frac{8}{5}}$$

$n_2 = 7$: s'il y a 7 sommets dans le graphe G_2, alors la formule devient :

$$\lfloor 10^{\frac{7}{5}} \rfloor + 2 \cdot \lfloor 3^{\frac{7}{3}} \rfloor + 7 \leq 25 + 2 \cdot 12 + 7 \leq 56 < 10^{\frac{9}{5}}$$

Ces deux cas sont donc eux-aussi valides, et nous les marquons "$C6$" et "$C7$" dans le tableau de vérification des cas.

Nous allons continuer avec les cas $n_1 = 3$ et $n_2 \in \{5, 6\}$. Comme nous l'avons vu de nombreuses fois maintenant, nous pouvons considérer qu'il n'y a pas de sommet universel dans le sous-graphe G_1 sinon la partition peut être redéfinie pour correspondre à l'un des cas déjà traités.

$n_2 = 5$: nous utilisons les valeurs pour le nombre de forêts induites maximales par inclusion et le nombre d'ensembles stables maximals par inclusion que nous pouvons retrouver dans la liste des co-graphes de moins de 5 sommets. On obtient alors que :

$$|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + |\mathcal{A}_{G_2}| + n_1 \cdot \mathcal{I}_{G_2} + n_2 \cdot \mathcal{I}_{G_1} \leq 1 + 10 + 5 \cdot 2 + 3 \cdot 6 = 39 < 10^{\frac{8}{5}}$$

$n_2 = 6$: l'argument est en fait le même que pour $n_2 = 5$, et la formule obtenue est alors :

$$|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + |\mathcal{A}_{G_2}| + n_1 \cdot \mathcal{I}_{G_2} + n_2 \cdot \mathcal{I}_{G_1} \leq 1 + 10^{\frac{6}{5}} + 6 \cdot 2 + 3 \cdot 3^{\frac{6}{5}} = 55 < 10^{\frac{9}{5}}$$

Les deux cas sont notés respectivement "$C8$" et "$C9$" dans le tableau de vérification des cas.

Le dernier cas qu'il nous reste à considérer est le cas avec $n_1 = n_2 = 4$. Encore une fois, si l'un des deux sous-graphes dispose d'un sommet universel, alors la partition peut être refaite différemment pour que le graphe corresponde à un cas déjà traité.

Si l'on regarde les valeurs données dans la liste des co-graphes de moins de 5 sommets, on voit que les deux sous-graphes ont au plus 4 forêts induites maximales, et au plus 4 ensembles stables maximals par inclusion. On peut même remarquer que, soit le sous-graphe (G_1 ou G_2) est un cycle de longueur 4, soit il a au plus 3 forêts induites maximales par inclusion.

- Si G_1 est un cycle de longueur 4, alors il y a dans G_1 4 forêts induites maximales et seulement 2 ensembles stables maximals. La formule nous donne alors :

$$|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + |\mathcal{A}_{G_2}| + n_1 \cdot \mathcal{I}_{G_2} + n_2 \cdot \mathcal{I}_{G_1} \leq 4 + 4 + 4 \cdot 4 + 4 \cdot 2 = 32 < 10^{\frac{8}{5}}$$

- Si G_1 n'est pas un cycle de longueur 4, alors il a au plus 3 forêts induites maximales, et 4 ensembles stables. La formule nous donne donc :

$$|\mathcal{A}_G| \leq |\mathcal{A}_{G_1}| + |\mathcal{A}_{G_2}| + n_1 \cdot \mathcal{I}_{G_2} + n_2 \cdot \mathcal{I}_{G_1} \leq 3 + 4 + 4 \cdot 4 + 4 \cdot 4 = 39 < 10^{\frac{8}{5}}$$

Cela nous permet de valider le cas marqué "$C10$" dans le tableau de vérification des cas.

Remarquez que nous avons démontré maintenant le théorème pour toutes les valeurs possibles du couple (n_1, n_2), et que grâce à l'argument que nous avions donné pour le cas "$R1$", les cas "$R3$" et "$R4$" sont eux-aussi validés.

Tous les cas étant inférieurs aux valeurs de base requises dans le lemme 3 sont donc valides, et nous pouvons sans risques appliquer le lemme pour assurer que le théorème est correct, quelles que soient les valeurs de n_1 ou n_2. Cela conclut notre démonstration. □

5.5 Les graphes d'intervalles circulaires

Avant de conclure, nous allons vous faire part de quelques réflexions intéressantes sur les graphes d'intervalles circulaires. Comme nous l'avons vu dans les préliminaires de ce chapitre 5.2.1, les graphes d'intervalles circulaires sont des graphes d'intersections entre des arcs autour d'un cercle, et ils sont très proches des graphes cordaux.

La raison pour la quelle nous nous intéressons aux graphes de permutations circulaires est que le graphe donné en exemple par Fomin et al. [39] pour définir une famille de graphes permettant d'établir une borne inférieure sur le nombre d'ensembles coupe cycles minimaux par inclusion dans un graphe est un graphe d'intervalles circulaires, ou plutôt une union disjointe de plusieurs graphes d'intervalles circulaires.

Le graphe donné en exemple est un graphe de 10 sommets, disposant de 105 ensembles coupe cycles minimaux par inclusion, ce qui permet, en faisant des copies disjointes de ce graphe, d'obtenir une famille infinie de graphes tels que le nombre d'ensembles coupe cycles minimaux par inclusion soit au moins $105^{\frac{n}{10}}$, lorsque le nombre de copies est de $\frac{n}{10}$.

Cependant, remarquez que l'union disjointe de graphes d'intervalles circulaires ne donne pas forcément un graphe d'intervalles circulaires. En particulier, la famille de graphes que nous venons de décrire n'est pas une famille de graphes d'intervalles circulaires et ne permet donc pas de définir de borne inférieure pour cette classe de graphe.

Par conséquent, deux questions intéressantes se posent :

- Quel est la borne supérieure pour le nombre d'ensembles coupe cycles minimaux par inclusion pour une collection de graphes d'intervalles circulaires ? Se pourrait-il qu'elle soit ajustée à la borne inférieure ?
- Combien y-a-t-il d'ensembles coupe cycles au maximum dans un graphe de permutations circulaires ?

Comme nous l'avons vu dans leurs définitions, les graphes de permutations circulaires disposent d'une représentation en cycle de cliques, avec au plus n cliques. Certaines de ces cliques, si on les supprime, coupent le cycle et ne laissent qu'une représentation en arbre de cliques ? Nous appellerons de telles cliques les cliques coupantes. Lorsqu'une clique coupante est supprimée, le graphe restant est un graphe cordal. En effet, nous avons vu, dans les définitions, qu'un graphe disposant d'une représentation en arbre de cliques est un graphe cordal.

Si l'on cherche un ensemble coupe cycle minimal, alors il est évident qu'au moins une des cliques coupantes doit être entièrement dans l'ensemble coupe cycle.

Nous savons déjà, grâce aux résultats de ce chapitre, que le nombre d'ensembles coupe cycles minimaux par inclusion (égal au nombre de forêts induites maximales) dans un graphe cordal est au plus $10^{\frac{n}{5}}$. Sachant qu'il y a au plus n cliques dans le graphe d'intervalles circulaires, alors on peut en déduire directement qu'il y a, au plus, $n \cdot 10^{\frac{n}{5}}$ ensembles coupe cycles minimaux par inclusion dans un graphe d'intervalles circulaires.

5.6 Conclusion

Nous avons montré que pour les graphes cordaux et les co-graphes, les bornes supérieures et inférieures, pour le nombre maximum d'ensembles coupe cycles minimaux par inclusion dans un graphe à n sommets, sont ajustées, et que leur valeur est de $10^{\frac{n}{5}}$. Grâce à l'algorithme de B. Schwikowski et E. Speckenmeyer [90], tous les ensembles coupe cycles minimaux par inclusion peuvent être énumérés en temps $O^*(10^{\frac{n}{5}})$ sur les graphes cordaux, les co-graphes, mais aussi les graphes d'intervalles circulaires.

On remarquera que, grâce à un argument de B.M. Bui-Xuan, J.A. Telle et M. Vatshelle [16], le comptage des ensembles coupe cycles minimaux par inclusion sur les co-graphes peut s'effectuer en temps polynomial.

Serait-il possible de compter les ensembles coupe cycles minimaux par inclusion en temps polynomial dans l'une de nos trois classes de graphe ? Ou juste dans l'une d'elles ? Il semble possible, par exemple, d'utiliser l'algorithme décrit par B.M. Bui-Xuan, J.A. Telle et M. Vatshelle [16], permettant de trouver un ensemble coupe cycle de taille minimum dans un co-graphe et de l'adapter pour compter les ensembles coupe cycles minimaux par inclusion en temps polynomial sur les co-graphes. Les graphes d'intervalles circulaires soulèvent quant à eux de nombreuses questions. Quelle est la borne supérieure sur le nombre d'ensembles coupe cycles minimaux par inclusion pour une union disjointe de graphes d'intervalles circulaires ? Quelle est la meilleure borne inférieure que l'on puisse trouver pour une famille de graphes d'intervalles circulaires ?

Quatrième partie

Classifications de complexités

Chapitre 6

Classification de complexité pour domination et variantes

6.1 Introduction

Au cours de cette thèse, nous avons beaucoup utilisé les structures et les propriétés particulières des classes de graphe afin de pouvoir, sur ces même classes de graphe, trouver des algorithmes plus rapides que dans le cas général. D'une certaine manière, les restrictions que l'on définit sur les familles de graphes étudiées nous permettent de simplifier les problèmes.

Ainsi, à chaque fois que l'on détermine un certain nombre de particularités, nous permettant de définir une famille de graphes plus restreinte que dans le cas général, nous avons une chance d'avoir, dans le même temps, défini une famille de graphes sur laquelle un problème difficile habituellement devient beaucoup plus simple.

Considérons le problème des ensembles dominants de taille minimum. Dans le cas général, ce problème est un problème NP-complet bien connu. Le meilleur algorithme à ce jour pour le résoudre est celui de Y. Iwata [55] qui récemment a réussi à améliorer celui de Y. Van Rooij [93] dans le chapitre cinq de sa thèse. Supposons que l'on choisisse de résoudre ce problème sur une famille de graphes dont une propriété particulière est que tous les graphes de cette famille sont des cliques. Dans ce cas, chaque sommet dominant l'intégralité du graphe, le problème devient excessivement simple, et même trivial.

Bien que la famille de graphes de cet exemple soit vraiment trop restreinte pour rendre l'étude de ce problème intéressante, elle nous permet de montrer que, lorsque les restrictions sont suffisantes, un problème qui est NP-complet sur le cas général peut devenir solvable en temps polynomial (voire même

dans ce cas, en temps constant).

Ces constatations nous amènent alors à une série de questions naturelles : où se trouve la limite à partir de laquelle un problème n'est plus NP-difficile ? Quelles sont les restrictions minimales à donner sur une famille de graphes pour qu'un problème, NP-complet dans le cas général, devienne solvable en temps polynomial ?

Alors que l'une des grandes questions ouvertes à l'heure actuelle est de déterminer si l'ensemble des problèmes déterministes polynomiaux est différent ou non de l'ensemble des problèmes non-déterministes polynomiaux, pourrait-on, grâce aux classes de graphes, définir une sorte de frontière entre ces deux ensembles ?

6.1.1 Les sous-graphes interdits

Afin de pouvoir définir plus aisément cette frontière entre les classes de graphe sur lesquelles les problèmes restent NP-complets, ou deviennent polynomiaux, nous avons besoin de pouvoir définir des familles de graphes que nous puissions comparer les unes avec les autres. Pour ce faire, nous définirons les classes de graphe que nous utiliserons dans nos classifications par un certain nombre de sous-graphes induits interdits. La notion de sous-graphe induit est à été définie pendant l'introduction [1.3.1]. Certaines des classes de graphe que nous avons utilisé jusque la peuvent être redéfinies sous forme de classes à sous-graphes induits interdits. Par exemple, les co-graphes sont des graphes sans P_4, et les graphes splits sont des graphes sans C_4, sans C_5 et sans $2K_2$.

Le fait d'interdire des sous-graphes pour définir nos classes de graphe a plusieurs avantages, mais le principal est de nous permettre d'établir entre les différentes familles de graphes une relation d'ordre partiel, sous forme de relation d'inclusion entre les classes de graphe.

Deux familles de graphes sont comparables si (et seulement si) les sous-graphes induits interdits qui les définissent sont comparables, et deux sous-graphes induits interdits sont comparables si l'un est un sous graphe induit de l'autre.

$\subseteq$

Par exemple, le graphe P_3 est un sous-graphe induit du graphe $K_{1,3}$. La classe des graphes sans P_3 est comparable à la classe des graphes sans $K_{1,3}$

puisque les sous-graphes interdits définissant ces classes de graphe peuvent êtres comparés. Nous remarquerons aussi que si un graphe est sans P_3, alors il est aussi sans $K_{1,3}$, et il y a alors une relation d'inclusion entre les deux familles de graphe de la manière qui suit :

$$(\text{sans-}P_3) \subseteq (\text{sans-}K_{1,3})$$

Par contre, la relation d'ordre n'est que partielle, car entre deux graphes distincts, la relation faisant de l'un un sous-graphe induit de l'autre n'existe pas toujours. La famille des graphes sans griffes (sans $K_{1,3}$) ne peut par exemple pas être comparée à la famille des graphes sans triangles (sans C_3).

Relation d'ordre partiel sur les classes de graphe. Soit H et H' deux graphes tels que le graphe H est un sous-graphe induit du graphe H'. La relation entre les deux classes de graphe définies comme étant la classe des graphes sans H et la classe des graphes sans H' se comporte de la façon qui suit :

$$(\text{sans-}H) \subseteq (\text{sans-}H')$$

Cette relation d'ordre partiel sur les classes de graphe, définies par des sous-graphes interdits et une relation d'inclusion entre les différentes familles d'ensembles, dispose d'une propriété très intéressante sur le plan de la complexité des problèmes.

En effet, supposons que l'on ait démontré qu'un problème P est NP-complet sur la famille des graphes sans H. Comme tous les graphes de la famille des graphes sans H font aussi partie de la famille des graphes sans H', alors il ne peut y avoir d'algorithme polynomial pour résoudre le problème P sur la famille des graphes sans H'. Sinon ce même algorithme serait aussi capable de résoudre le problème P sur la famille des graphes sans H, contredisant le fait que ce problème est NP-complet sur cette famille de graphes. Nous pouvons en conclure que si H est un sous-graphe induit de H', alors si le problème P est NP-complet sur la famille de graphes définie avec le graphe H comme un sous-graphe interdit, cela implique directement que le problème P est NP-complet aussi sur la famille de graphes définie avec H' comme un sous-graphe interdit.

Cette propriété fonctionne dans les deux sens, et si l'on suppose que l'on dispose d'un algorithme polynomial permettant de résoudre le problème P' sur la famille des graphes sans H', alors cet algorithme peut très bien prendre en entrée un graphe de la famille des graphes sans H. Il existe donc un algorithme polynomial permettant de résoudre le problème P' sur la famille des graphes sans H. Nous pouvons en conclure que si H est un sous-graphe induit de H', alors si le problème P' est polynomial sur la famille des graphes

sans H', cela implique directement que ce problème P' est polynomial sur la famille des graphes sans H.

Avec cette propriété, il nous reste à déterminer quels sont les sous-graphes interdits les plus grands possibles tels que un problème P soit solvable en temps polynomial, et quels sont les sous-graphes interdits les plus petits tels que le problème P reste NP-complet. Si nous arrivons à déterminer une frontière de sorte que pour toutes classes de graphe, nous soyons capable de la positionner parmi les classes de graphe pour lesquelles le problème P est NP-complet, ou parmi celles pour lesquelles le problème P est polynomial, alors nous aurons réalisé une dichotomie sur les classes de graphe pour le problème P.

Bien entendu, pour chaque problème P différent, la classification sur les classes de graphe sera typiquement différente. Ainsi, la classification pour le problème d'Ensemble stable ne sera pas la même que celle pour le problème d'Ensemble dominant par exemple.

En 1990, P. Hell et J. Nesetril [53] ont publié un document établissant que le problème de H-coloration est solvable en temps polynomial si le graphe H est bipartie, et que ce même problème est NP-complet dans tous les autres cas. Il s'agit là d'une classification complète sur le problème de H-coloration en fonction du graphe H. Dans le cas de classification complète, on parlera de dichotomie.

6.1.2 Ensemble stable

Pour commencer, nous allons donner un exemple de classification en rappelant un certain nombre de résultats sur le problème d'Ensemble stable. Le problème consiste à trouver un ensemble stable de taille maximum dans le graphe donné.

Comme nous allons le voir, ce problème, qui est un des problèmes NP-complets les plus connus et les plus étudiés, peut très vite, et selon la classe de graphe considérée, devenir un problème solvable en temps polynomial.

6.1.3 Les cas polynomiaux

Nous allons commencer par nous intéresser aux classes de graphe pour lesquelles le problème devient solvable en temps polynomial.

Les graphes sans $S_{1,1,2}$

Il existe dans la littérature beaucoup de documents traitant de la complexité d'un problème donné sur une classe de graphe particulière. Pour le graphe sans $S_{1,1,2}$, nous utilisons la notation introduite par V.V. Lozin et R. Mosca [71] afin de désigner un graphe $S_{i,j,k}$ comme étant constitué d'un sommet central qui serait le point de départ de trois chemins de longueurs respectivement i, j et k.

Le graphe $S_{1,1,2}$ est souvent désigné par le terme *fork* dans les documents en anglais.

Nous allons particulièrement nous intéresser à un document de V.E. Alekseev paru en 2002 [1]. Dans ce document, il démontre que le problème consistant à trouver un ensemble stable de taille maximum dans un graphe sans $S_{1,1,2}$ peut être résolu en temps polynomial.

Comme nous avons pu le voir, le fait que le problème d'ensemble stable soit solvable en temps polynomial sur les graphes sans $S_{1,1,2}$ signifie qu'il est aussi solvable en temps polynomial sur toutes les familles de graphes sans H, telles que le graphe H soit un sous-graphe induit du graphe $S_{1,1,2}$.

Nous pouvons notamment retrouver une démonstration à propos de l'un de ces sous-graphes induits du graphe $S_{1,1,2}$ dans un document de G.J. Minty en 1980 [75] qui traite du problème d'ensemble stable sur les graphes sans griffes.

Dans ce document, Minty fournit un algorithme polynomial pour trouver le plus grand ensemble stable dans un graphe sans griffes d'ordre 3. Ce que Minty appelle une griffe (*claw* en anglais) d'ordre 3 est un graphe de quatre sommets, l'un étant de degré 3 et les trois autres de degré 1, c'est-à-dire un graphe bipartie complet avec trois sommets d'un coté et un seul de l'autre. Nous appellerons ce graphe $K_{1,3}$. Clairement, le graphe $K_{1,3}$ est un sous-graphe induit du graphe $S_{1,1,2}$.

Dans son document, Minty précise aussi que le problème devient NP-complet dans le cas d'un graphe sans $K_{1,4}$, mais nous y reviendrons plus tard.

Notons que parmi les sous-graphes induits possibles du graphe $S_{1,1,2}$, il y a aussi le graphe P_4, c'est-à-dire un chemin de quatre sommets. Les graphes sans P_4 font donc partie des classes de graphe pour lesquelles le problème d'ensemble stable peut être résolu en temps polynomial.

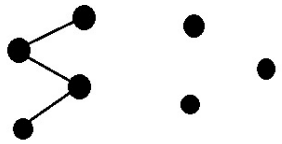
Les graphes sans $P_4 + sP_1$

Grâce à la partie précédente, nous savons maintenant que le problème d'ensemble stable est solvable en temps polynomial sur les graphes sans $S_{1,1,2}$,

mais aussi sur tous les graphes sans H tels que H est un sous-graphe induit du graphe $S_{1,1,2}$.

Nous allons ici nous intéresser plus particulièrement aux graphes sans P_4. La famille des graphes sans P_4, que l'on peut aussi appeler les co-graphes, est une famille qui englobe les graphes sans $S_{1,1,2}$, puisque le P_4 est un sous-graphe induit du graphe $S_{1,1,2}$. Le problème d'ensemble stable peut donc être résolu en temps polynomial sur les graphes sans P_4.

Nous allons voir cependant que nous pouvons étendre cette classe de graphe tout en restant polynomial.



$P_4 + 3P_1$

Soit s un entier. Cet entier peut être quelconque, mais sa valeur est fixée, faisant en fait partie intégrante de la définition des graphes qui vont suivre.

Le graphe $P_4 + sP_1$ est un graphe constitué d'un chemin de longueur 4, associé à une collection de s sommets isolés. C'est-à-dire une union disjointe entre un P_4 et un ensemble stable de taille s. La famille des graphes sans $P_4 + sP_1$ est donc une famille de graphes dans laquelle on peut trouver deux types de graphes possibles :

- Le graphe G dispose d'un P_4 comme sous-graphe induit. Dans ce cas, comme nous savons qu'il n'y a pas de $P_4 + sP_1$, cela signifie qu'il n'y a pas de sP_1 dans le graphe $G \backslash P_4$. Or, un graphe sP_1 est un ensemble stable de taille s, ce qui signifie que nous pouvons borner le nombre de sommets qui font partie d'un ensemble stable dans le graphe G. Dans un P_4, il y aura tout au plus deux sommets qui pourront faire partie d'un ensemble stable. Il nous suffit alors d'énumérer tous les ensembles de sommets de taille $s + 2$ et de vérifier s'ils sont stables pour trouver l'ensemble stable de taille maximum. Cette opération peut bien entendu s'effectuer en temps $O(n^{s+2})$, et reste donc polynomiale.
- Il n'y a dans le graphe G aucun P_4 induit. Dans ce cas, ce graphe est aussi inclus dans la famille des graphes sans $S_{1,1,2}$, pour laquelle nous avons déjà vu que la résolution du problème Ensemble stable est polynomiale.

Nous pouvons en conclure que sur les graphes sans $P_4 + sP_1$, le problème d'ensemble stable peut être résolu en temps polynomial. Cette méthode per-

mettant d'ajouter un nombre fini de sommets isolés peut être utilisée pour n'importe quel graphe H qui définirait une famille de graphes sur laquelle nous aurions déjà démontré que le problème d'ensemble stable est solvable en temps polynomial. Ainsi, nous pourrions de la même manière montrer que le problème reste polynomial sur la classe des graphes sans $S_{1,1,2} + sP_1$.

Remarquez que si le P_4 est un sous-graphe induit du graphe $S_{1,1,2}$, et que le graphe $P_4 + sP_1$ est un sous-graphe induit de $S_{1,1,2} + sP_1$, les graphes $S_{1,1,2}$ et $P_4 + sP_1$ ne sont pas comparables.

Les graphes sans $K_{1,3} + P_2$

Une autre publication importante, si l'on cherche à établir la complexité du problème d'ENSEMBLE STABLE selon les classes de graphe, est le document publié par V.V. Lozin et R. Mosca en 2005 sur le problème d'ENSEMBLE STABLE sur les graphes sans $2P_2$ [71].

Ils démontrent notamment dans ce document que le problème d'ENSEMBLE STABLE peut être résolu en temps polynomial si le graphe ne possède pas de de sous-graphe induit $K_{1,3} + P_2$.

Cela implique, une nouvelle fois, que pour tous graphes H tels que H est un sous-graphe induit de $K_{1,2} + P_2$, alors le problème d'ensemble stable peut être résolu en temps polynomial sur la classe des graphes sans H.

Lozin et Mosca démontrent aussi que le problème est polynomial sur les graphes sans $2P_2$, et nous pouvons retrouver un résultat similaire dans le document de E. Balas et C.S. Yu [4] qui démontrent que trouver une clique de taille maximum dans un graphe sans $\overline{sP_2}$ peut être résolu en temps polynomial. Le problème de CLIQUE et LE PROBLÈME D'ENSEMBLE STABLE étant dual, nous pouvons utiliser ce résultat pour en conclure que trouver un ensemble stable dans un graphe sans sP_2 est solvable en temps polynomial.

Remarquez que si le graphe $2P_2$ est un sous-graphe induit du graphe $K_{1,3} + P_2$, et peut donc lui être comparé, ce n'est pas le cas du graphe sP_2 lorsque $s \geq 3$.

6.1.4 Les cas NP-complets

Nous allons maintenant parler d'un certain nombre de classes de graphe dont nous savons que les propriétés ne suffisent pas à rendre le problème d'ensemble stable solvable en temps polynomial.

Les graphes sans $K_{1,4}$

Revenons un instant sur le document de G.J. Minty [75] dans lequel il nous démontre que le problème d'ensemble stable est solvable en temps polynomial sur les graphes sans griffes. Dans ce même document, Minty montre que son résultat est, d'après sa propre formulation : "the best possible in a sense", c'est-à-dire le mieux que l'on puisse espérer. Il montre alors que sur des graphes qui auraient comme sous-graphes interdits ce qu'il appelle des griffes d'ordre 4, le problème d'Ensemble stable reste NP-complet. Il s'appuie, pour affirmer cela, sur une réduction au problème d'affectation tri-dimensionnelle, Problème NP-complet d'après des travaux de E.L. Lawler [67].

Les graphes sans cycles

Prenons un graphe G quelconque. Le problème consistant à chercher sur le graphe G un ensemble stable S est NP-complet. Supposons que l'on construise un graphe G' en remplaçant chaque arête uv dans le graphe G par un chemin de longueur 4 $uxyv$. Nous cherchons sur le graphe G' un ensemble stable S'.

Nous avons alors plusieurs possibilités :

- $u \in S \wedge v \notin S$ Si l'un des deux sommets u fait partie de l'ensemble stable S, alors on peut dans le graphe transformé G' prendre le sommet u dans l'ensemble stable S'. Pour avoir un ensemble stable de taille maximum, il faut alors prendre le sommet y dans l'ensemble S'.
- $u \notin S \wedge v \notin S$ Si aucun des deux sommets ne fait partie de l'ensemble stable, cela signifie qu'ils ont chacun au moins un voisin qui fait partie de l'ensemble stable. Les arêtes correspondantes se comportant comme dans la règle précédente, les sommets u et v ne peuvent pas être pris dans l'ensemble stable S', par contre, on peut prendre l'un des deux sommets x ou y dans l'ensemble stable S'.

Dans tous les cas, un seul sommet par arête est ajouté à l'ensemble S.

Supposons maintenant que nous trouvions un ensemble stable S' de taille maximum sur le graphe G', et qu'il y ait, dans cet ensemble stable, un des chemins ajoutés $uxyv$ tel que ni le sommet x ni le sommet y ne font partie de l'ensemble stable S'.

Dans ce cas, cela signifie que les deux sommets u et v font tous les deux partie de l'ensemble stable. Si l'on décide alors de prendre l'ensemble $S' \backslash \{u\} \cup \{x\}$, on obtient un ensemble stable de même taille. Nous pouvons avec cet argument d'échange affirmer que pour tout ensemble stable S' dans le graphe G', il existe un ensemble stable de même taille qui utiliserait un sommet par

chemin ajouté.

Cela signifie que si nous avions un algorithme pour trouver l'ensemble stable S' sur le graphe G', alors on pourrait dans le même temps trouver l'ensemble S sur le graphe G. Le problème d'ensemble stable est donc NP-complet sur le graphe G'.

Or si l'on regarde bien, pendant la construction de G', toutes les arêtes on été remplacées par des P_4, les cycles induits les plus petits dans le graphe G' sont des C_9.

Nous pouvons en conclure que le problème d'ensemble dominant est aussi NP-complet sur les graphes sans C_t, pour $t \leq 8$.

Si l'on décide de remplacer les arêtes du graphes, non plus par des P_4, mais par des P_{4+i} pour $i \in 2\mathbb{N}$, on obtiendra de la même manière que si l'on trouve un ensemble stable S de taille k dans le graphe G, il correspondra à un ensemble stable S' de taille $k + E(G) \cdot i$ dans le graphe G'.

Nous pouvons de cette manière augmenter de façon arbitraire la taille du plus petit cycle présent dans le graphe, et le problème reste NP-complet.

Finalement, le problème d'ensemble stable est NP-complet pour tous les graphes sans C_t, avec $t \geq 3$.

6.1.5 Des cas ouverts

La relation d'ordre que nous avons sur les classes de graphe n'est qu'une relation d'ordre partielle, cela entraine le fait que certaines classes de graphe particulières ne peuvent pas forcément être comparées à celles pour lesquelles nous savons déterminer si un problème est solvable en temps polynomial ou s'il est NP-complet.

Les graphes sans P_5

La question des graphes sans P_5 reste un grand problème ouvert dans la dichotomie du problème d'ensemble stable. Récemment, en 2010, B. Randerath et I. Schiermeyer on donné un algorithme sous-exponentiel permettant de trouver un ensemble stable dans un graphe sans P_5 [86], mais malheureusement, ce résultat ne permet pas de conclure sur la complexité de ce cas.

6.1.6 Récapitulatif

Finalement, en ce qui concerne le problème d'ensemble stable, nous pouvons résumer les résultats connus dans le tableau qui suit :

Ensemble stable	
Polynomial	NP-complet
$S_{1,1,2} + sP_1$ sP_2 $K_{1,3} + P_2$	$K_{1,4}$ C_t, $t \geq 3$

Dans ce tableau, nous indiquons dans la colonne *Polynomial* une liste de graphes H maximaux selon la relation de sous-graphes induits définie entre les graphes, et tels que le problème d'ensembles stables est solvable en temps polynomial sur les graphes sans H. Dans la colonne *NP-complet*, nous indiquons une liste de graphes H minimaux selon la relation de sous-graphes induits définie entre les graphes, et tels que le problème d'ensembles stables sur les graphes sans H est NP-complet. Pour tous les problèmes que nous étudierons dans ce chapitre, nous donnerons un tableau similaire.

Bien entendu, cette classification est incomplète, mais elle donne déjà une bonne idée du comportement du problème d'Ensemble stable selon les classes de graphe.

6.2 Domination

Nous allons maintenant nous intéresser au problème de l'Ensemble dominant. Dans un premier temps, nous parlerons d'ensemble dominant au sens classique, c'est-a-dire que dans un graphe $G = (V, E)$ nous recherchons un ensemble $D \subseteq V$ tel que $\forall x \in V, x \in D \vee (\exists y \in D \wedge xy \in E)$. Ce problème, que nous avons déjà étudié à plusieurs reprises au cours de cette thèse, est bien connu pour être NP-complet dans le cas général. Nous citerons une nouvelle fois les travaux de J. Van Rooij [93] qui dans sa thèse donne de nombreuses explications sur le sujet. Les résultats de J. Van Rooij ayant été améliorés que très récemment par Y. Iwata [55]. Voyons maintenant comment ce problème se comporte sur les différentes classes de graphe.

6.2.1 Les cas polynomiaux

Comme nous l'avons fait pour Ensemble stable, nous allons commencer par traiter le cas des classes de graphe pour lesquelles le problème de domination devient polynomial.

Les graphes sans P_4

Un graphe sans P_4 est un graphe pour lequel il n'y a pas de sous-graphe induit qui soit un chemin de quatre sommets. On sait aussi qu'un graphe est

sans P_4 si et seulement s'il s'agit d'un co-graphe.

Comme nous avons pu le voir dans un autre chapitre, les co-graphes sont des graphes qui peuvent être partitionnés en deux sous-graphes tels qu'il y a entre ces deux sous-graphes, soit une jonction, soit une union disjointe, et ces deux sous-graphes sont eux-même des co-graphes.

Pour chaque co-graphe G, il existe une (ou plusieurs) représentation(s) en arbre qui indique(nt) la succession des opérations de type jonction ou union disjointe à faire depuis un groupe de sommets isolés afin d'obtenir le co-graphe G.

En 1984, D.G. Corneil et Y. Perl [19] nous montre comment utiliser cette représentation en co-tree pour obtenir un algorithme polynomial qui résout le problème de domination sur les co-graphes.

Cela nous donne bien sûr un algorithme pour résoudre le problème de domination en temps polynomial sur les graphes sans P_4.

Les graphes sans $P_4 + sP_1$

Le problème de domination est donc polynomial sur les co-graphes, mais nous allons vous montrer que ce résultat peut être légèrement étendu.

Considérons la classe des graphe sans $P_4 + sP_1$. Un graphe G appartenant à cette classe de graphe peut se comporter de l'une des deux manières suivantes :

- Il n'y a pas de P_4 comme sous-graphe induit du graphe G. Dans ce cas, le graphe G est un co-graphe, et nous avons vu juste avant que le problème de domination est polynomial dans ce cas là.
- Il y a un P_4 comme sous-graphe induit dans le graphe G. Dans ce cas, nous savons qu'il y a moins de s sommets isolés qui ne fassent pas partie du voisinage du P_4.
  Le P_4 domine son voisinage, et il suffit de prendre chaque sommet isolé pour se dominer lui-même afin d'obtenir un ensemble dominant. Nous sommes donc capable de borner le nombre de sommets dans un ensemble dominant. En énumérant tous les sous-ensembles de $s + 4$ sommets, nous trouverons l'ensemble dominant minimum, et cette énumération peut être faite en $O(n^{s+4})$, ce qui est polynomial.
  Bien entendu, cela n'est valable que parce que s représente un entier fixé.

Dans tous les cas, si l'on a un graphe sans $P_4 + sP_1$ induit, alors on peut trouver un algorithme polynomial pour résoudre le problème de domination.

6.2.2 Les cas NP-complets

Nous allons maintenant nous intéresser au cas des classes de graphe sur lesquelles le problème de domination reste NP-complet.

Les graphes sans $2P_2, C_4, C_5$

Nous allons nous intéresser à une classe de graphe tel qu'il n'y a pas, dans cette classe, de sous-graphe induit tel que le sous-graphe induit est un $2P_2$, le sous-graphe induit est un C_4, ou le sous-graphe induit est un C_5. Cette classe de graphe particulière est tout simplement la classe des graphes *split*.

Les graphes *split* sont une classe de graphe très connue et très étudiée, et nous savons que le problème de domination est NP-complet sur cette classe de graphe. On peut retrouver dans le document de D.G. Corneil et Y. Perl [19] une démonstration de la NP-complétude du problème de domination sur les graphes *split*, ainsi que dans un document de A.A. Bertossi [7].

Le fait que la classe des graphes *split* soit NP-complet nous donne beaucoup de renseignement pour les graphes sans H. En effet, la classe des graphes *split* est inclue dans la classe des graphes sans $2P_2$, et s'il existait un algorithme polynomial pour résoudre le problème de domination sur les graphes sans $2P_2$, alors il pourrait aussi résoudre le problème de domination en temps polynomial sur les graphes *split*.

Nous pouvons en conclure que le problème de domination est NP-complet sur les graphes sans $2P_2$.

Exactement avec le même raisonnement, nous pouvons montrer que le problème de domination est NP-complet sur les graphes sans C_4, ou sur les graphes sans C_5.

Le graphe sans $2P_2$ est un sous graphe induit pour tout cycle de longueur supérieure ou égale à 6. On peut donc affirmer que pour tout H tel que H est un cycle de longueur supérieure à 4, alors le problème de domination est NP-complet sur le graphe sans H.

Enfin, le graphe sans $2P_2$ est aussi un sous-graphe induit pour tout chemin de longueur supérieure ou égale à 5. Ainsi, le problème de domination est NP-complet pour les graphes sans P_5, et pour tous les graphes sans P_k avec $k \geq 5$.

Les graphes sans C_k, pour $k = 2t + 1$, et $t \in \mathbb{N}$

Dans les documents de D.G. Corneil et Y. Perl [19] et de A.A. Bertossi [7], on trouve aussi des démonstrations sur la NP-complétude du problème de domination sur les graphes biparties.

La NP-complétude des graphes *split* nous a déjà permis d'établir que le problème de domination est NP-complet pour les cycles de taille supérieure à 4. Le fait que le problème soit NP-complet sur les graphes biparties nous permet de compléter cela car le problème est alors aussi NP-complet sur les graphes sans C_3.

Nous pouvons alors affirmer que le problème de domination est NP-complet pour toutes les classes de graphe sans cycles.

Les graphes sans $K_{1,3}$

Intéressons nous maintenant à la classe des graphes sans $K_{1,3}$. Dans ce but, nous allons nous intéresser à un type de graphe particulier : les *line graph*. Le *line graph* $L(G)$ d'un graphe $G = (V, E)$ est un graphe obtenu en représentant chaque arête du graphe G par un sommet, et en les reliant si les arêtes correspondantes étaient adjacentes dans G. En clair, $L(G) = (E(G), \{uv | u \in E(G), v \in E(G), u \text{ et } v \text{ adjacents dans } G\})$.

Les *line graphs* nous intéressent notamment parce qu'ils ont la propriété d'être des graphes sans griffes, c'est-à-dire des graphes sans $K_{1,3}$.

En 1980, M. Yannakakis et F. Gavril [101] montrent dans leur document que le problème de l'ensemble dominant minimum, ainsi que le problème d'ensemble stable dominant, sont NP-complets sur les *line graphs*.

Comme tous les *line graphs* sont des graphes sans $K_{1,3}$, ce résultat nous donne directement que le problème d'ensemble dominant est NP-complet sur les graphes sans $K_{1,3}$.

6.2.3 Une dichotomie

Nous allons utiliser le tableau qui suit pour résumer quels sont les graphes H les plus grands tels que le problème de domination sur les graphes sans H est polynomial, et les graphes H' les plus petits tels que le problème de domination sur les graphes sans H' reste NP-complet.

Ensemble dominant minimum	
Polynomial	NP-complet
$P_4 + sP_1$	$2K_2$
	C_3
	C_4
	C_5
	$K_{1,3}$

Ce que nous allons montrer maintenant, c'est que cette classification sur les classes de graphe pour lesquelles le problème de domination minimum est

soit solvable en temps polynomial, soit NP-complet, est complète. C'est-à-dire que pour tout graphe H, nous pouvons utiliser notre tableau de façon à savoir si le problème de domination est solvable en temps polynomial, ou s'il est NP-complet, sur la classe des graphes sans H. Nous parlerons alors de dichotomie sur les classes de graphe.

Ce que notre tableau nous permet de savoir pour l'instant, c'est que pour un graphe H, si H est le sous-graphe induit d'un graphe H' tel que le problème de domination est solvable en temps polynomial sur les graphes sans H', alors le problème de domination est solvable en temps polynomial sur les graphes sans H.

Inversement, si le graphe H a comme sous-graphe induit un graphe H' tel que le problème d'ensemble dominant est NP-complet sur les graphes sans H', alors le problème d'ensemble dominant est NP-complet sur les graphes sans H.

Lemme 4. *Pour tout graphe H, soit H est le sous-graphe induit d'un $P_4 + sP_1$, soit il a comme sous-graphe induit un graphe $2P_2$, un graphe $K_{1,3}$ ou un cycle.*

Démonstration. Supposons que nous ayons un graphe H tel que ce graphe ne dispose pas comme sous -graphe induit de $2P_2$, de $K_{1,3}$ ni de cycle.

Pour commencer, si le graphe H ne dispose pas de cycles comme sous-graphe induit, alors H est une forêt.

Le graphe H ne dispose pas non plus de $K_{1,3}$ comme sous-graphe induit, cela signifie que tous les arbres de la forêt sont des chemins. Nous avons donc une forêt linéaire.

Enfin, le graphe H n'a pas de $2P_2$ comme sous-graphe induit. Cette propriété nous apprend deux choses, premièrement, que seul un chemin peut avoir une longueur supérieure à 1, les autres ne sont en fait que des sommets isolés. Secondement, la taille de l'unique chemin ne peut pas dépasser 4, car un chemin de taille 5 a un $2P_2$ comme sous-graphe induit.

Le graphe H est donc composé d'un chemin de taille inférieure ou égale à 4, et d'un nuage de sommets isolés. C'est-à-dire $H = P_t + sP_1$ avec $t \leq 4$. On voit facilement que tous les graphes possibles sont des sous-graphes induits de $P_4 + sP_1$. □

Nous pouvons en conclure que, quelque soit le graphe H, alors soit on a H est un sous-graphe induit de $P_4 + sP_1$ et dans ce cas, le problème d'ensemble dominant est solvable en temps polynomial sur les graphes sans H, soit on a que H a comme sous graphe induit, un cycle, un $2P_2$ ou un $K_{1,3}$, et dans ce cas, le problème de l'ensemble dominant sur les graphes sans H est NP-complet. En découle le théorème suivant :

Théorème 18. *Soit H un graphe. Si, pour un entier s fixé, le graphe H est un sous-graphe induit du graphe $P_4 + sP_1$, alors le problème d'ensemble dominant peut être résolu en temps polynomial sur la classe des graphes sans H. Si non, alors il y a un $2P_2$, $K_{1,3}$ ou un cycle comme sous-graphe induit du graphe H et le problème d'ensemble stable est NP-complet sur la classe des graphes sans H.*

6.3 Domination stable

Nous allons maintenant essayer de faire un travail similaire avec un problème légèrement différent. Le problème consiste désormais à trouver un ensemble stable et dominant de taille minimum. Ce problème est bien évidement NP-complet dans le cas d'un graphe quelconque.

6.3.1 Les cas polynomiaux

Intéressons nous donc aux classes de graphe pour lesquelles le problème d'Ensemble stable dominant peut se résoudre en temps polynomial.

Les graphes sans $P_4 + sP_1$

Nous avons pu voir dans la partie sur le problème d'ensemble dominant, que D.G. Corneil et Y. Perl [19] avaient montré comment utiliser la représentation en $co-tree$ afin d'obtenir un algorithme polynomial pour résoudre le problème de domination. Il n'est pas difficile d'adapter ce même algorithme pour que l'ensemble recherché soit un ensemble à la fois dominant et stable.

Le problème d'ensemble stable dominant est polynomial sur les graphes sans P_4, mais si l'on a un graphe avec un P_4 induit, et que l'on sait que le graphe est sans $P_4 + sP_1$, alors, avec la même technique que précédemment, nous pouvons borner le nombre de sommets dans notre ensemble stable dominant à $s+2$ sommets. Il ne reste qu'à énumérer le nombre de sous-ensembles de taille $s+2$, ce qui se fait en $O(n^{s+2})$.

Bien entendu, une nouvelle fois cela n'a de sens que parce que le nombre s est un nombre fixé.

Les graphes sans $P_2 + P_3$

Dans le document de V.V. Lozin et R. Mosca [71], les auteurs ont, entre autre chose, démontré que le problème qu'ils appellent Ensemble stable dominant de poid minimum peut être résolu en temps polynomial sur les

graphes sans $P_2 + K_{1,2}$, que nous préférerons appeler $P_2 + p_3$. La même technique que précédemment nous permet de montrer que le problème ensemble stable dominant est donc solvable en temps polynomial pour les graphes sans $P_3 + P_2 + sP_1$.

6.3.2 Les cas NP-complets

Voyons maintenant les classes de graphe pour lesquelles le problème de domination stable reste NP-complet.

Les graphes sans $2P_3$

Nous pouvons à nouveau nous référer au papier de V.V. Lozin et R. Mosca [71], qui, pour motiver l'intérêt d'avoir montré que le problème d'ENSEMBLE STABLE DOMINANT est solvable en temps polynomial sur les graphes sans $P_2 + P_3$, rappellent que le problème est NP-complet sur les graphes sans $2P_3$. Ce résultat vient en fait d'un rapport technique écrit par I.E. Zverovich en 2003, mais il est possible de le redémontrer par une simple réduction au problème SAT.

Soit F une formule logique, en forme normale conjonctive, c'est-à-dire une conjonction de clauses composées elles-mêmes d'une disjonction de littéraux. Nous pouvons construire un graphe G de la façon qui suit :

- Chaque clause de F est représentée par un sommet, et tous les sommets représentant une clause sont assemblés en une clique.
- Chaque littéral x est représenté par un triangle de trois sommets :
  - Les deux premiers sommets permettent de représenter le littéral x et son inverse $\overline{x}$. Comme ils sont reliés par une arête et que l'on cherche un ensemble stable, on a l'assurance que l'on ne pourra pas prendre à la fois x et $\overline{x}$. On relie alors le sommet x à tous les sommets représentant des clauses dans lesquelles le littéral apparaît, et le sommet $\overline{x}$ pour toutes les clauses où apparaît son inverse.
  - Le troisième sommet du triangle n'est lié à aucun autre sommet, et sert uniquement à s'assurer que l'on ne puisse pas avoir d'ensemble stable dominant plus petit que le nombre de littéraux en choisissant des sommets dans la clique.

Ce graphe est sans $2P_3$ car pour chaque P_3 induit possible, l'un des sommets appartient forcément à la clique.

Avec cette réduction, on peut voir que la formule F est satisfiable si, et seulement si, il existe dans le graphe G un ensemble stable dominant dont la taille est égale au nombre de littéraux de la formule.

Le problème d'ensemble stable dominant est donc NP-complet sur les graphes sans $2P_3$.

Les graphes sans cycles

Pour commencer, nous savons grâce à D.G. Corneil et Y. Perl [19] que le problème de domination stable est NP-complet sur les graphes biparties. Un graphe bipartie est un graphe dans lequel il n'existe aucun cycle de longueur impaire, nous pouvons donc affirmer que, pour tout $t = 2k + 1$ et $k \in \mathbb{N}$, alors le problème de domination stable est NP-complet sur les graphes sans C_t.

Pour traiter les graphes sans cycles pairs, nous allons devoir recourir au même type de construction que nous avons utilisé dans la partie sur le problème Ensemble stable.

Remplaçons chaque arête uv par un chemin de longueur 5 $uxyzv$. Le graphe G d'origine est alors transformé en un graphe G'.

Supposons que l'on ait un ensemble stable dominant D sur le graphe G, nous pouvons le faire correspondre à un ensemble stable dominant D' sur le graphe G' de la façon qui suit :

- $u \in D \wedge v \notin D$ Si le sommet u sert à dominer le sommet v, alors, une fois le graphe transformé, le sommet u domine le sommet x, mais il est nécessaire de dominer les sommets y, z et v. Dans ce cas, il faut prendre le sommet z dans l'ensemble D'.
- $u \notin D \wedge v \notin D$ Si aucun des deux sommets ne se trouve dans l'ensemble stable dominant, cela signifie que les sommets u et v seront dominés depuis une autre arête. Il faut et il suffit alors de dominer les sommets x, y et z. Dans ce cas, on prend le sommet y dans l'ensemble dominant stable D'.

Dans les deux cas, on ajoute exactement un sommet dans l'ensemble D' par arête de G.

Reste à montrer que, pour tout ensemble stable dominant minimum que l'on trouvera dans le graphe G', on pourra lui faire correspondre un ensemble stable dominant dans le graphe G.

La seule possibilité pour que l'une des arêtes transformées $uxyzv$ se retrouve avec plus d'un sommet dans l'ensemble stable dominant D' est que $x \in D' \wedge z \in D'$.

Supposons que l'on ait une arête dans ce cas. Nous pouvons alors décider d'échanger dans l'ensemble dominant D' les sommets u et x. Cela peut avoir trois impacts dans les arêtes adjacentes $u'x'y'z'v'$ pour lesquelles, alors que le sommet v' n'était pas dans l'ensemble D', on décide de l'y ajouter.

– Si cette arête ne dominait pas v', et donc que $z' \notin D'$, cela n'a aucune influence sur cette arête.
– Si $z' \in D'$ et que $x' \notin D'$, alors on peut échanger z' avec y'. La taille de l'ensemble ne change pas.
– Si $z' \in D'$ et que $x' \in D'$, alors le sommet z' est devenu inutile dans l'ensemble stable dominant, et on peut le supprimer. On a trouvé un ensemble stable dominant de taille inférieure, ce qui est en contradiction avec le fait que D' soit minimum.

Dans tous les cas, on a un argument d'échange qui nous permet d'affirmer que, pour tout ensemble stable dominant D' sur G', nous pouvons faire correspondre un ensemble stable dominant D sur G.

Il y a un ensemble stable dominant de taille k sur G si, et seulement si, il y a un ensemble stable dominant de taille $E(G) + k$ sur le graphe G'.

Cette construction peut être étendue en remplaçant les arêtes par n'importe quel chemin de taille $5 + 3 \cdot i$, avec $i \in \mathbb{N}$.

Dans cette construction, on peut voir que nous avons fait disparaître tous les cycles de taille inférieure à $2 \cdot (5+3 \cdot i) - 2$, et le problème d'ensemble stable dominant reste NP-complet. En faisant augmenter la taille des chemins, nous pouvons faire disparaître des cycles de tailles arbitrairement grandes.

Le problème d'ENSEMBLE STABLE DOMINANT est NP-complet pour tous les graphes sans cycles.

Les graphes sans $K_{1,3}$

Pour ce qui est des graphes sans griffes, R. Faudree et al. nous rappellent dans [34] que quand un graphe est sans $K_{1,3}$, alors on a la propriété que la taille de l'ensemble dominant minimum est également la taille de l'ensemble stable dominant minimum.

Grâce à ce théorème, et au fait que nous savons déjà que le problème d'ENSEMBLE DOMINANT est NP-complet sur les graphes sans griffes, nous pouvons conclure que le problème d'ENSEMBLE STABLE DOMINANT est lui aussi NP-complet.

Des cas encore ouverts

Pour le problème d'ENSEMBLE STABLE DOMINANT, la classification que nous avons établie n'est pas complète. Nous pouvons cependant établir le tableau qui suit afin de montrer quels sont les graphes H les plus grands tels que le problème de domination stable sur les graphes sans H devienne polynomial, et les graphes H' les plus petits tels que le problème de domination stable sur les graphes sans H' reste NP-complet.

Ensemble stable dominant	
Polynomial	NP-complet
$P_4 + sP_1$	$K_{1,3}$
$P_2 + P_3$	$2P_3$
	C_k

La NP-complétude du problème sur les graphes sans $K_{1,3}$ et sans cycles nous permet de savoir que seuls des graphes H de type forêts linéaires peuvent nous permettre d'avoir des graphes sans H pour lesquels le problème d'ENSEMBLE STABLE DOMINANT deviendrait solvable en temps polynomial.

Seulement, nous n'avons pour l'instant aucune idée en ce qui concerne, par exemple, les graphes sans P_5 ou les graphes sans $P_4 + P_2$.

Nous pouvons tout de même écrire le théorème suivant :

Théorème 19. *Soit H un graphe. Si, pour un entier s fixé, le graphe H est un sous-graphe induit d'un graphe $P_4 + sP_1$ ou de $P_2 + P_3$, alors le problème d'ensemble stable dominant peut être résolu en temps polynomial sur les graphes sans H. Si le graphe H a comme sous-graphe induit, un $K_{1,3}$, un $2P_3$ ou encore un cycle, alors le problème de domination stable reste NP-complet sur les graphes sans H.*

6.4 Domination connexe

Dans cette partie, nous allons désormais nous intéresser au problème de domination connexe. Un ensemble dominant connexe dans le graphe G est un sous-ensemble de sommets $D \subseteq V(G)$ tel que le sous-graphe induit par l'ensemble D_c est un graphe connexe et que, pour tout sommet dans le graphe G, soit il appartient à l'ensemble D_c, soit il a au moins un voisin qui en fait partie.

6.4.1 Les cas polynomiaux

L'argument que nous allons utiliser pour traiter les cas polynomiaux est en fait un argument général, indépendant du graphe sur lequel le problème est posé.

Le fait est que, pour tout graphe G, il est possible de borner le nombre de sommets nécessaires pour obtenir un ensemble dominant connexe minimum à partir du nombre de sommets nécessaires pour obtenir un ensemble dominant de taille minimum.

Comme nous l'explique D. Kratsch dans son document [64], la taille d'un ensemble dominant connexe est au plus trois fois supérieure à la taille d'un ensemble dominant.

La preuve est assez rapide. Supposons que l'on ait un ensemble dominant D de taille minimum sur le graphe G. Nous pouvons alors utiliser l'ensemble D pour créer un ensemble dominant connexe (qui ne sera pas forcement minimum).

Pour cela, observez que, puisque l'ensemble D est dominant, alors pour tout sommet de D, les autres sommets de D les plus proches de lui sont à une distance inférieure à 3, c'est-à-dire qu'il y a moins de 3 arêtes sur le plus court chemin entre un sommet de D et les autres sommets de D les plus proches de lui.

Si l'on choisit arbitrairement un sommet de D comme point de départ, on voit qu'il existe alors toujours au moins un sommet de D pour lequel il suffit d'ajouter au plus deux sommets dans l'ensemble D pour que cet ensemble, qui est déjà dominant, devienne connexe.

Les graphes sans $P_4 + sP_1$

Nous savons déjà que le problème consistant à trouver un ensemble dominant dans un graphe sans $P_4 + sP_1$ est polynomial. C'est donc en temps polynomial que l'on peut trouver une borne pour la taille maximum d'un ensemble dominant connexe.

Il nous suffit en fait d'énumérer tous les ensembles de taille $3 \cdot (s + 2)$ pour trouver un ensemble dominant connexe de taille minimum, et cela peut se faire en $O(n^{3\cdot(s+2)})$, ce qui reste polynomial.

6.4.2 Les cas NP-complets

Voyons maintenant les différents cas pour lesquels le problème de domination connexe reste NP-complet.

Les graphes sans $2P_2, C_4, C_5$

Tout comme pour le problème d'ensemble dominant, nous allons ici utiliser le fait que le problème d'ensemble dominant connexe soit NP-complet sur les graphes *split* pour montrer qu'il est NP-complet sur les graphes sans $2P_2$, sans C_4 ou encore sans C_5.

La NP-complétude du problème de domination connexe sur les graphes *split* à été démontrée par R.C. Laskar et J. Pfaff [87] dans un rapport technique.

Le problème de domination connexe est donc NP-complet pour les graphes sans $2P_2$, ainsi que pour les graphes sans C_4 et les graphes sans C_5.

Les graphes sans cycles

Étant donné que le graphe $2P_2$ est un sous-graphe induit pour tout cycle de taille supérieure à 6, ou de tout chemin de taille supérieure à 5, il en découle que le problème de domination est aussi NP-complet sur les graphes sans C_k pour $k \geq 4$ et pour les graphes sans P_t pour $t \geq 5$.

Le seul cycle restant est alors le C_3. Or le problème de domination connexe est NP-complet sur les graphes biparties. Cela a été démontré par J. Pfaff, R. Laskar et S.T. Hedetniemi en 1983 [85]. Les graphes biparties sont bien évidement sans C_3, et cela nous permet de conclure que le problème d'ensemble dominant connexe est NP-complet pour les graphes sans cycles.

Les graphes sans $K_{1,3}$

Cette fois encore, pour montrer la NP-complétude du problème de domination connexe sur les graphes sans $K_{1,3}$, nous allons utiliser les *line graphs*. Pour cela, nous allons en fait montrer qu'en utilisant une succession de réduction le problème d'Ensemble dominant connexe sur un *line graph* peut être réduit à un problème de Couverture connexe d'arête (*connected vertex cover* en anglais) qui est un problème connu pour être NP-complet [46].

Pour commencer, rappelons qu'un *line graph* $L(G)$ est une représentation d'un graphe quelconque G telle que, pour toutes arêtes de G, il y a un sommet dans $L(G)$, et telle que deux sommets de $L(G)$ ne sont adjacents que si les arêtes correspondantes dans G avaient un sommet en commun. Chercher un ensemble dominant connexe sur les sommets du graphe $L(G)$ revient donc à chercher un sous-ensemble d'arêtes dans le graphe G tel qu'elles soient toutes reliées et que toutes les arêtes de G soit couvertes.

Pour qu'une arête couvre une autre arête, il faut et il suffit qu'elles aient toutes deux un sommet en commun. Nous allons montrer maintenant que si l'on trouve un ensemble d'arêtes de taille minimum tel que toutes les arêtes du graphes soient couvertes, alors le sous-graphe induit défini par ce sous-ensemble d'arêtes est un arbre.

Supposons que l'on ait un sous-ensemble d'arêtes S de taille minimum couvrant le graphe G tel que l'on ait un cycle dans le sous-graphe induit défini par S. Soit uv l'une des arêtes du cycle dans S. Clairement, puisque uv fait parie d'un cycle, les deux sommets de u et v sont inclus chacun dans au moins une autre arête de l'ensemble S. Toutes les arêtes couvertes par l'arête uv sont donc déjà couvertes par d'autres arêtes de S. uv n'est pas nécessaire pour la couverture du graphe.

De plus, comme l'arête uv fait partie d'un cycle, alors le fait de supprimer

uv ne rompt pas la connexité de l'ensemble S, l'arête uv n'est pas nécessaire pour cela non plus.

Cela signifie que l'ensemble $S \backslash \{uv\}$ est un ensemble d'arêtes couvrantes de taille inférieure à l'ensemble S, ce qui contredit le fait que S était de taille minimale.

Lemme 5. *Le plus petit ensemble d'arêtes couvrant toutes les autres arêtes dans un graphe est toujours un arbre.*

Finalement, nous cherchons un ensemble de k arêtes permettant de définir un arbre tel que les sommets de cet arbre couvrent toutes les arêtes du graphe G. Or on sait que, dans un arbre, le nombre de sommets est égal au nombre d'arêtes augmenté de 1.

Ce que nous cherchons est donc, en fait, un ensemble connexe de $k+1$ sommets couvrant toutes les arêtes du graphe.

Le problème d'Ensemble de sommets couvrant les arêtes est un problème bien connu, que l'ensemble de sommets soit connexe ou pas. Le problème est, entre autres, référencé dans le livre de M.R. Garey et D.S. Johnson [46] sous le nom de *vertex cover*, ou *connected vertex cover* dans le cas connexe, parmi les problèmes NP-complets connus.

Tout cela nous permet de conclure que le problème de domination connexe reste NP-complet sur les *line graphs*, et donc, sur les graphes sans $K_{1,3}$.

6.4.3 Dichotomie

Nous pouvons à nouveau résumer les résultats dans un tableau qui indique les plus grands graphes H tels que le problème de domination connexe est solvable en temps polynomial sur les graphes sans H, et qui indique les plus petits H' tels que sur les graphes sans H', le problème de domination connexe reste NP-complet.

Ensemble dominant connexe	
Polynomial	NP-complet
$P_4 + sP_1$	$2K_2$ C_3 C_4 C_5 $K_{1,3}$

Grâce à ce tableau, on peut voir immédiatement que la classification est identique à celle que nous avons établie pour le problème d'ensemble

dominant. Nous avons déjà montré avec le lemme 4 que, dans ce cas là, la classification est complète.

Nous pouvons en déduire le théorème suivant :

Théorème 20. *Soit H un graphe. Si, pour un entier s fixé, le graphe H est un sous-graphe induit du graphe $P_4 + sP_1$, alors le problème de domination connexe peut se résoudre en temps polynomial sur la classe des graphes sans H. Si non, alors il y a un $2P_2$, un $K_{1,3}$ ou un cycle comme sous-graphe induit du graphe H et le problème d'ensemble dominant connexe est NP-complet sur la classe des graphes sans H.*

6.5 Domination totale

Cette fois, nous allons nous intéresser au problème de domination totale. Un ensemble dominant total dans le graphe G est un sous-ensemble de sommets $D \subseteq V(G)$ tel que, pour tout sommet $u \in V(G)$ du graphe G, alors il a au moins un voisin $v \in D$ dans l'ensemble dominant.

6.5.1 Les cas polynomiaux

Nous allons utiliser le même type d'arguments que pour le problème de domination connexe. Si l'on regarde bien la définition d'un ensemble dominant total, on voit qu'il s'agit en fait d'un ensemble dominant dont les sommets ne peuvent pas se dominer eux-même.

Supposons que nous connaissions un ensemble dominant D dans le graphe G. Nous avons besoin au plus d'un sommet supplémentaire par sommet de D afin de dominer les sommets de D. De cette manière, nous pouvons trouver un ensemble dominant total dont la taille soit au plus le double de la taille d'un ensemble dominant. On peut retrouver cet argument dans le même document de D. Kratsch [64].

Les graphes sans $P_4 + sP_1$

Une nouvelle fois, comme l'on sait que l'on peut déterminer directement une borne supérieure sur la taille d'un ensemble dominant total sur les graphes sans $P_4 + sP_1$, nous pouvons en déduire que le problème est solvable en temps polynomial sur ce type de graphe. Il suffit d'énumérer, par exemple, tous les ensembles de taille $2 \cdot (s+2)$, et cela peut se faire en temps $O(n^{2\cdot(s+2)})$, ce qui est polynomial.

6.5.2 Les cas NP-complets

Nous allons voir maintenant les cas pour lesquels le problème de domination total est NP-complet.

Les graphes sans $2P_2$, C_4, C_5

Une nouvelle fois, nous utilisons le fait que le problème de domination totale est NP-complet sur les graphes *splits*, et cela a été démontré par R.C. Laskar et J. Pfaff [87].

Le problème étant NP-complet sur les graphes *splits*, cela implique qu'il reste NP-complet sur les graphes sans $2P_2$, ou sur les graphes sans cycles de taille supérieure à 4.

Les graphes sans C_3

L'argument est une fois de plus que le problème est NP-complet sur les graphes biparties. Le graphe bipartie nous permet de conclure que le problème reste NP-complet sur les graphes sans cycles de taille impaire. Le C_3 étant le seul cycle n'ayant pas déjà été classifié dans cette catégorie par la NP-complétude du problème sur les graphes *splits*, nous pouvons conclure que le problème reste NP-complet pour tous les graphes sans cycles.

Les graphes sans $K_{1,3}$

Le fait que le problème de domination totale reste NP-complet sur les *line graphs* a été démontré par A.A. McRae [74] pendant sa thèse.

Comme nous l'avons vu désormais, à plusieurs reprises, Les *line graphs* sont des graphes sans $K_{1,3}$, ce qui nous permet d'affirmer que le problème de domination totale reste NP-complet sur les graphes sans $K_{1,3}$.

6.5.3 Dichotomie

Bien que le problème soit différent, la classification des graphes sans H en fonction de la NP-complétude, ou de la solvabilité en temps polynomial, du problème de domination totale est la même que pour le problème de domination connexe, et surtout, la même que pour le problème de domination classique.

Nous pouvons résumer cette classification dans le tableau suivant :

Ensemble dominant total	
Polynomial	NP-complet
$P_4 + sP_1$	$2K_2$ C_3 C_4 C_5 $K_{1,3}$

Ici aussi, le lemme 4 nous permet d'affirmer que cette classification est complète, et que, quelque soit le graphe H, nous saurons déterminer si, sur un graphe sans H, le problème de domination totale est NP-complet ou solvable en temps polynomial.

Théorème 21. *Soit H un graphe. Si, pour un entier s fixé, H est le sous-graphe induit du graphe $P_4 + sP_1$, alors le problème de domination totale peut être résolu en temps polynomial sur un graphe sans H. Sinon, il y a un $2P_2$, un $K_{1,3}$, ou encore un cycle comme sous-graphe induit du graphe H, et le problème de domination totale reste $NP-complet$ sur les graphes sans H.*

6.6 Clique dominante

Le dernier problème que nous allons étudier ici est le problème de clique dominante. Le problème consiste à trouver dans le graphe G une clique D_C de plus petite taille possible telle que les sommets de cette clique dominent tous les autres sommets du graphe.

6.6.1 Les cas polynomiaux

Comme nous allons le voir, sur certains types de graphes sans H, le problème de clique dominante est solvable en temps polynomial, voire même est trivial dans certains cas.

Les graphes sans P_4

Comme nous le savons déjà, un graphe sans P_4 est un co-graphe. Il s'agit donc d'un graphe qui est constitué soit de l'union disjointe de deux co-graphes plus petits, soit d'une jonction entre deux co-graphes plus petits. Dans les deux cas, trouver une clique dominante, si elle existe, est très facile.

$G = G_1 \uplus G_2$: Si le graphe G est une union disjointe entre deux co-graphes plus petits, alors le graphe G n'est pas un graphe connexe. Dans ce cas, il est évident que jamais il ne sera possible de trouver de clique dominante.

$G = G_1 \bowtie G_2$: Si le graphe G est une jonction entre deux co-graphes plus petits, alors le graphe G est une biclique non-induite. Dans ce cas là, à moins qu'il n'existe un sommet universel, il suffit de prendre un sommet de chaque côté de la biclique pour avoir une arête, et donc, une clique de taille 2 dominante.

Le problème consistant à trouver une clique dominante dans un co-graphe, et donc, dans un graphe sans P_4, est solvable en temps polynomial.

Les graphes sans K_s

Un K_s est une clique de taille s. D'une façon évidente, si l'on sait que le graphe dans lequel on cherche une clique dominante n'a pas de clique de taille s, on peut en conclure que la clique dominante, si elle existe, est de taille inférieure à s.

Prenons comme exemple le C_3. Le C_3 est un cycle de taille 3, mais c'est aussi une clique de taille 3 que l'on pourrait noter K_3. Si l'on sait que le graphe est sans C_3, alors la plus grande clique dans le graphe est une arête. Il ne reste alors qu'à vérifier toutes les arêtes pour savoir si elles sont dominantes ou pas.

Le raisonnement reste le même pour toutes les tailles possibles de clique. Si le graphe est sans K_s, alors en énumérant simplement les ensembles de sommets de taille $s-1$ et en vérifiant si ce sont des cliques dominantes, on pourra résoudre le problème en un temps polynomial.

Les graphes sans *pattes*

Un graphe *patte* (*paw graph* en anglais) est un graphe composé de 4 sommets et de 4 arêtes, que l'on peut décrire comme un triangle avec un sommet pendant.

Les graphes sans *pattes* on été notamment étudiés par S. Olariu [82] qui a pu déterminer que un graphe G est sans pattes si et seulement si le graphe G est un graphe sans triangles, ou que le graphe G est un graphe multiparties complet.

Nous avons donc deux cas distincts :

- Si le graphe G est sans triangles, alors la clique la plus grande est de taille 2. Nous sommes dans le cas d'un graphe sans K_s que nous avons déjà traité juste avant. Nous savons que, dans ce cas, le problème de clique dominante est polynomial.
- Si le graphe est multiparties complet, cela signifie que le graphe peut être partitionné en ensembles $\mathcal{S}_1$ à $\mathcal{S}_t$ tels que, pour tout sommet $x \in \mathcal{S}_i$, $y \in \mathcal{S}_j$, avec $i, j \in [1, t]$, alors $xy \in E(G)$. Il suffit alors de prendre

deux sommets dans deux parties différentes pour que le graphe entier soit dominé. Comme les deux sommets proviennent de deux parties différentes, on a la certitude qu'ils sont reliés par une arête, ce qui forme une clique de taille 2. On peut en conclure que, soit le graphe possède un sommet universel, et donc, une clique dominante de taille 1, soit il a une clique dominante de taille 2.

Dans les deux cas, la solution peut être trouvée en temps polynomial.

Le problème de clique dominante est solvable en temps polynomial sur les graphes sans pattes.

Les graphes sans diamants

Un diamant est un graphe de 4 sommets et de 5 arêtes. On peut le décrire comme deux triangles partageant un côté, ou comme un cycle de taille 4 avec une corde.

Si un cycle est sans diamants, cela signifie que, pour chaque arête uv, les voisins communs de u et de v forment une clique C_{uv}.

Si l'on s'intéresse aux sommets x du graphe G qui ne sont pas dans le voisinage de l'arête uv, alors on peut voir que x a au plus 1 voisin dans la clique C_{uv} car, sinon, deux sommets de la clique associés à x et à l'un des sommets u ou v forment des diamants.

Finalement, pour chaque arête uv, on peut déterminer, si elle existe, la clique dominante minimale par inclusion qui contient uv en vérifiant pour chaque sommet x de G quel est le sommet de C_{uv} qui domine x.

Le fait de constituer la clique dominante minimale par inclusion contenant l'arête uv se fait en $O(n)$, et nous avons besoin de faire ce travail pour toutes les arêtes du graphe en $O(n^2)$ afin de pouvoir comparer les résultats et trouver la clique dominante de taille minimum.

Il existe donc un algorithme polynomial en $O(n^3)$ pour trouver une clique dominante de taille minimum dans un graphe sans diamants.

6.6.2 Les cas NP-complets

L'étendue des classes de graphe sur lesquelles le problème de clique dominante est solvable en temps polynomial est plus étendue que pour les autres problèmes. Il n'en reste pas moins que ce problème reste NP-complet dans certains cas.

Les graphes sans $2P_2$, sans C_4 et sans C_5

En 1987, A. Brandstädt et D. Kratsch [14] ont montré que le problème de clique dominante est NP-complet sur les graphes *splits*.

Comme nous l'avons fait pour les autres problèmes, le fait que le problème de clique dominante soit NP-complet pour les graphes splits nous permet de conclure qu'il reste NP-complet sur les graphes sans $2P_2$, sur les graphes sans C_3 ou sur les graphes sans C_4.

Les graphes sans $K_{1,4}$

Pour la démonstration qui va suivre, nous nous sommes inspiré des documents de D. Kratsch avec L. Stewart [65], et avec A. Brandstadt[14] dans lesquels les auteurs montrent que le problème de clique dominante est NP-complet respectivement sur les graphes de co-comparabilité et les graphes de permutation. Les réductions données dans ces documents donnent comme conséquence directe la NP-complétude du problème de clique dominante sur les graphes sans $K_{1,5}$.

Théorème 22. *Le problème de clique dominante est NP-complet sur les graphes sans $K_{1,4}$.*

Démonstration. Soit F une formule logique en forme normale conjonctive. Nous construisons le graphe G de la façon qui suit :

- Chaque clause j est représentée par un sommet c_j, l'ensemble des sommets représentant une clause forme une clique.
- Chaque littéral i est représenté par une paire de sommets l_i et $\overline{l_i}$, symbolisant respectivement l'assignation du littéral à vrai ou faux. L'ensemble des sommets représentant les littéraux forme une clique ôtée d'un couplage, les deux assignations différentes l_i et $\overline{l_i}$ d'un même littéral n'étant pas reliées par une arête.
- Pour chaque clause, le sommet qui la représente est relié par une arête aux sommets représentant ses littéraux.
- On ajoute un sommet u adjacent à tous les sommets représentant les littéraux, et un autre sommet de degré 1, adjacent à u, qui sera le voisin privé de u.

Le sommet u est le seul à pouvoir dominer son voisin privé. Si une clique dominante existe, alors le sommet u en fait forcément partie, cela nous permet de nous assurer que la clique dominante, si elle existe, n'est composée que de sommets représentant des littéraux.

Supposons qu'une clique dominante existe, elle est composée du sommet u et d'un certain nombre de sommets représentant des littéraux. Comme

il n'y a pas d'arêtes entre un sommet l_i et son contraire $\overline{l_i}$, on sait qu'au plus une assignation possible de chaque littéral est représentée dans la clique dominante. Puisque la clique est dominante, alors on sait que l'ensemble de littéraux représenté par la clique contient au moins un littéral de chaque clause de la formule F. Il suffit alors que les littéraux représentés dans la clique dominante soient assignés à vrai pour que toutes les clauses de la formule F soient satisfaites. Dans l'autre sens, si l'on connait une assignation des littéraux de la formule F qui permet de satisfaire toutes ses clauses, alors il suffit de prendre tous les sommets représentant les littéraux assignés à vrai et d'ajouter le sommet u pour obtenir une clique dominante dans le graphe G.

Soit n le nombre de littéraux dans la formule F, la formule F est satisfiable si, et seulement si, il existe une clique dominante de taille $n+1$ dans le graphe G correspondant.

Analysons maintenant la construction de ce graphe. Le voisin privé de u ne peut évidement pas avoir d'ensemble stable de taille supérieure à 1 dans son voisinage. Le voisinage du sommet u se compose de son voisin privé et de l'ensemble des sommets représentant les littéraux. On peut ainsi trouver dans le voisinage de u un ensemble stable de taille 3 en utilisant son voisin privé, et les deux sommets représentant les assignations contraires d'un même littéral i, c'est-à-dire l_i et $\overline{l_i}$. Cependant, il n'est pas possible de trouver d'ensemble stable de taille 4, car tout autre voisin de u est un sommet représentant un littéral $k \neq i$ et dans ce cas, l_k (et $\overline{l_k}$) est voisin de l_i et $\overline{l_i}$. Si l'on observe le voisinage d'un sommet l_i (ou $\overline{l_i}$) quelconque, l'ensemble stable de plus grande taille que l'on peut trouver se compose des deux assignations contraires d'un même littéral $k \neq i$, et d'un sommet représentant une clause. Il n'est pas possible de trouver d'ensembles stables de taille 4 dans le voisinage de l_i (ou $\overline{l_i}$). Enfin, la même observation peut être faite si l'on regarde dans le voisinage d'un sommet c_j.

Dans tous les cas, il est impossible de trouver un sommet dont le voisinage contienne un ensemble de taille 4. Le graphe G est sans $K_{1,4}$. □

6.6.3 Des cas ouverts

Les résultats que nous connaissons peuvent être résumés dans un tableau regroupant les plus grands graphes H tels que le problème de clique dominante est solvable en temps polynomial sur un graphe sans H, et regroupant les plus petits graphes H' tels que le problème reste NP-complet sur les graphes sans H'.

MDC	
Polynomial	NP-complet
P_4	$2K_2$
K_s	C_4
pattes	C_5
diamants	$K_{1,4}$

Cette classification n'est pas complète, certains cas très intéressants restent encore ouverts. Comme par exemple le cas des graphes sans $K_{1,3}$, pour lesquels nous ne savons dire si le problème reste NP-complet, ou s'il est solvable en temps polynomial. Le fait que le problème soit solvable en temps polynomial pour les graphes sans C_3 pose aussi beaucoup de questions sur toutes formes de petits graphes cordaux.

Il reste finalement beaucoup de travail si l'on souhaite arriver un jour à une dichotomie pour ce problème aussi.

6.7 Un algorithme exact pour les graphes sans $P_2 + P_3$

Dans cette section, nous allons nous rapprocher des autres chapitres de la thèse et donner un algorithme pour trouver un ensemble dominant de taille minimum dans un graphe sans $2P_2$.

Grâce à notre classification, nous pouvons affirmer, juste en regardant notre tableau, que le problème de domination sur les graphes sans $2P_2$ est NP-complet. Ce que nous souhaitons montrer maintenant, c'est que, bien que NP-complet, le problème est plus facile à résoudre sur une classe restreinte (ici la classe sans $P_2 + P_3$) que dans le cas général. Le graphe $2P_2$ est un sous-graphe induit du graphe $P_2 + P_3$, le fait de savoir que le problème de domination est NP-complet sur les graphes sans $2P_2$ nous permet d'affirmer qu'il est NP-complet sur les graphes sans $P_2 + P_3$.

L'algorithme que nous allons vous donner permet de trouver sur les graphes sans $P_2 + P_3$ un ensemble dominant de taille minimum avec un temps d'exécution de $O^*(1.2279^n)$, alors que le meilleur algorithme connu pour le cas général est pour l'instant celui de Y. Iwata [55] avec un temps en $O^*(1.4864^n)$.

6.7.1 Ensemble stable dominant

Dans un premier temps, nous allons chercher un ensemble stable dominant. L'ensemble dominant de taille minimum sur un graphe sans $P_2 + P_3$

n'est pas forcément stable, mais si l'on est capable d'énumérer tous les ensembles stables dominants, alors les solutions qu'il nous restera à chercher par la suite contiendront toutes au moins une arête, et c'est une propriété qui rendra ces ensembles plus facile à trouver.

Nous avons étudié, dans ce chapitre, le problème de domination stable, et si l'on regarde le tableau obtenu, on voit que le problème de domination stable est polynomial sur les graphes sans $P_2 + P_3$.

6.7.2 Ensemble dominant sur graphes sans P_3

Le graphe sur lequel nous travaillons est sans $P_2 + P_3$, mais est-ce parce que l'on n'a pas de P_3 dans le graphe ou parce que l'on a un P_3 et que tous les sommets hors de son voisinage sont stables entre eux ?

Si l'on suppose que le graphe est sans P_3, alors, une nouvelle fois, il nous suffit de regarder l'un de nos tableaux. Le problème de domination sur un graphe sans P_3 est polynomial.

6.7.3 Ensemble dominant non-stable avec P_3

Nous savons maintenant que, dans notre graphe, s'il y a un P_3, alors les sommets hors de son voisinage forment un ensemble stable. Le nombre de P_3 possibles dans le graphe est de n^3.

Choisissons une arête xy dans le graphe. Nous supposons que cette arête fait partie de l'ensemble dominant D.

Cela implique que les sommets x, y et $N[xy]$ sont dominés.

Choisissons dans le voisinage de y un sommet z qui ne fait pas partie du voisinage de x. Si un tel sommet n'existe pas, cela signifie que le sommet y n'a pas de voisins privés, et donc qu'il n'a aucune raison de faire partie de l'ensemble dominant, on peut donc stopper.

Les sommets x, y et z forment un P_3, l'ensemble des sommets en dehors du voisinage de ce P_3 est donc un ensemble stable S.

Distinguons maintenant deux cas :

– Tous les sommets de $N(z)$ sont déjà dominés par x et y. Dans ce cas, les seuls sommets qu'il reste à dominer forment un ensemble stable. Si nous avons des sommets isolés, nous pouvons les éliminer car la seule façon possible de les dominer est qu'ils se dominent eux-mêmes. Les sommets restant à dominer ont donc tous au moins un voisin dans le voisinage de xy. supposons que l'on ait un ensemble dominant D de taille minimum avec un des sommets u de l'ensemble stable S qui fasse partie de D. Dans ce cas, si l'on prend dans D l'un des voisins

de u à la place de u, on se retrouve avec un ensemble de même taille, et cet ensemble est toujours dominant. Nous avons donc un argument d'échange nous permettant d'affirmer qu'il y a un ensemble dominant de taille minimum qui ne contient aucuns sommets de S. Il ne nous reste alors qu'à dominer les sommets de S en utilisant ceux de $N[xy]$. Il s'agit alors d'un problème de RED-BLUE DOMINATING SET. Dans sa thèse, Y. Van Rooij nous donne un algorithme pour RED-BLUE DOMINATING SET qui s'exécute en $O^*(1.2279^n)$ [93].

– Il y a au moins un voisin de z qui n'est pas dominé par xy. Appelons ce voisin z'. Les sommets x, y, z, z' forment un P_4. À ce stade, nous avons aussi besoin de distinguer si le sommet x a un voisin qui ne ferait pas partie du voisinage de y ou z.

  $N(x) \subseteq N(z) \cup N(y)$: Dans ce cas, nous pouvons affirmer que s'il y a un ensemble dominant de taille minimum qui contient x, alors il y a un ensemble dominant de même taille qui contient z et pas x. Nous avons donc un argument d'échange pour dire que si x n'a pas de voisins hors du voisinage de z, alors x n'a pas de raison d'être dans l'ensemble dominant, et on stoppe.

  $N(x) \nsubseteq N(z) \cup N(y)$: Il existe donc un sommet $x' \in N(x)$ tel que $x' \notin N(y) \cup N(z)$. Les sommets x', x, y, z, z' forment un P_5. Or le graphe est sans $P_2 + P_3$. Cela signifie que x', et surtout z', n'ont pas d'autres voisins ! Sinon, le graphe contiendrait un P_6. Or le P_2+P_3 est un sous-graphe induit du P_6. Ce qui nous intéresse particulièrement, c'est que, par conséquence, les voisins de z ne peuvent être utilisés pour dominer les sommets de l'ensemble stable S, puisqu'ils n'y ont pas de voisin.

  Le meilleur moyen de dominer $N(z)$ est alors évidemment de prendre z dans l'ensemble dominant D, et les sommets qu'il reste à dominer sont les sommets de l'ensemble stable S qui, avec le même argument que précédemment, ne peuvent être dominés que par les sommets dans le voisinage de xy.

  Il s'agit là encore d'un problème de RED-BLUE DOMINATING SET, que l'on résoudra en $O^*(1.2279^n)$ grâce à l'algorithme de Y. Van Rooij [93].

Au pire des cas, nous aurons besoin d'essayer pour chaque arête associée à chacun des sommets dans le voisinage de cette arête. Cela signifie que l'on aura à faire n^3 essais pour l'attribution de x, y et z.

Le temps d'exécution complet de cet algorithme sera donc $O(n^3 \cdot 1.2279^n)$, c'est-à-dire $O^*(1.2279^n)$.

6.8 Conclusion

Nous avons étudié la complexité des problèmes de domination, de domination stable, de domination connexe, de domination totale et de clique dominante sur les graphes sans H, en s'efforçant de les classer selon que les problèmes sont solvables en temps polynomial ou NP-complet. Nous avons pu établir une dichotomie pour le problème de domination : sur un graphe sans H, si H est un sous-graphe induit d'un $P_4 + sP_1$, alors le problème de domination est solvable en temps polynomial sur les graphes sans H. Il reste NP-complet sinon. Nous avons pu établir des dichotomies similaires pour les problèmes de domination connexe et de domination stable.

Pour le problème de domination stable, plusieurs cas restent ouverts, les plus importants sont les graphes sans P_5, les graphes sans P_6, les graphes sans $P_4 + P_2$, ou encore les graphes sans $3P_2$. En ce qui concerne le problème de clique dominante, les graphes sans $K_{1,3}$, les graphes sans P_5, ou encore les graphes sans $S_{1,1,2}$, sont les classes qui méritent d'être étudiées.

Conclusion

Dans cette thèse, nous avons étudié une diversité de problèmes difficiles sur les graphes, chaque chapitre portant sur un, voir plusieurs, problèmes différents. Nous avons aussi étudié diverses catégories de problèmes, puisque nos travaux portent aussi bien sur des problèmes d'optimisation ou de décision, et leurs classifications, que sur des problèmes de comptage et d'énumération. Au commencement de cette thèse, sous la direction de Dieter Kratsch, notre but était simplement d'étudier la construction et l'analyse d'algorithmes donnant un résultat exact, en utilisant un temps d'exécution exponentiel au rapport de la taille de l'entrée, pour des problèmes NP-complets sur les graphes, sans plus de précisions sur la nature des problèmes que nous allions considérer. Les problèmes de décision et d'optimisation étant omniprésents dans l'étude des problèmes NP-complets, notre intérêt s'est finalement porté sur une catégorie de problèmes moins répandue, les problèmes de comptage et finalement, les problèmes d'énumération.

Pour construire et analyser nos algorithmes, nous avons essentiellement utilisé la technique de branchement et réduction, l'associant le cas échéant, avec de la programmation dynamique. Les calculs de temps d'exécution ont été effectués en utilisant des mesures basées sur les poids de sommets avec la technique mesurer et conquérir, les poids utilisés pouvant, selon les algorithmes, être très simples ou au contraire très compliqués.

Dans le Chapitre 2 sur le problème d'#Ensemble Stable Bicoloré, nous cherchons à compter le nombre d'ensembles stables sur un graphe dont les sommets sont partitionnés en deux ensembles de façon arbitraire. L'algorithme que nous avons construit pour cela est constitué de deux parties, la première utilisant la technique de branchement et réduction, et la seconde utilisant de la programmation dynamique sur une décomposition arborescente du graphe. Au cours de ce chapitre, nous avons volontairement détaillé deux approches pour l'analyse du temps d'exécution, selon l'utilisation des poids par la méthode mesurer et conquérir ou pas. La raison pour laquelle nous avons ainsi détaillé les deux méthodes était à la fois de montrer l'efficacité de la méthode mesurer et conquérir, mais aussi de pouvoir plus facilement

expliquer son fonctionnement. Finalement, notre algorithme permettant de compter les ensembles stables sur un graphe bicoloré s'exécute avec un temps $O^*(1.2691^n)$.

Dans le Chapitre 3 sur la coloration et l'étiquetage d'un graphe cubique connexe, nous commençons par donner un algorithme permettant d'énumérer les 3-colorations d'arêtes sur un graphe cubique. Cet algorithme est construit grâce à la méthode de branchement et réduction, et l'analyse de son temps d'exécution est faite à l'aide d'arguments sur la structure de l'arbre de recherche décrivant l'exécution de l'algorithme, démontrant que celui-ci ne peut avoir plus de $2^{\frac{5n}{8}}$ feuilles et impliquant que l'algorithme s'exécute au pire des cas en un temps $O^*(2^{\frac{5n}{8}})$. Nous donnerons par la suite dans ce chapitre une borne inférieure sur le nombre de 3-colorations des arêtes d'un graphe cubique connexe qui est d'au moins $2^{\frac{n}{2}}$.

Dans la deuxième partie de ce chapitre, nous présentons un algorithme de branchement et réduction grâce auquel nous pouvons énumérer les étiquetages $L(2,1)$ à six couleurs dans un graphe cubique connexe. L'analyse de cet algorithme se fait à l'aide de la méthode mesurer et conquérir et l'utilisation d'un seul poids ϵ réel. Finalement, cet algorithme s'exécute avec un temps $O^*(1.7790^n)$. Une fois encore, nous donnerons par la suite une borne inférieure sur le nombre d'étiquetages $L(2,1)$ dans un graphe cubique connexe, qui est d'au moins $2^{\frac{2n}{7}}$.

Dans le Chapitre 4 sur les ensembles dominants minimals par inclusion, nous avons cherché à établir des bornes inférieures et supérieures sur le nombre d'ensembles dominants minimals par inclusion dans différentes classes de graphe, les classes de graphe étudiées étant les graphes cordaux, les graphes cobiparties, les graphes splits, les graphes d'intervalles propres, les co-graphes et enfin, les graphes trivialement parfaits.

Les bornes supérieures pour le nombre d'ensembles dominants minimals par inclusion sont démontrées à l'aide d'algorithmes de branchement et réduction, les analyses de temps d'exécution de ces algorithmes étant faites par une analyse simple où la mesure d'une instance est la somme des poids des sommets, les poids des sommets ne pouvant être que de 1 ou 0. Il est intéressant de noter que dans ces problèmes, l'utilisation de la méthode mesurer et conquérir et de poids plus compliqués n'a pas permis d'obtenir d'améliorations significatives sur les temps d'exécution des algorithmes.

Les bornes inférieures sont données grâce à plusieurs familles de graphes pour lesquelles nous montrons à la fois leur appartenance aux classes de graphe étudiées, et l'existence dans ces graphes d'un nombre d'ensembles dominants minimals par inclusion égal à celui que nous définissons comme borne inférieure.

Dans le Chapitre 5 sur les ensembles coupe cycles minimals par inclusion, nous utilisons des démonstrations par induction sur la structure des classes de graphe étudiées pour établir les bornes supérieures sur le nombre d'ensembles coupe cycles minimals par inclusion dans ces graphes. Les classes de graphe que nous avons étudiées sont les graphes cordaux, les co-graphes, et plus succinctement les graphes d'arcs circulaires. Pour ces trois classes de graphes, nous avons pu établir que le nombre d'ensembles coupe cycles minimals par inclusions est au plus $10^{\frac{n}{5}}$.

En ce qui concerne les graphes cordaux et les graphes splits, nous donnons aussi deux familles de graphes permettant de montrer que les bornes inférieures sur le nombre d'ensembles coupe cycles minimals par inclusion dans ces classes de graphe sont ajustées aux bornes supérieures.

Enfin, dans le Chapitre 6, nous avons porté notre attention sur la complexité de différents problèmes d'optimisation fortement liés au problème de domination d'un graphe sur les classes de graphe définies comme étant des graphes pour lesquels un graphe H fixé est un sous-graphe induit interdit. Nous avons pu établir pour les problèmes de domination, de domination connexe et de domination totale, une dichotomie des classes de graphe en fonction du fait que ces problèmes soient solvable en temps polynomial, ou restent NP-complets. Il est intéressant de noter que pour les trois problèmes, la dichotomie est identique. Ainsi, sur un graphe sans H, si H est un sous-graphe induit d'un graphe $P_4 + sP_1$, alors les problèmes de domination, de domination connexe et de domination totale peuvent être résolus en temps polynomial sur les graphes sans H, les trois problèmes restent NP-complets sinon.

En ce qui concerne de futures directions de recherche, bien entendu, toutes les questions posées dans les conclusions intermédiaires de chaque chapitre restent des problèmes intéressants.

D'une façon plus générale, il serait intéressant de pouvoir examiner de façon plus complète la relation entre les différentes catégories de problème que nous avons abordées dans notre thèse, à savoir les problèmes de décision, d'optimisation, de comptage ou d'énumération. Nous sommes déjà capable de dire que la solution pour un problème d'énumération permet d'obtenir facilement la solution pour le problème de comptage, et pour les problèmes d'optimisation et de décision associés. La relation entre le problème de comptage et le problème d'optimisation est un peu plus complexe : compter des ensembles minimals par inclusion ne permet, par exemple, aucune conclusion sur un éventuel minimum.

Il existe des exemples de problèmes pour lesquels la version de décision est solvable en temps polynomial, mais pour lesquels compter le nombre

de solutions peut s'avérer difficile, tel le problème de couplage parfait [92]. Serait-il possible de caractériser les raisons qui font qu'un comptage, ou une énumération, peuvent être difficiles ou pas ?

Il peut être intéressant aussi de remarquer que Y. Okamoto, T. Uno et R. Uehara [81] ont démontré que, sur les graphes cordaux, compter les ensembles stables maximums peut être fait en temps linéaire, mais que compter les ensembles stables maximals par inclusion est un problème $\#P$-complet.

L'étude de la relation entre la complexité d'un problème de comptage et la complexité du problème d'optimisation qui lui est associé, cela supposant que le problème d'optimisation soit NP-complet, pourrait avoir un certain intérêt. Pouvoir répondre à un problème de comptage permet *a priori* de répondre au problème d'optimisation qui lui correspond. Cependant, la complexité du problème de comptage est-elle vraiment plus importante que celle du problème d'optimisation, ou sont-elles finalement relativement proches ? Ici, nous allons estimer la complexité d'un problème seulement par la valeur du temps d'exécution du meilleur algorithme connu permettant de le résoudre.

Contrairement à ce que l'on pourrait imaginer, pour un certain nombre de problèmes fondamentaux sur les problèmes NP-complets de graphe, les temps d'exécution des deux types de problème peuvent être très différents, ou au contraire, très proches.

Pour le problème SAT, ainsi que pour le problème du Nombre Chromatique [11], les meilleurs algorithmes s'exécutent en $O^*(2^n)$, que ce soit pour le comptage ou pour l'optimisation.

En ce qui concerne le problème de domination, le meilleur algorithme pour la version d'optimisation s'exécute en temps $O^*(1.4864^n)$ [55], alors que le meilleur algorithme de comptage se résout avec un temps de $O^*(1.5673^n)$ sur un espace polynomial, et $O^*(1.5002^n)$ sur un espace exponentiel [93]. Nous pouvons aussi remarquer que le problème d'ensemble stable, dans sa version d'optimisation, peut être résolu avec un temps de $O^*(1.2127^n)$ [13], alors que sa version de comptage nécessite un temps d'exécution de $O^*(1.2377)$ (M. Wahlstrom, private communication).

Une autre direction de recherche qui paraît intéressante sera, dans le cadre des algorithmes d'énumération, de chercher des algorithmes dont le temps d'exécution peut être estimé d'après la taille, non plus de l'entrée, mais du résultat attendu. La question de savoir s'il existe ou non un algorithme à délai polynomial, ou simplement polynomial en la taille de la sortie, sur l'énumération des transversaux dans un hypergraphe [32] est à l'heure actuelle un grand problème ouvert sollicitant beaucoup d'intérêt. Le problème d'énumération des Transversaux minimaux dans un Hypergraphe étant fortement lié avec le problème d'énumération des Ensemble

DOMINANT MINIMALS par inclusion, cela ouvre un large panel de problèmes pour lesquels réussir à déterminer si un algorithme polynomial en la taille de la sortie existe pourrait s'avérer être un progrès non négligeable.

Index

Bibliographie

[1] V. Alekseev, *Polynomial algorithm for finding the largest independent sets in graphs without forks*, Discrete Applied Mathematics, 135 (2004), pp. 3 – 16.

[2] G. Alexe, S. Alexe, Y. Crama, S. Foldes, P. L. Hammer, and B. Simeone, *Consensus algorithms for the generation of all maximal bicliques*, Discrete Applied Mathematics, 145 (2004), pp. 11–21.

[3] J. Amilhastre, M.-C. Vilarem, and P. Janssen, *Complexity of minimum biclique cover and minimum biclique decomposition for bipartite domino-free graphs*, Discrete Applied Mathematics, 86 (1998), pp. 125–144.

[4] E. Balas and C. S. Yu, *On graphs with polynomially solvable maximum-weight clique problem*, Networks, 19 (1989), pp. 247–253.

[5] R. Beigel and D. Eppstein, *3-coloring in time $o(1.3289^n)$*, Journal of Algorithms, 54 (2005), pp. 168–204.

[6] R. Bellman, *Dynamic programming and stochastic control processes*, Information and Control, 1 (1958), pp. 228–239.

[7] A. A. Bertossi, *Dominating sets for split and bipartite graphs*, Information Processing Letters, 19 (1984), pp. 37–40.

[8] S. Bessy and F. Havet, *Enumerating the edge-colourings and total colourings of a regular graph*, Journal of Combinatorial Optimization, pp. 1–13.

[9] D. Binkele-Raible, H. Fernau, S. Gaspers, and M. Liedloff, *Exact exponential-time algorithms for finding bicliques*, Information Processing Letters, 111 (2010), pp. 64–67.

[10] A. Björklund and T. Husfeldt, *Inclusion–exclusion algorithms for counting set partitions*, in FOCS, IEEE Computer Society, 2006, pp. 575–582.

[11] A. Björklund, T. Husfeldt, and M. Koivisto, *Set partitioning via inclusion-exclusion*, SIAM Journal on Computing, 39 (2009), pp. 546–563.

[12] J. R. S. Blair and B. W. Peyton, *An introduction to chordal graphs and clique trees*, Graph Theory and Sparse Matrix Computations, 56 (1998), pp. 1–27.

[13] N. Bourgeois, B. Escoffier, V. T. Paschos, and J. M. M. van Rooij, *A bottom-up method and fast algorithms for max independent set*, in SWAT, 2010, pp. 62–73.

[14] A. Brandstädt and D. Kratsch, *On domination problems for permutation and other graphs*, Theoretical Computer Science, 54 (1987), pp. 181–198.

[15] A. Brandstädt, V. Le, and J. Spinrad, *Graph Classes : A Survey*, Siam Monographs on Discrete Mathematics and Applications, Society for Industrial and Applied Mathematics, 1999.

[16] B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle, *Feedback vertex set on graphs of low cliquewidth*, in IWOCA, vol. 5874 of Lecture Notes in Computer Science, Springer, 2009, pp. 113–124.

[17] S. A. Cook, *The complexity of theorem-proving procedures*, in STOC, 1971, pp. 151–158.

[18] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction To Algorithms*, MIT Press, 2001.

[19] D. Corneil and Y. Perl, *Clustering and domination in perfect graphs*, Discrete Applied Mathematics, 9 (1984), pp. 27 – 39.

[20] D. G. Corneil and J. Fonlupt, *The complexity of generalized clique covering*, Discrete Applied Mathematics, 22 (1989), pp. 109–118.

[21] D. G. Corneil, Y. Perl, and L. K. Stewart, *A linear recognition algorithm for cographs*, SIAM Journal on Computing, 14 (1985), pp. 926–934.

[22] J.-F. Couturier, P. A. Golovach, D. Kratsch, and D. Paulusma, *List coloring in the absence of a linear forest*, in WG, 2011, pp. 119–130.

[23] ——, *On the parameterized complexity of coloring graphs in the absence of a linear forest*, Journal of Discrete Algorithms, 15 (2012), pp. 56–62.

[24] J.-F. Couturier, P. Heggernes, P. van 't Hof, and D. Kratsch, *Minimal dominating sets in graph classes : Combinatorial bounds and enumeration*, in SOFSEM, 2012, pp. 202–213.

[25] J.-F. Couturier, P. Heggernes, P. van 't Hof, and Y. Villanger, *Maximum number of minimal feedback vertex sets in chordal graphs and cographs*, in COCOON, vol. 7434 of Lecture Notes in Computer Science, Springer, 2012, pp. 133–144.

[26] J.-F. Couturier and D. Kratsch, *Bicolored independent sets and bicliques*, in CTW, 2011, pp. 130–133.

[27] ——, *Bicolored independent sets and bicliques*, Information Processing Letters, 112 (2012), pp. 329–334.

[28] M. Dawande, P. Keskinocak, J. M. Swaminathan, and S. Tayur, *On bipartite and multipartite clique problems*, Journal of Algorithms, 41 (2001), pp. 388–403.

[29] V. M. F. Dias, C. M. H. de Figueiredo, and J. L. Szwarcfiter, *Generating bicliques of a graph in lexicographic order*, Theoretical Computer Science, 337 (2005), pp. 240–248.

[30] ——, *On the generation of bicliques of a graph*, Discrete Applied Mathematics, 155 (2007), pp. 1826–1832.

[31] R. Diestel, *Graph Theory*, Graduate Texts in Mathematics, Springer, 2006.

[32] T. Eiter and G. Gottlob, *Hypergraph transversal computation and related problems in logic and ai*, in Logics in Artificial Intelligence, S. Flesca, S. Greco, G. Ianni, and N. Leone, eds., vol. 2424 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2002, pp. 549–564.

[33] D. Eppstein, *Small maximal independent sets and faster exact graph coloring*, Journal of Graph Algorithms and Applications, 7 (2003), pp. 131–140.

[34] R. J. Faudree, E. Flandrin, and Z. Ryjácek, *Claw-free graphs - a survey*, Discrete Mathematics, 164 (1997), pp. 87–147.

[35] H. Fernau, S. Gaspers, D. Kratsch, M. Liedloff, and D. Raible, *Exact exponential-time algorithms for finding bicliques in a graph*, in CTW, 2009, pp. 205–209.

[36] J. Fiala, T. Kloks, and J. Kratochvíl, *Fixed-parameter complexity of lambda-labelings*, Discrete Applied Mathematics, 113 (2001), pp. 59–72.

[37] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Texts in Theoretical Computer Science, Springer, 2006.

[38] F. Fomin and D. Kratsch, *Exact Exponential Algorithms*, Texts in Theoretical Computer Science. an EATCS Series, Springer, 2010.

[39] F. V. Fomin, S. Gaspers, A. V. Pyatkin, and I. Razgon, *On the minimum feedback vertex set problem : Exact and enumeration algorithms*, Algorithmica, 52 (2008), pp. 293–307.

[40] F. V. Fomin, S. Gaspers, and S. Saurabh, *Improved exact algorithms for counting 3- and 4-colorings*, in COCOON, 2007, pp. 65–74.

[41] F. V. Fomin, S. Gaspers, S. Saurabh, and A. A. Stepanov, *On two techniques of combining branching and treewidth*, Algorithmica, 54 (2009), pp. 181–207.

[42] F. V. Fomin, F. Grandoni, and D. Kratsch, *A measure & conquer approach for the analysis of exact algorithms*, Journal of the ACM, 56 (2009).

[43] F. V. Fomin, F. Grandoni, A. V. Pyatkin, and A. A. Stepanov, *Combinatorial bounds via measure and conquer : Bounding minimal dominating sets and applications*, ACM Transactions on Algorithms, 5 (2008).

[44] F. V. Fomin and A. A. Stepanov, *Counting minimum weighted dominating sets*, in COCOON, 2007, pp. 165–175.

[45] F. V. Fomin and Y. Villanger, *Finding induced subgraphs via minimal triangulations*, in STACS, 2010, pp. 383–394.

[46] M. R. Garey and D. S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness*, W. H. Freeman, 1979.

[47] S. Gaspers, D. Kratsch, and M. Liedloff, *On independent sets and bicliques in graphs*, in WG, 2008, pp. 171–182.

[48] S. Gaspers and M. Mnich, *Feedback vertex sets in tournaments*, in ESA (1), 2010, pp. 267–277.

[49] P. A. Golovach, D. Kratsch, and J.-F. Couturier, *Colorings with few colors : Counting, enumeration and combinatorial bounds*, in WG, 2010, pp. 39–50.

[50] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, vol. 57 of Annals of Discrete Mathematics, Elsevier, 2004.

[51] T. Haynes, S. Hedetniemi, and P. Slater, *Fundamentals of Domination in Graphs*, Monographs and Textbooks in Pure and Applied Mathematics, Marcel Dekker, 1998.

[52] M. Held and R. Karp, *A dynamic programming approach to sequencing problems*, SIAM Journal on Applied Mathematics, 10 (1962), pp. 196–210.

[53] P. Hell and J. Nesetril, *On the complexity of h-coloring*, Journal of Combinatorial Theory, Series B, 48 (1990), pp. 92–110.

[54] I. Holyer, *The np-completeness of edge-coloring*, SIAM Journal on Computing, 10 (1981), pp. 718–720.

[55] Y. Iwata, *A faster algorithm for dominating set analyzed by the potential method*, in IPEC, 2011, pp. 41–54.

[56] D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis, *On generating all maximal independent sets*, Information Processing Letters, 27 (1988), pp. 119–123.

[57] M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine, *Enumeration of minimal dominating sets and variants*, in FCT, 2011, pp. 298–309.

[58] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, 1972, pp. 85–103.

[59] ——, *Dynamic programming meets the principle of inclusion and exclusion*, Operations Research Letters, 1 (1982), pp. 49 – 51.

[60] J. Kleinberg and É. Tardos, *Algorithm Design*, Tsinghua University Press, 2006.

[61] M. Koivisto, *An $o^*(2^n)$ algorithm for graph coloring and other partitioning problems via inclusion–exclusion*, in FOCS, IEEE Computer Society, 2006, pp. 583–590.

[62] L. Kowalik, *Improved edge-coloring with three colors*, Theoretical Computer Science, 410 (2009), pp. 3733–3742.

[63] J. Kratochvíl, D. Kratsch, and M. Liedloff, *Exact algorithms for l (2, 1)-labeling of graphs*, in MFCS, 2007, pp. 513–524.

[64] D. Kratsch, *Domination and total domination on asteroidal triple-free graphs*, Discrete Applied Mathematics, 99 (2000), pp. 111–123.

[65] D. Kratsch and L. Stewart, *Domination on cocomparability graphs*, SIAM Journal on Discrete Mathematics, 6 (1993), pp. 400–417.

[66] K. Kutzkov, *An exact exponential time algorithm for counting bipartite cliques*, Information Processing Letters, 112 (2012), pp. 535–539.

[67] E. Lawler, *Combinatorial optimization : networks and matroids*, Holt, Rinehart and Winston, 1976.

[68] E. L. Lawler, *A note on the complexity of the chromatic number problem*, Information Processing Letters, 5 (1976), pp. 66–67.

[69] M. Liedloff, *Algorithmes exacts et exponentiels pour les problèmes NP-difficiles : Domination, variantes et généralisations*, 2007.

[70] P. J. Looges and S. Olariu, *Optimal greedy algorithms for indifference graphs*, Computers & Mathematics with Applications, 25 (1993), pp. 15–25.

[71] V. V. Lozin and R. Mosca, *Independent sets in extensions of $2k_2$-free graphs*, Discrete Applied Mathematics, 146 (2005), pp. 74–80.

[72] K. Makino and T. Uno, *New algorithms for enumerating all maximal cliques*, in SWAT, 2004, pp. 260–272.

[73] R. M. McConnell, *Linear-time recognition of circular-arc graphs*, Algorithmica, 37 (2003), pp. 93–147.

[74] A. A. McRae, *Generalizing np-completeness proofs for bipartite graphs and chordal graphs*, Department of Mathematical science, Clemson university, Ph.D. Dissertation (1994).

[75] G. J. Minty, *On maximal independent sets of vertices in claw-free graphs*, Journal of Combinatorial Theory, Series B, 28 (1980), pp. 284–304.

[76] J. Moon and L. Moser, *On cliques in graphs*, Israel Journal of Mathematics, 3 (1965), pp. 23–28.

[77] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.

[78] J. Nederlof, *Fast polynomial-space algorithms using möbius inversion : Improving on steiner tree and related problems*, in ICALP (1), vol. 5555 of Lecture Notes in Computer Science, Springer, 2009, pp. 713–725.

[79] R. Niedermeier, *Invitation to Fixed-Parameter Algorithms*, Oxford Lecture Series in Mathematics And Its Applications, Oxford University Press, 2006.

[80] Y. Okamoto, R. Uehara, and T. Uno, *Counting the number of matchings in chordal and chordal bipartite graph classes*, in WG, 2009, pp. 296–307.

[81] Y. Okamoto, T. Uno, and R. Uehara, *Counting the number of independent sets in chordal graphs*, Journal of Discrete Algorithms, 6 (2008), pp. 229–242.

[82] S. Olariu, *Paw-fee graphs*, Information Processing Letters, 28 (1988), pp. 53–54.

[83] C. Papadimitriou, *Computational complexity*, Addison-Wesley, 1994.

[84] R. Peeters, *The maximum edge biclique problem is np-complete*, Discrete Applied Mathematics, 131 (2003), pp. 651–654.

[85] J. Pfaff, R. Laskar, and S. Hedetniemi, *Np-completeness of total and connected domination, and irredundance for bipartite graphs*, Department of Mathematical science, Clemson university, Technical Report (1983).

[86] B. Randerath and I. Schiermeyer, *On maximum independent sets in p_5-free graphs*, Discrete Applied Mathematics, 158 (2010), pp. 1041–1044.

[87] R.C.Laskar and J.Pfaff, *Domination and irredundance in split graphs*, Department of Mathematical science, Clemson university, Technical Report (1983).

[88] F. S. Roberts, *Indifference graphs*, Proof techniques in graph theory, (1969), pp. 139–146.

[89] A. Sánchez-Arroyo, *Determining the total colouring number is np-hard*, Discrete Mathematics, 78 (1989), pp. 315–319.

[90] B. Schwikowski and E. Speckenmeyer, *On enumerating all minimal solutions of feedback problems*, Discrete Applied Mathematics, 117 (2002), pp. 253–265.

[91] J. Spinrad, *Efficient Graph Representations*, vol. 19, 2003.

[92] L. G. Valiant, *The complexity of computing the permanent*, Theoretical Computer Science, 8 (1979), pp. 189–201.

[93] J. M. M. van Rooij, *Exact Exponential-Time Algorithms for Domination Problems in Graphs*.

[94] J. M. M. van Rooij, J. Nederlof, and T. C. van Dijk, *Inclusion/exclusion meets measure and conquer*, in ESA, 2009, pp. 554–565.

[95] V. G. Vizing, *On an estimate of the chromatic class of a p-graph (in russian)*, 1964.

[96] D. West, *Introduction to graph theory*, Prentice Hall, 2001.

[97] D. Williamson and D. Shmoys, *The Design of Approximation Algorithms*, Cambridge University Press, 2011.

[98] G. J. Woeginger, *Exact algorithms for np-hard problems : A survey*, in Combinatorial Optimization - Eureka, You Shrink!, vol. 2570 of Lecture Notes in Computer Science, Springer, 2003, pp. 185–208.

[99] E. Wolk, *The comparability graph of a tree*, in Proceedings of the American Mathematical Society, vol. 13, 1962, pp. 189–175.

[100] M. Yannakakis, *Node- and edge-deletion np-complete problems*, in STOC, 1978, pp. 253–264.

[101] M. Yannakakis and F. Gavril, *Edge dominating sets in graphs*, in Society for Industrial and Applied Mathematic, vol. 38, 1980, p. 364–372.