Algorithmes exacts et exponentiels pour les problèmes NP-difficiles : domination, variantes et généralisations

Mathieu Liedloff

▶ **To cite this version:**

Mathieu Liedloff. Algorithmes exacts et exponentiels pour les problèmes NP-difficiles : domination, variantes et généralisations. Autre [cs.OH]. Université Paul Verlaine - Metz, 2007. Français. NNT : 2007METZ027S . tel-01749022

HAL Id: tel-01749022
https://hal.univ-lorraine.fr/tel-01749022v1

Submitted on 29 Mar 2018

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

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 à

l'UNIVERSITÉ PAUL VERLAINE – METZ

pour obtenir le titre de
DOCTEUR DE L'UNIVERSITÉ PAUL VERLAINE – METZ

Spécialité : informatique

Mathieu LIEDLOFF

Algorithmes exacts et exponentiels pour les problèmes NP-difficiles :

domination, variantes et généralisations

Soutenue publiquement à Metz le 7 décembre 2007

Composition du jury :

Président:	
Maurice MARGENSTERN	Professeur des Universités, Université Paul Verlaine, Metz, France
Directeur de thèse:	
Dieter KRATSCH	Professeur des Universités, Université Paul Verlaine, Metz, France
Rapporteurs:	
Pierre FRAIGNIAUD	Directeur de recherche CNRS, Université Denis Diderot, Paris 7, France
Rolf NIEDERMEIER	Professeur Docteur, Université Friedrich Schiller, Iéna, Allemagne
Examinateurs:	
Michel HABIB	Professeur des Universités, Université Denis Diderot, Paris 7, France
John Michael ROBSON	Professeur des Universités, Université de Bordeaux 1, France
Peter ROSSMANITH	Professeur Docteur, Université de Aix-la-Chapelle, Allemagne

LABORATOIRE D'INFORMATIQUE THÉORIQUE ET APPLIQUÉE

Résumé

Les premiers algorithmes exacts exponentiels pour résoudre des problèmes NP-difficiles datent des années soixante. Ces dernières années ont vu un intérêt croissant pour la conception de tels algorithmes tout comme pour l'amélioration de la précision de l'analyse de leur temps d'exécution. Ils sont motivés par les larges applications de problèmes réputés difficiles et qui, sous l'hypothèse P $\neq$ NP, n'admettent pas d'algorithme polynomial en calculant une solution exacte. Dans cette thèse on s'intéresse au problème classique de la domination dans un graphe. On étudie également plusieurs variantes et généralisations de ce problème fondamental. Nous proposons des algorithmes exponentiels pour déterminer un ensemble dominant de taille minimum sur les graphes c-denses, cordaux, 4-cordaux, faiblement cordaux, cercles et bipartis. Puis, nous étudions le problème de la clique dominante qui demande de trouver un ensemble dominant qui soit aussi une clique du graphe. Nous proposons un algorithme Brancher & Réduire qui détermine une clique dominante de taille minimum. L'analyse du temps d'exécution est réalisée en utilisant la technique Mesurer pour Conquérir. Nous donnons ensuite un algorithme général pour énumérer tous les ensembles (σ, ϱ)-dominants d'un graphe en temps $\mathcal{O}(c^n)$, avec $c < 2$, sous certaines conditions sur les ensembles σ et ϱ, et établissons une borne supérieure combinatoire sur leur nombre. Finalement, nous nous intéressons à un problème de domination partielle et obtenons un algorithme pour le problème de la domination romaine. Grâce à un algorithme basé sur le paradigme de la Programmation Dynamique, nous proposons un algorithme pour le problème de la domination avec des puissances variables.

Mots clés

algorithmes, graphes, problèmes NP-complets, algorithmes exacts, algorithmes en temps exponentiels, problèmes de domination

---

Abstract

The first exact exponential-time algorithms solving NP-hard problems date back to the sixties. The last years have seen an increasing interest for designing such algorithms as well as analysing their running time. The existence of many applications of well known hard problems is one of the main motivations. Moreover, under the hypothesis P $\neq$ NP, a polynomial time algorithm for these problems does not exist. In this thesis, we deal with the classical domination problem in graphs. We are also interested in some variants and generalizations of this fondamental problem. We give exponential-time algorithms for computing a minimum dominating set on c-dense graphs, chordal graphs, 4-chordal graphs, weakly chordal graphs, circle graphs and bipartite graphs. Then, we study the dominating

clique problem requiring to find a minimum dominating set inducing a clique of the graph. We provide a Branch & Reduce algorithm computing a minimum dominating clique. The analysis of the running time is done by using the Measure and Conquer technique. Afterwards, we propose a general algorithm for enumerating all (σ, ϱ)-dominating sets of a graph in time $\mathcal{O}(c^n)$, with $c < 2$, under some assumptions on the sets σ and ϱ. Subsequently, we establish a combinatorial upper bound on the number of such sets in a graph. Finally, we consider a partial dominating set problem and we give an algorithm for solving the Roman domination problem. Using the dynamic programming paradigm, we obtain an algorithm for the domination problem with flexible powers.

Keywords

algorithms, graphs, NP-complete problems, exact algorithms, exponential-time algorithms, domination type problems

Remerciements

En premier lieu, je tiens à remercier très chaleureusement Dieter Kratsch, mon directeur de thèse, sans qui celle-ci n'aurait vu le jour. Son accompagnement, sa patience et l'intérêt qu'il a porté à mes travaux ont mené à l'aboutissement de mes recherches. J'aurais tant à raconter sur ces années passées à ses côtés, si riche d'enseignement. Non seulement Dieter m'a transmis de nombreuses connaissances mais il a provoqué en moi une véritable passion pour la recherche. Mes maigres connaissances de la langue de Goethe, me permettent seulement d'apporter ce court témoignage : « *Herzlichen Dank an meinen Doktorvater* ».

Je me dois aussi de remercier Nora Kratsch, l'épouse de Dieter, et leur fils Stefan pour l'accueil qu'ils m'ont plusieurs fois réservé à Jena (*Iéna*, en français).

Je remercie vivement Pierre Fraigniaud et Rolf Niedermeier, rapporteurs de cette thèse. L'attention qu'ils ont accordée à mes travaux ainsi que leurs commentaires m'ont été précieux. Mes remerciements les plus sincères vont également à Michel Habib, à Maurice Margenstern, à John Michael Robson ainsi qu'à Peter Rossmanith qui ont accepté d'examiner ma thèse.

Je souhaite également témoigner toute ma reconnaissance à mes coauteurs et aux personnes avec qui j'ai travaillé : Fedor Fomin, Serge Gaspers, Petr Golovach, Ton Kloks, Jan Kratochvíl, Dieter Kratsch, Haiko Müller, Sheng-Lung Peng, Michaël Rao, Saket Saurabh, Ioan Todinca ainsi qu'à Jim Liu, pour qui j'ai une pensée particulière. Les collaborations entretenues avec eux m'ont beaucoup appris. J'exprime toute ma gratitude à Fedor Fomin pour son invitation durant deux semaines dans le groupe d'algorithmique à Bergen en Norvège.

Bien sûr, je remercie les membres du LITA avec qui j'ai partagé mes travaux lors de discussions ou d'un séminaire. En particulier, je remercie Maurice Margenstern, directeur du LITA, avec qui j'ai eu grand plaisir à partager régulièrement quelques instants de conversation.

Je remercie les membres du département d'informatique de l'université Paul Verlaine – Metz, et en particulier Aristide Grange, pour son encadrement en tant que tuteur pédagogique et son amitié, Daniel Singer, pour nos discussions scientifiques, Christian Minich, pour son intérêt pour l'algorithmique, Loïc Colson, pour ses invitations gourmandes.

Je veux aussi remercier le personnel du département, du laboratoire et du service informatique : Martine, Catherine, Jocelyne, Dominique, Damien et Éric, en pensant aussi à Martine. Leur présence quotidienne permet de s'affranchir des petits tracas techniques.

J'en profite aussi pour remercier mes amis jeunes docteurs, doctorants et ex-doctorants : Mihaela, David, Nicolas, Sergey, Thomas et en particulier Michaël Rao et Serge Gaspers.

Enfin, merci à tous mes amis qui m'ont soutenu au cours de ces dernières années. Je remercie très chaleureusement mes parents, sans oublier Paco, pour leur affection et leur soutien logistique. C'est à eux que je dédie cette thèse.

à tous, merci,
Mathieu Liedloff

Table des matières

Introduction

Des problèmes difficiles

La NP-complétude est une notion bien connue des algorithmiciens et de la plupart des personnes qui écrivent des algorithmes. Depuis le début des années 1970, en particulier depuis la démonstration de la NP-complétude du problème SAT en 1971 par Cook [Coo71], l'étude des problèmes NP-complets, tout comme le nombre de problèmes montrés être difficiles, n'ont cessé de croître. De nombreuses attaques ont été entreprises pour tenter de répondre à la grande question ouverte de l'informatique : « est-ce que P = NP ? ». Cette question fondamentale demande d'affirmer ou d'infirmer le fait qu'un problème NP-complet puisse être résolu en temps polynomial. Bien que cette question soit encore une véritable énigme, elle suscite toujours un intérêt incontestable ; intérêt justifié par l'étendue des domaines où apparaissent ces problèmes si difficiles : mathématiques, réseaux, ordonnancement, biologie, jeux... Même le jeu très en vogue appelé *Sudoku* est NP-complet lorsqu'on considère des grilles de côté n^2 et que l'on doit décider si une grille donnée admet une solution [YS02]. Dans ce jeu, une grille de côté n^2 contient n^2 blocs de côté n qu'il faut réussir à compléter de sorte que pour chaque ligne, pour chaque colonne et pour chaque bloc, les n symboles apparaissent exactement une fois. Les problèmes NP-complets sont non seulement l'objet d'étude dans cette thèse, mais ils sont partout dans notre quotidien !

L'intuition erronée qui a parfois été développée autour de ces problèmes est que l'on ne peut pas les résoudre. Difficiles à résoudre certainement, mais pas impossible. D'un point de vue mathématique, cette intuition est incorrecte : une résolution par *recherche exhaustive* permet toujours la découverte d'une solution et donc la résolution du problème. D'un point de vue pratique, cela est discutable. Si l'instance du problème à résoudre est *grande*, on ne sait pas, à ce jour, déterminer une solution dans un temps raisonnable. Néanmoins tout dépend de la signification qui est donnée au mot *grande*. En tout état de cause, l'objectif de cette thèse est de donner une signification bien plus « grande » à ce mot. Précisément nous chercherons à résoudre quelques problèmes NP-complets par l'entremise d'algorithmes dont le temps d'exécution sera asymptotiquement inférieur à celui nécessaire à une recherche exhaustive.

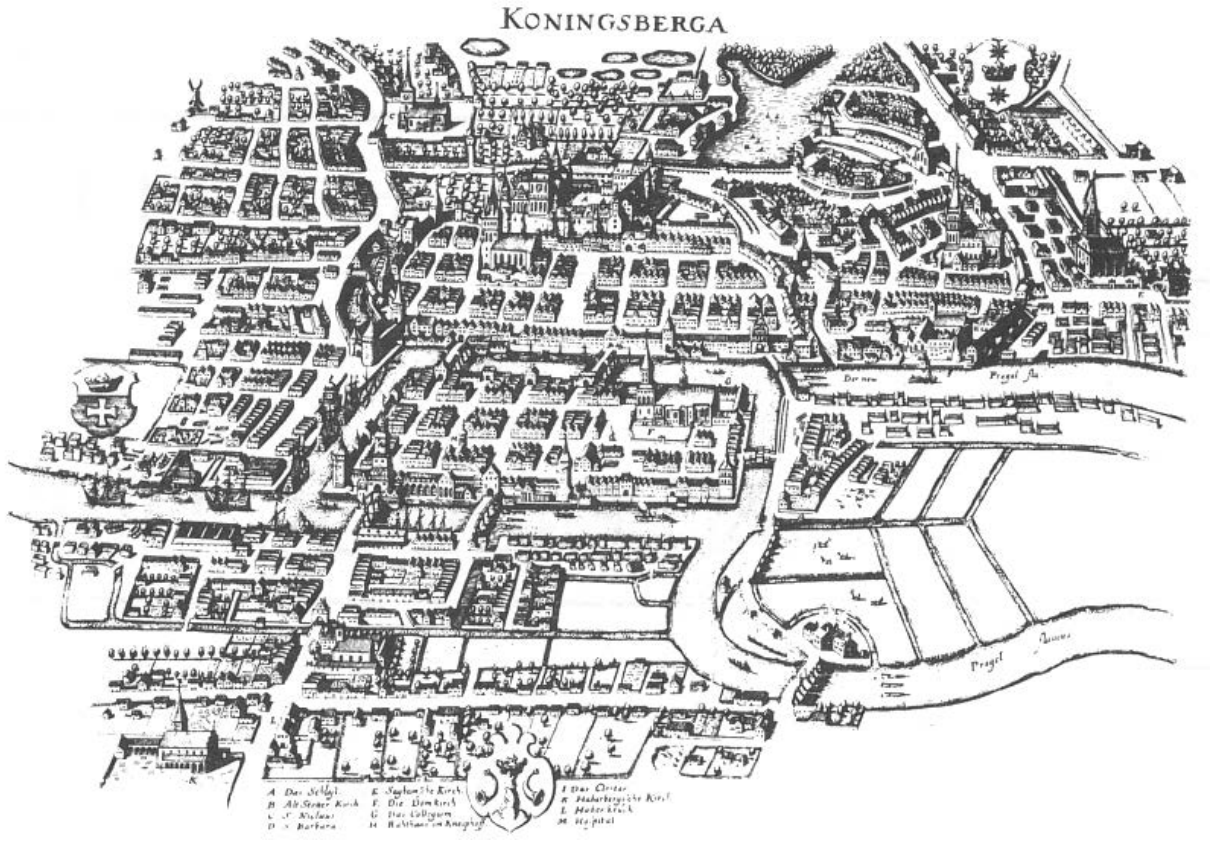
L'obtention d'un algorithme polynomial pour résoudre un problème NP-complet permettrait immédiatement de répondre à la question P = NP. Sous l'hypothèse P $\neq$ NP, de tels algorithmes sont impossibles à obtenir. Aujourd'hui, peu d'indices permettent de se faire une idée sur cette question et beaucoup de chercheurs pensent que P $\neq$ NP (voir

[G02]). Les algorithmes qui sont proposés dans cette thèse ont donc un temps d'exécution exponentiel au pire des cas.

De simples graphes

Un graphe est une structure simple composée de sommets, également appelés nœuds, et d'arêtes permettant d'établir des relations entre ceux-ci. Ils permettent de modéliser divers problèmes dont les champs d'applications sont larges : mathématiques, biologie, télécommunication, physique, économie, sciences humaines, informatique, électronique, ...

L'émergence de la théorie des graphes se situe au XVIIIe siècle avec un article datant de 1736 écrit par Leonhard Euler (1707 - 1783). On doit à ce mathématicien suisse la naissance de la topologie et les tous premiers fondements de la théorie des graphes. Le problème, maintenant devenu classique, qui se pose dans [Eul1736] est le suivant : comment traverser précisément une seule fois les sept ponts de la ville de Königsberg (voir la figure suivante) ? On sait aujourd'hui que cela est impossible car un graphe connexe admet un cycle eulérien si et seulement si le graphe ne possède que des sommets de degré pair.



La ville de Königsberg et ses sept ponts.

Des problèmes de graphes pas si simples

Les problèmes se modélisant et se posant sur les graphes sont parfois simples à résoudre et parfois beaucoup plus difficiles. Le temps d'exécution d'un algorithme est un facteur important dans l'évaluation de son efficacité et c'est pourquoi l'algorithmique des graphes occupe aujourd'hui une place importante en informatique fondamentale et appliquée.

L'objet du présent travail est de s'intéresser à quelques problèmes de graphes pour lesquels on ne connaît pas d'algorithme polynomial pour les résoudre, malgré d'importants efforts depuis une trentaine d'années. L'angle d'attaque que nous entreprenons est de résoudre de façon exacte un problème par l'entremise d'algorithmes dont le temps d'exécution au pire des cas est exponentiel. Notre but étant de réduire le plus possible ce temps d'exécution, c'est-à-dire de réduire la base de l'exponentiel. En utilisant la notation asymptotique $\mathcal{O}^*$, le temps d'exécution sera de la forme $\mathcal{O}^*(c^n)$ où n est la taille de l'instance et c une constante la plus petite possible. La notation $\mathcal{O}^*$ que nous emploierons au long de cette thèse, introduite par Woeginger dans [Woe03], autorise la suppression de tous les termes bornés par un facteur polynomial. Ainsi, une complexité de la forme $\mathcal{O}(f(n) \cdot poly(n))$, où $poly(n)$ est une fonction polynomiale, s'écrira par $\mathcal{O}^*(f(n))$. L'objectif de cette notation est de se concentrer sur la croissance exponentielle du temps d'exécution ; c'est sur ce terme que nos efforts se porteront.

Les problèmes auquels on s'attache sont des problèmes de domination. Il s'agit soit proprement du problème classique Domination, soit de variantes ou de généralisations. Ce problème, référencé par [GT2] dans l'ouvrage de Garey et Johnson [GJ79], se définit simplement par :

Domination (Dominating Set)

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Existe-t-il un sous-ensemble $D \subseteq V$ de cardinalité au plus k tel que pour tout sommet $u \in V \setminus D$, u a au moins un voisin dans D ?

Malgré la grande simplicité de sa définition, il est NP-complet. De par sa remarquable utilité, ce problème de graphe fondamental a fait l'objet de nombreux travaux et deux monographies [HHS98a, HHS98b] le traitant ont été publiées en 1998. Il permet notamment d'exprimer des problèmes qui, par exemple dans un réseau, demandent qu'une certaine ressource soit disponible en local ou chez l'un des voisins immédiats en utilisant le moins possible de cette ressource. Les applications peuvent également survenir en sciences sociales : on dispose d'une population pour laquelle on connaît des relations de connaissances entre individus ; on souhaite trouver un plus petit sous-ensemble de la population tel que chaque individu n'appartennant pas à ce sous-ensemble connaît au moins une personne du sous-ensemble sélectionné.

Des algorithmes exponentiels

Attaquer un problème NP-complet sans se confronter directement à la grande question ouverte « P = NP » n'est pas une idée nouvelle. Parmi ces attaques, il y a :

- l'élaboration d'algorithmes déterminant en temps polynomial une solution approchée ;
- la restriction à certaines instances pour lesquelles un algorithme polynomial produisant une solution exacte est possible ;
- la conception d'algorithme à paramètre fixé.

Toutes ces attaques ont donné naissance à des domaines de recherche emprunt d'une réelle dynamique. Mais elles subissent également quelques « limites ». En témoigne le problème DOMINATION pour lequel un algorithme polynomial donnant une solution approchée à un facteur constant de l'optimum ou encore l'obtention d'un algorithme à paramètre fixé est impossible, sauf si des relations de complexité supposées être improbables sont vérifiées. De plus, le problème reste NP-complet pour de nombreuses classes de graphes. La résistance de ce problème à de telles attaques motive l'étude de « bons » algorithmes exponentiels.

Les premiers algorithmes exponentiels non triviaux pour résoudre un problème NP-complet datent des années soixante et soixante-dix. Des algorithmes pour le problème SAT sont alors publiés [DP60, DLL62] et leur popularité font, qu'encore aujourd'hui, ils sont connus sous le nom de leurs auteurs : Davis-Putnam et Davis-Logemann-Loveland. Held et Karp donnent également en 1962 [HK62] un algorithme pour résoudre le problème du voyageur de commerce en temps $\mathcal{O}^*(2^n)$ alors que l'algorithme naïf était en $\mathcal{O}^*(n!)$. Encore aujourd'hui, cet algorithme est le meilleur connu. Plus tard, Horowitz et Sahni [HS74] proposent un algorithme en $\mathcal{O}^*(2^{n/2})$ pour résoudre le problème du sac à dos binaire, améliorant le précédent algorithme naïf en $\mathcal{O}^*(2^n)$. La signification d'un tel résultat est importante : si en une certaine unité de temps on est capable de traiter une entrée de taille k avec l'algorithme en $\mathcal{O}^*(2^n)$, alors grâce à un algorithme en $\mathcal{O}^*(2^{n/2})$, dans la même unité de temps, une entrée de taille $2k$ peut être traitée. En 1977, Tarjan et Trojanowski donnent un algorithme en $\mathcal{O}^*(2^{n/3})$ pour résoudre le problème ENSEMBLE STABLE. Cet article provoquera un certain engouement puisque ensuite des publications proposeront d'autres algorithmes exponentiels [Jia86, Rob86, ST90, Bei99, FGK06a].

Depuis 2003, plusieurs états de l'art ont été écrits [FGK05b, Iwa04, Sch05, Woe03, Woe04] ainsi que de nombreux articles proposant des algorithmes exponentiels. Cela témoigne de l'intérêt croissant pour la résolution des problèmes NP-complets par le biais d'algorithmes exacts exponentiels. Ces états de l'art décrivent les progrès du domaine et présentent les principales techniques utilisées jusqu'alors.

Présentation des chapitres de la thèse

Chapitre 1 : préliminaires

Le premier chapitre présentera les définitions et notions habituelles sur la complexité des algorithmes, les graphes et divers problèmes bien connnus. Plusieurs pistes pour attaquer un problème NP-complet seront également évoquées. Enfin, le chapitre terminera par la présentation de quelques éléments de combinatoire qui seront utiles par la suite.

Chapitre 2 : algorithmes exponentiels

Nous motiverons au chapitre 2, l'intérêt croissant pour les algorithmes exacts exponentiels. Nous en profiterons pour donner des résultats connus pour des problèmes NP-difficiles ; en particulier, on s'intéressera au problème Ensemble Stable qui a provoqué la publications de nombreux algorithmes exponentiels. La seconde partie de ce chapitre se consacrera à l'étude des principales techniques couramment utilisées pour obtenir des algorithmes exponentiels. Les applications de ces techniques seront illustrées par des exemples qui témoigneront de l'essort des algorithmes exponentiels. Ces techniques pourront servir de base pour la conception de nouveaux algorithmes. Elles sont le pendant des techniques habituelles de l'algorithmique lorsqu'il s'agit de concevoir des algorithmes polynomiaux, comme *diviser pour régner*, *stratégie gloutonne*, *programmation dynamique*. Nous verrons que quelque unes de ces techniques classiques peuvent aussi être utilisées dans la conception d'algorithmes exponentiels. On s'attachera également à donner quelques analyses de temps d'exécution afin de se familiariser aux techniques usuellement employées pour établir une borne asymptotique supérieure au pire des cas.

Chapitre 3 : problème de la domination et état de l'art

Cette thèse gravite autour du problème fondamental Domination. Jusqu'en 2004, le seul algorithme connu pour résoudre le problème de façon exacte demandait un temps $\mathcal{O}^*(2^n)$. Aujourd'hui, le meilleur algorithme publié est dû à Fomin *et al.* et réclame un temps $\mathcal{O}(1.5137^n)$. Au chapitre 3, nous présenterons les algorithmes publiés depuis 2004 pour résoudre ce problème de la domination. Ceci permettra d'établir un historique de la résolution de Domination du point de vue des algorithmes exponentiels et montrera que la conception de tels algorithmes pour ce problème demande quelques efforts.

Chapitre 4 : ensemble dominant sur quelques classes de graphes

Il est connu que le problème Domination reste NP-complet, même lorsqu'il est restreint aux graphes cordaux, 4-cordaux, faiblement cordaux, cercles et bipartis. Nous montrons au chapitre 4 que le problème l'est aussi lorsque l'on considère des graphes c-denses.

Au chapitre 4, nous proposons une approche générale basée sur la largeur arborescente du graphe donné en entrée pour résoudre Domination sur diverses classes de graphes.

Nous établissons des bornes linéaires supérieures sur la largeur arborescente en fonction du degré maximum. Précisément, nous montrons que $tw(G) \leq 4\Delta(G)$ si G est un graphe cercle, $tw(G) \leq 3\Delta(G)$ si G est un graphe 4-cordal, $tw(G) \leq 2\Delta(G)$ si G est un graphe faiblement cordal. Ces résultats, combinés avec l'approche générale que nous proposons, aboutiront à un algorithme en temps $\mathcal{O}(1.4956^n)$ pour les graphes cercles, en temps $\mathcal{O}(1.4913^n)$ pour les graphes 4-cordaux et en temps $\mathcal{O}(1.4842^n)$ pour les graphes faiblement cordaux.

D'un autre côté, si le graphe est cordal alors on sait que $tw(G) \leq \Delta(G)$. Alber *et al.* ont publié dans [ABFKN02] un algorithme qui résout DOMINATION en temps $\mathcal{O}^*(4^{tw(G)})$. Nous montrerons qu'il est possible d'adapter l'algorithme de Alber *et al.* de sorte que si l'entrée est un graphe cordal alors la résolution de DOMINATION demande un temps $\mathcal{O}^*(3^{tw(G)})$. Grâce à cet algorithme, combiné avec une approche que nous développerons et intitulerons « beaucoup de sommets de grand degré », nous établirons un algorithme en temps $\mathcal{O}(1.4173^n)$ pour résoudre DOMINATION sur les graphes cordaux. De plus, nous montrerons que l'application de l'approche « beaucoup de sommets de grand degré » implique un algorithme qui s'exécute en temps $\mathcal{O}(1.2303^{n(1+\sqrt{1-2c})})$ pour résoudre DOMINATION sur les graphes c-denses.

Finalement, ce chapitre répond à la question demandant si DOMINATION peut-être résolu en temps $\mathcal{O}^*(\alpha^n)$ sur les graphes bipartis, avec $\alpha < 1.5137$. Nous montrons que pour tout graphe ayant un ensemble stable de taille z, la résolution de DOMINATION peut être réalisée en temps $\mathcal{O}^*(2^{n-z})$ grâce à la conception d'un algorithme basé sur le paradigme de la Programmation Dynamique. En particulier, ce résultat implique que le problème DOMINATION peut être résolu en temps $\mathcal{O}(1.7088^n)$ sur un graphe quelconque et en temps $\mathcal{O}^*(2^{n/2}) = \mathcal{O}(1.4143^n)$ sur un graphe biparti.

Ce chapitre est principalement basé sur les travaux publiés dans :

[GKL06] Serge Gaspers, Dieter Kratsch, Mathieu Liedloff,
Exponential Time Algorithms for the Minimum Dominating Set Problem on Some Graph Classes,
Proceedings of SWAT 2006, Springer-Verlag, 2006, Berlin, LNCS 4059, p. 148–159.

[GKLT07] Serge Gaspers, Dieter Kratsch, Mathieu Liedloff, Ioan Todinca,
Exponential Time Algorithms for the Minimum Dominating Set Problem on Some Graph Classes,
article sousmis à *ACM Transactions on Algorithms*.

Chapitre 5 : problème de la clique dominante

Le chapitre 5 s'intéresse à une variante du problème classique de la domination appelée CLIQUE DOMINANTE. Ce problème demande de déterminer un ensemble dominant d'un graphe qui soit aussi une clique. Déjà le problème d'existence « décider si un graphe a une clique dominante » est NP-complet ; ceci implique que les deux versions d'optimisation qui demandent de trouver une clique dominante de plus grande taille possible ou de plus petite taille sont NP-difficiles.

Un algorithme exponentiel de type Brancher & Réduire de Culberson *et al.* et publié en 2005 [CGA05] propose la résolution du problème d'existence. L'objet de l'algorithme de Culberson *et al.* est la réalisation d'études expérimentales et aucune analyse du temps d'exécution au pire des cas de leur algorithme n'est donnée. Nous nous intéressons naturellement à la complexité de cet algorithme et montrons que son temps d'exécution au pire des cas est borné inférieurement par $\Omega(1.4142^n)$.

Nous étudierons ensuite au problème de la clique dominante de taille minimum. Des règles de réduction et de branchement nous permettront d'établir un algorithme Brancher & Réduire pour résoudre récursivement ce problème. Grâce à une analyse utilisant une mesure non standard sur la taille d'un (sous-)problème, asssocié à la technique Mesurer pour Conquérir, nous montrerons que notre algorithme s'exécute en temps $\mathcal{O}(1.3387^n)$ au pire des cas et utilise un espace polynomial pour déterminer une clique dominante de taille minimum d'un graphe à n sommets. On note également que cet algorithme résout, dans le même temps, le problème d'existence et est donc au pire des cas plus rapide que celui de Culberson *et al.*.

Puisque les meilleures méthodes d'analyse dont nous disposons pour établir la complexité des algorithmes de type Brancher & Réduire semblent sur-estimer le temps d'exécution des algorithmes, une borne asymptotique inférieure $\Omega(1.2599^n)$ sur le temps d'exécution au pire des cas de notre algorithme sera démontrée.

Enfin, nous montrons qu'au détriment de la nécessité d'un espace exponentiel, il est possible d'améliorer la borne asymptotique supérieure de $\mathcal{O}(1.3387^n)$ à $\mathcal{O}(1.3234^n)$ en utilisant la technique Mémorisation.

Ce chapitre est basé sur les travaux publiés dans :

[KL06] Dieter Kratsch, Mathieu Liedloff,
An Exact Algorithm for the Minimum Dominating Clique Problem,
Proceedings of IWPEC 2006, Springer-Verlag, 2006, Berlin, LNCS 4169, p. 130–141.

[KL07] Dieter Kratsch, Mathieu Liedloff,
An Exact Algorithm for the Minimum Dominating Clique Problem,
accepté pour publication dans *Theoretical Computer Science*.

Chapitre 6 : une généralisation de la domination

Au chapitre 6 nous étudions une généralisation du problème DOMINATION : la (σ, ϱ)-domination. Étant donné deux ensembles $\sigma \subseteq \mathbb{N}$ et $\varrho \subseteq \mathbb{N}$, un sous-ensemble de sommets S d'un graphe $G = (V, E)$ est un ensemble (σ, ϱ)-dominant si :
(i) $|N(v) \cap S| \in \sigma$ pour tout sommet $v \in S$, et
(ii) $|N(v) \cap S| \in \varrho$ pour tout sommet $v \in V \setminus S$.
L'ensemble $N(v)$ désigne l'ensemble des voisins du sommet v dans le graphe G.

Ce problème, introduit par Telle [Tel94], généralise un grand nombre de problèmes de domination. Pour beaucoup de choix d'ensembles σ et ϱ, les problèmes demandant de décider si un graphe possède un ensemble (σ, ϱ)-dominant ou de chercher un ensemble (σ, ϱ)-dominant de taille minimum ou maximum, sont des problèmes NP-complets bien étudiés.

Au chapitre 6, nous nous intéressons aux problèmes de la (σ, ϱ)-domination demandant de décider l'existence, d'énumérer et de compter ces ensembles, ainsi que de déterminer un ensemble (σ, ϱ)-dominant de plus petite ou de plus grande taille. Notons que la résolution du problème d'énumération qui demande de lister tous les ensembles (σ, ϱ)-dominants permet immédiatement de résoudre le problème d'existence, de minimisation et de maximisation ainsi que de compter le nombre de ces ensembles.

Nous proposons un algorithme général énumérant tous les ensembles (σ, ϱ)-dominants d'un graphe G en temps $\mathcal{O}^*(c^n)$ pour un certain $c < 2$, nécessitant seulement un espace polynomial, si l'ensemble σ est sans successeur (c'est-à-dire que σ ne contient pas deux entiers consécutifs) et si les deux ensembles σ et ϱ sont finis, ou si l'un d'eux est infini et $\sigma \cap \varrho = \emptyset$. La technique utilisée dans l'algorithme ainsi que dans l'analyse du temps d'exécution est une combinaison de la technique standard de branchement avec un astucieux mécanisme de rechargement. Cette nouvelle technique que nous appelons *Brancher & Recharger* nous semble assez générale pour être utile pour résoudre d'autres problèmes.

Une conséquence combinatoire de notre algorithme est une borne supérieure non triviale c^n, avec $c < 2$, sur le nombre d'ensembles (σ, ϱ)-dominants d'un graphe à n sommets sous les conditions sus-citées concernant σ et ϱ. Ces bornes supérieures établies par l'approche générale ne sont probablement pas égales aux bornes inférieures pour des valeurs particulières de (σ, ϱ) et c'est pourquoi nous établirons également quelques bornes inférieures.

Enfin, nous utiliserons le paradigme peu connu Trier & Chercher pour établir des algorithmes permettant de trouver les ensembles $(\{p\}, \{q\})$-dominants de taille maximum et de taille minimum, ainsi que pour compter le nombre d'ensembles $(\{p\}, \{q\})$-dominants en temps $\mathcal{O}^*(2^{n/2})$. En utilisant une astuce nous montrerons comment étendre la technique pour résoudre le problème (σ, ϱ)-domination pour quelques ensembles σ et ϱ non réduits à des singletons.

Ce chapitre est principalement basé sur les travaux publiés dans :

[FGKKL07] Fedor V. Fomin, Petr A. Golovach, Jan Kratochvíl, Dieter Kratsch, Mathieu Liedloff,
Branch and Recharge : Exact Algorithms for Generalized Domination,
Proceedings of WADS 2007, Springer-Verlag, 2007, Berlin, LNCS 4619, p. 508–519.

Chapitre 7 : autres généralisations de la domination

Dans le dernier chapitre de cette thèse, nous nous intéressons à plusieurs problèmes généralisant la domination. Tout d'abord le problème Domination Partielle demande, étant donné un graphe $G = (V, E)$ et un entier t, de déterminer un ensemble dominant tel qu'au moins $t \leq |V|$ sommets de G soient dominés. L'algorithme sera de type Brancher & Réduire et son analyse fera appel à une fonction de mesure non standard afin d'établir un temps d'exécution au pire des cas de $\mathcal{O}(1.6183^n)$. Nous montrerons qu'une solution de coût minimum au problème Domination Romaine, qui permet à un sommet v de se dominer pour un coût de 1 et de dominer son voisinage $N[v]$ pour un cout de 2, peut

également être déterminée en temps $\mathcal{O}(1.6183^n)$. Nous étudierons ensuite le problème k-CENTRE PARTIEL, généralisant le problème plus connu k-CENTRE et pour lequel $t = n$ dans la définition suivante. Étant donné un graphe $G = (V, E)$, une fonction de pondération $w : E \to \mathbb{R}$ définie pour chaque arête du graphe, et deux entiers $k, l \in \mathbb{N}$, le problème k-CENTRE PARTIEL réclame de trouver un ensemble S tel que

(i) $|S| \leq k$; et

(ii) si on ordonne les sommets de $V = \{v_1, \ldots, v_n\}$ de sorte que

$$w(v_1, S) \leq w(v_2, S) \leq \cdots \leq w(v_n, S),$$

$$\text{alors } w(v_t, S) \leq l, \text{ avec } w(v_i, S) = \begin{cases} 0 & \text{si } v_i \in S; \\ \min\limits_{u \in S} w(\{v_i, u\}) & \text{si } v_i \notin S. \end{cases}$$

Nous montrerons que k-CENTRE PARTIEL peut être résolu en temps $\mathcal{O}^*(c^n)$ au pire des cas si et seulement s'il est possible de résoudre DOMINATION PARTIELLE en temps $\mathcal{O}^*(c^n)$ au pire des cas.

Dans la seconde partie du chapitre 7, nous étudierons une autre généralisation du problème classique de la domination, appelée DOMINATION AVEC DES PUISSANCES VARIABLES, où les sommets peuvent se voir attribuer des puissances qui leur permettent de dominer des sommets jusqu'à une certaine distance. À chaque puissance est associé un coût et le problème demande de déterminer la puissance à attribuer à chaque sommet de façon à dominer tous les sommets du graphe et à minimiser le coût total. Nous montrerons quelques résultats de NP-complétude : si l'ensemble des puissances autorisées est l'ensemble des entiers positifs et si la fonction de coût est égale à d^i pour toute puissance i, avec $d \geq 2$ un entier fixé, alors le problème DOMINATION AVEC DES PUISSANCES VARIABLES est NP-complet. Pour terminer, nous donnerons un algorithme basé sur le paradigme de la Programmation Dynamique pour résoudre ce problème de façon générale en temps $\mathcal{O}^*(2^n)$ et nécessitant un espace exponentiel.

Une partie des travaux présentés au début du chapitre 7, concernant le problème DOMINATION PARTIELLE, a été initiée lors d'une collaboration avec Fedor Fomin, Serge Gaspers, Dieter Kratsch et Saket Saurabh, pendant une visite à l'université de Bergen en Norvège.

Chapitre 1

Préliminaires

On s'attache dans ce premier chapitre à rappeler quelques définitions et notions fondamentales relatives aux algorithmes, à la complexité, aux graphes et aux problèmes auxquels nous nous intéresserons dans la suite de cette thèse. Nous en profitons également pour introduire le vocabulaire et les notations utilisées. Une liste des symboles usités est proposée à la fin de cet ouvrage et le lecteur pourra s'y référer en cas de nécessité. Enfin, nous donnons quelques éléments de combinatoire qui serviront à établir et à analyser quelques algorithmes présentés.

1.1 Algorithmes et complexité

1.1.1 Algorithmes

Un *algorithme* est une procédure de calcul bien spécifiée qui attend en entrée une valeur, ou un ensemble de valeurs, et qui produit en sortie une valeur ou un ensemble de valeurs. C'est un séquencement d'étapes de calcul permettant de passer d'une valeur d'entrée à une valeur de sortie de sorte que l'algorithme résout un certain problème. La notion d'*efficacité* d'un algorithme demande d'ordinaire à considérer l'ensemble des ressources mises en œuvre pour résoudre un problème ; souvent ce qui nous intéresse est de minimiser le temps d'exécution requis par l'algorithme ou l'espace de stockage nécessaire à l'algorithme durant son exécution. C'est essentiellement sur ces deux ressources que nous devons concentrer nos efforts afin d'en limiter le coût. Le temps d'exécution est exprimé en fonction de la taille de l'instance du problème ; par exemple pour un algorithme qui attend en entrée un graphe, le temps d'exécution pourra dépendre du nombre de sommets ou du nombre d'arêtes du graphe. Pour l'analyse de la complexité des algorithmes, nous utiliserons la notation $\mathcal{O}$ habituelle ou la notation $\mathcal{O}^*$ définie à la prochaine section. Nous donnerons un rappel de sa définition à la section suivante. De plus, afin de nous affranchir des contraintes propres à chaque langage de programmation et en ayant pour objectif de rendre la compréhension et la description des algorithmes la plus aisée possible, ceux-ci seront écrits dans un pseudo-code naturel proche du langage Pascal ou C. Le pseudo-code employé se voudra ni

équivoque ni ambigu et sera celui traditionnellement utilisé dans les cours d'algorithmique ainsi que dans nombre d'articles de recherche.

1.1.2 Classes de complexité et réductions polynomiales

Pour les différentes classes de complexité usuelles, nous ne donnerons pas dans cette section des définitions techniques de la théorie de la complexité. Le lecteur intéressé par de telles définitions formelles pourra se référer aux ouvrages [GJ79, Pap94] (ainsi qu'au mémoire [Lie03]). On y trouvera également des explications quant aux modèles de calcul utilisés pour introduire formellement ces définitions, notament les machines de Turing déterministes et non déterministes.

1.1.2.1 Les notations $\mathcal{O}$, Ω, Θ et $\mathcal{O}^*$

Lors de l'évaluation de la complexité d'un algorithme, il est habituel d'utiliser la notation $\mathcal{O}$ qui permet d'exprimer une borne supérieure asymptotique sur le temps d'exécution de l'algorithme. Formellement, si on désigne par $T(n)$ le temps d'exécution d'un algorithme sur une entrée de taille n, étant donné une fonction $f(n)$, on dit que $T(n)$ est en $\mathcal{O}(f(n))$, s'il existe deux constantes positives c et n_0 telles que pour tout $n \geq n_0$, on a la relation $T(n) \leq c \cdot f(n)$.

De façon similaire, la notation $\Omega(f(n))$ permet de donner une borne asymptotique inférieure sur le temps d'exécution d'un algorithme sur une entrée de taille n. Précisément, on dit qu'un temps d'exécution $T(n)$ est en $\Omega(f(n))$, s'il existe deux constantes positives c et n_0 telles que pour tout $n \geq n_0$, on a $T(n) \geq c \cdot f(n)$.

Si une fonction $T(n)$ est à la fois en $\mathcal{O}(f(n))$ et en $\Omega(f(n))$, on dit que $T(n)$ est en $\Theta(f(n))$.

Dans la suite, lorsque nous étudierons le temps d'exécution d'un algorithme il s'agira généralement du temps d'exécution au pire des cas. Cela signifie que si $T(n)$ dénote le temps d'exécution au pire des cas d'un algorithme, alors pour toute entrée son temps d'exécution n'exédera pas $T(n)$, à un facteur multiplicatif constant près. Pour davantage de précisions sur ces notions, nous nous référons aux ouvrages [CLRS02] et [KT06].

Pour compléter ces trois notations, Woeginger a introduit dans [Woe03] la notation $\mathcal{O}^*$ qui autorise la suppression de tous les termes bornés par un facteur polynomial dépendant de la taille de l'entrée. Ainsi une complexité de la forme $\mathcal{O}(f(n) \cdot poly(n))$, où n est la taille de l'entrée et $poly(n)$ un polynôme dépendant de n, se notera $\mathcal{O}^*(f(n))$. Cela nous permettra en particulier d'insister sur la croissance exponentielle de la fonction qui borne le temps d'exécution. Par exemple un temps d'exécution $2 + n^3 \cdot 1.8788^n$ se réécrira $\mathcal{O}^*(1.8788^n)$. Cette nouvelle notation asymptotique se justifie par le fait que $2 + n^3 \cdot 1.8788^n$ est compris entre 1.8788^n et 1.8789^n pour toute valeur de n supérieure à 26. Par conséquent, la fonction $2 + n^3 \cdot 1.8788^n$ pourra également se réécrire simplement sous la forme $\mathcal{O}(1.8789^n)$. C'est cette dernière forme de notation que nous privilégierons par la suite.

1.1.2.2 Les classes P et NP

La classe de complexité P contient l'ensemble des problèmes pour lesquels il existe un algorithme déterministe qui s'exécute au pire des cas en temps polynomial pour les résoudre. Typiquement, ce temps d'exécution est de la forme $\mathcal{O}(n^k)$ où n représente la taille de l'entrée et k est une constante (qui dépend de l'algorithme considéré). Bien souvent, chercher un algorithme dont la constante k est la plus petite possible pour résoudre un certain problème est un véritable défi.

Un problème de décision est un problème pour lequel la réponse est soit « oui », soit « non ». La classe de complexité NP (pour *non deterministic polynomial*) contient l'ensemble des problèmes de décision pour lesquels il existe un algorithme non déterministe qui permet de le résoudre en temps polynomial ; ou encore pour lesquels il existe un algorithme polynomial qui permet de vérifier une solution (certificat) au problème. Il est évident que $\mathrm{P} \subseteq \mathrm{NP}$, et la grande question ouverte en informatique est de savoir si $\mathrm{P} = \mathrm{NP}$ ou si $\mathrm{P} \neq \mathrm{NP}$.

1.1.2.3 Problèmes NP-complets et réductions polynomiales

La raison principale qui laisse penser que $\mathrm{P} \neq \mathrm{NP}$ est l'existence de problèmes NP-complets. Une propriété importante de cette famille de problèmes est que si l'on savait résoudre l'un d'eux en temps polynomial, alors on pourrait tous les résoudre en temps polynomial, et par conséquent on obtiendrait la relation $\mathrm{P} = \mathrm{NP}$. Jusqu'à présent, de nombreux problèmes ont été montrés NP-complets et pour aucun d'entre eux un algorithme polynomial n'a été trouvé.

La théorie de la NP-complétude fait appel aux réductions polynomiales pour montrer qu'un problème de décision est NP-complet. Soit deux problèmes Π_1 et Π_2, on dit que Π_1 se réduit en temps polynomial (ou se réduit polynomialement) à Π_2, noté $\Pi_1 \leq_p \Pi_2$, s'il existe une fonction f calculable en temps polynomial telle que toute instance i_1 de Π_1 admet une solution si et seulement si l'instance $f(i_1)$ admet une solution pour le problème Π_2. La fonction f réalise une transformation entre une instance de Π_1 et une instance de Π_2. C'est usuellement par l'entremise des réductions polynomiales que l'on montre qu'un problème est NP-complet. En particulier, on a la définition suivante :

Définition 1.1.1. Un problème Π est NP-complet si et seulement si :

1. $\Pi \in \mathrm{NP}$;
2. pour tout problème $\Pi' \in \mathrm{NP}$, $\Pi' \leq_p \Pi$.

La classe NP ne contient que des problèmes de décision. La plupart d'entre eux admettent une version d'optimisation qui est au moins aussi difficile à résoudre que le problème de décision correspondant. Un problème Π (pas nécessairement de décision) pour lequel tout problème $\Pi' \in \mathrm{NP}$ satisfait $\Pi' \leq_p \Pi$, est alors dit NP-difficile. Dans cette thèse,

nous considérons habituellement des problèmes d'optimisation et nous ne ferons pas de distinction entre la version d'optimisation et la version de décision, sauf en cas d'ambiguïté ou lorsqu'on souhaitera précisément s'intéresser à l'une des versions.

De manière pratique, pour démontrer la NP-complétude d'un problème, il n'est pas nécessaire de démontrer explicitement que pour tout $\Pi' \in \text{NP}$, $\Pi' \leq_p \Pi$. En effet, grâce aux réductions polynomiales et par la transitivité des réductions, on a le théorème suivant :

Théorème 1.1. *Un problème* Π *est NP-complet si et seulement si :*

1. $\Pi \in NP$ *;*
2. *pour un problème NP-complet* Π'*, on a la réduction* $\Pi' \leq_p \Pi$*.*

En effet, puisque le problème Π' est NP-complet alors, par définition, pour tout problème Π'' de NP on a $\Pi'' \leq_p \Pi'$. Comme la réduction polynomiale est transitive, il s'ensuit que $\Pi'' \leq_p \Pi$. Le théorème 1.1 permet donc de montrer commodément la NP-complétude d'un problème. Pour pouvoir l'utiliser il faut néanmoins connaître au moins un problème NP-complet. Cook [Coo71] a montré en 1971 que le problème SAT est NP-complet.

SAT

entrée : Une formule booléenne sur un ensemble de variables booléennes X.

question : Existe-t-il une affectation de valeurs de vérité pour X tel que la formule soit satisfaite ?

Depuis la démonstration de ce résultat, de nombreux problèmes ont été montrés NP-complets et, en 1979, Garey et Johnson en recensent déjà plus de 300 dans un ouvrage qui fait encore référence [GJ79]. À ce jour, aucun algorithme s'exécutant en temps polynomial pour résoudre un problème NP-complet n'a été trouvé. Ce sont ces problèmes qui ont motivés de larges travaux (algorithmes d'approximation, algorithmes à paramètres fixés, études de décompositions de graphes, ...) et qui motivent également les travaux de cette thèse.

1.2 Graphes et classes de graphes

1.2.1 Définitions et notations usuelles

Un *graphe* $G = (V, E)$ est un couple où V désigne un ensemble de *sommets* et E un ensemble d'*arêtes*. Nous ne considérerons par la suite que des graphes finis, c'est-à-dire des graphes pour lesquels les ensembles V et E sont finis. On note habituellement par $V(G)$ (respectivement $E(G)$) l'ensemble des sommets (respectivement des arêtes) d'un graphe

G, et l'on définit $n(G) = |V(G)|$ et $m(G) = |E(G)|$. S'il n'y a pas d'ambiguïté on utilisera simplement V, E, n et m en omettant de préciser le graphe considéré.

Etant donné un graphe $G = (V, E)$, on dit que deux sommets $u \in V$ et $v \in V$ sont *adjacents*, ou encore *voisins*, si l'arête $\{u, v\}$ appartient à E. Pour un sommet $v \in V$, on désigne par $N_G(v) = \{u \in V \ : \ \{u, v\} \in E\}$ le *voisinage (ouvert)* de v dans G. Le *voisinage fermé* est désigné par $N_G[v] = N_G(v) \cup \{v\}$. De même, pour un sous-ensemble $S \subseteq V$ de sommets, on définit $N_G(S) = \bigcup_{v \in S} N_G(v)$ et $N_G[S] = N_G(S) \cup S$.

Le *degré* d'un sommet v est défini par $d_G(v) = |N_G(v)|$. Un sommet est dit *isolé* s'il a pour degré 0, c'est-à-dire s'il n'a aucun voisin. Un graphe ayant n sommets est dit *complet* si chaque sommet a pour degré $n - 1$, c'est-à-dire si chaque sommet est voisin de tous les autres sommets du graphe.

S'il n'y a pas d'ambiguïté sur le graphe considéré, nous supprimerons l'indice G des notations $N_G(v)$, $N_G[v]$, $N_G(S)$, $N_G[S]$ et $d_G(v)$ pour écire plus simplement $N(v)$, $N[v]$, $N(S)$, $N[S]$ et $d(v)$.

On définit le *degré minimum* et le *degré maximum* de G par $\delta(G) = \min\{d(v) \ : \ v \in V\}$ et par $\Delta(G) = \max\{d(v) \ : \ v \in V\}$.

Un *chemin* dans un graphe $G = (V, E)$ est une séquence de sommets $(v_1, \ldots, v_k)$ telle que $\{v_i, v_{i+1}\} \in E$ pour $1 \leq i \leq k - 1$. La longueur d'un tel chemin est $k - 1$. Étant donné deux sommets u et v, la *distance* $d(u, v)$ entre ces deux sommets est la longueur d'un plus court chemin entre ceux-ci. On appelle *diamètre* d'un graphe $G = (V, E)$ la valeur $diam(G) = \max_{u,v \in V} d(u, v)$. L'*excentricité* d'un sommet u, notée par $exc(u)$, est donnée par $exc(u) = \max_{v \in V} d(u, v)$.

Un graphe $G = (V, E)$ est dit *connexe* si, pour toute paire de sommets $u, v \in V$, il existe un chemin dans le graphe joignant u à v.

On dit que $H = (V', E')$ est un *sous-graphe* de $G = (V, E)$ si $V' \subseteq V$ et $E' \subseteq E$. On appelle *composante connexe* de G un sous-graphe connexe maximal de G. La notion de *maximalité* sera définie à la section 1.5.1 (page 31). Etant donné un graphe $G = (V, E)$ et un sous-ensemble de sommets $S \subseteq V$, le graphe $G[S] = (S, \{\{u, v\} \in E \ : \ u, v \in S\})$ est appelé *sous-graphe* de G *induit* par S.

Le graphe *complémentaire* d'un graphe $G = (V, E)$, noté $\overline{G}$, est défini par $\overline{G} = (V, \{\{u, v\} \ : \ \{u, v\} \notin E\})$.

Un sous-ensemble $S \subseteq V$ de sommets qui induit un graphe complet (c'est-à-dire $G[S]$ est un graphe complet) est appelé une clique de G ; et un sous-ensemble $S \subseteq V$ tel que $\overline{G[S]}$ est un graphe complet est appelé un ensemble stable de G.

Un graphe *orienté* G est une paire (V, A) où V est un ensemble de sommets et A un ensemble d'arcs. Pour un arc $a = (u, v) \in A$, u est l'*origine* de l'arc orienté et v son *extrémité* ; on dit que l'arc va de u vers v. Dans la suite, lorsque nous utiliserons la notion d'arc orienté, la notation (u, v) désignera un arc. Lorsque nous considérerons des arêtes non orientées, la notation $\{u, v\}$ désignera une arête, ou plus simplement uv s'il n'y a pas d'ambiguïté.

Lors de l'introduction de la définition de composantes connexes, nous avons déjà défini la notion de chemin. On dit que $(v_1, \ldots, v_k)$ est un *cycle* de longueur k si $(v_1, \ldots, v_k)$ est un chemin de longueur au moins 3 et si l'arête $\{v_k, v_1\}$ appartient au graphe. Le cycle est dit *sans corde* si le graphe induit par l'ensemble de sommets $\{v_1, \ldots, v_k\}$ n'a pas d'autres arêtes que celles du cycle.

Pour un entier $l \in \mathbb{N}$, on note habituellement K_l le graphe complet ayant l sommets, P_l un chemin avec l sommets, et C_l un cycle contenant l sommets (voir la figure 1.1).

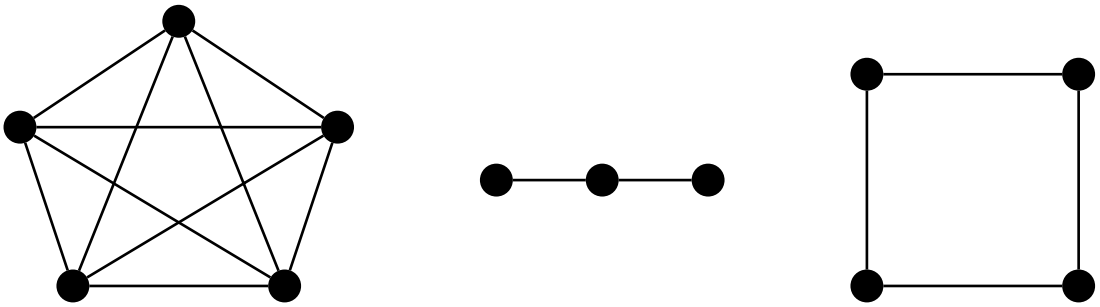


Fig. 1.1 – Un graphe possèdant trois composantes connexes : un K_5, un P_3 et un C_4.

Nous donnerons des définitions ou des propriétés supplémentaires au moment où elles seront nécessaires à la discussion d'un chapitre. Néanmoins, nous renvoyons le lecteur aux ouvrages de Diestel [Die05], de Golumbic [Gol80] ou celui de West [Wes96] pour davantage d'informations.

1.2.2 Quelques classes de graphes

Un ensemble de graphes peut être regroupé en familles ; on parle alors de *classe de graphes*. Bien souvent, ces classes contiennent des graphes ayant des propriétés communes. Une classe de graphes est dite *héréditaire* si pour chaque graphe de la classe, tous ses sous-graphes induits appartiennent aussi à la classe.

Un grand nombre de classes a été défini. De façon exhaustive, le site internet [ISGCI] en liste une très grande collection. Les motivations pour l'étude de ces différentes classes sont diverses mais en particulier l'analyse de la complexité de certains problèmes a abouti à la publication de nombreux articles. Par exemple, il est aujourd'hui bien connu qu'un simple algorithme glouton permet de résoudre le problème DOMINATION en temps linéaire sur les arbres [GJ79], alors que ce problème reste NP-complet sur les graphes pour lesquels tous les sommets ont pour degré au plus 3 [GJ79]. Dans les 16$^{\text{e}}$ colonnes sur la NP-complétude [Joh87], Johnson se consacre à l'étude des restrictions de graphes et leurs effets. On peut en particulier s'y référer pour connaître la complexité, sur diverses classes de graphes, de quelques problèmes NP-complets classiques dont ENSEMBLE STABLE, CLIQUE, NOMBRE CHROMATIQUE, CIRCUIT HAMILTONIEN, DOMINATION ou encore COUPE MAXIMUM. Quelques-uns de ces problèmes seront introduits à la section 1.3 (page 21).

Passons maintenant à quelques classes de graphes connues qui nous seront utiles par la suite. Pour une vue plus générale sur un large ensemble de classes de graphes, le livre de Brandstädt, Le et Spinrad [BLS99] servira de référence.

1.2.2.1 Graphes sans cycle

Un graphe sans cycle est appelé une *forêt*. Si de plus le graphe est connexe, alors on parle d'*arbre*.

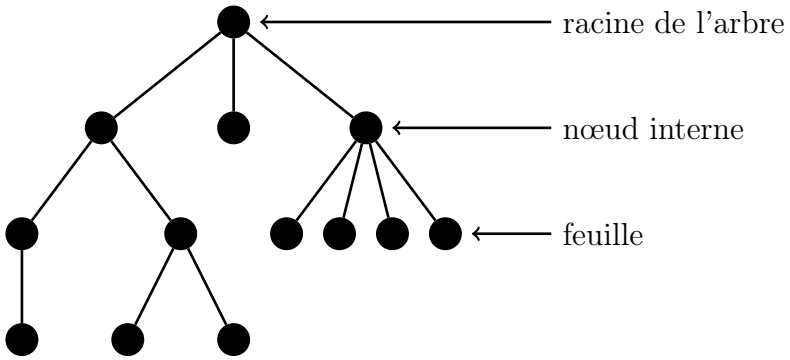


Fig. 1.2 – Un exemple d'arbre enraciné.

On appelle *nœuds* les sommets de l'arbre. L'arbre est dit *enraciné* s'il possède un nœud r désigné comme *racine* de l'arbre. On définit alors le *père* d'un nœud v comme le sommet p où $(r, \ldots, p, v)$ est l'unique chemin reliant la racine r au nœud v (avec éventuellement $p = r$). Par ailleurs, on dit également que v est un *fils* de p. Les nœuds sans fils sont appelés *feuilles* de l'arbre. Les *nœuds internes* désignent tous les nœuds de l'arbre, excepté les feuilles. La figure 1.2 illustre un arbre enraciné.

1.2.2.2 Graphes de degré borné

Un graphe $G = (V, E)$ de *degré borné par une constante* c vérifie la condition

$$\forall v \in V \ , \ d(v) \leq c$$

que l'on peut directement écrire sous la forme $\Delta(G) \leq c$.

En particulier, si les degrés d'un graphe sont bornés par 1, cela signifie que le graphe est une union disjointe de K_2 et de sommets isolés. De même, si $\Delta(G)$ est borné par 2 cela indique que le graphe est une union disjointe de cycles et de chemins (contenant éventuellement un unique sommet).

Si tous les sommets d'un graphe ont pour degré 3, le graphe est dit cubique.

Notons que de nombreux problèmes (comme Domination) peuvent être résolus en temps polynomial sur les graphes de degré borné par 2 alors qu'ils sont NP-complets pour les graphes dont le degré est borné par 3.

1.2.2.3 Graphes bipartis

Un graphe $G = (V, E)$ est dit *biparti* si l'ensemble de ses sommets peut être partitionné en deux sous-ensembles A et B qui sont chacun des ensembles stables de G. On dénote alors le graphe G par (A, B, E). La figure 1.3 donne un exemple d'un tel graphe. Si de plus l'ensemble E des arêtes est égal à $\{\{u, v\} \ : \ u \in A \text{ et } v \in B\}$ alors le graphe est dit *biparti complet*.

1.2.2.4 Graphes splits

Un graphe $G = (V, E)$ est *split* si l'ensemble de ses sommets peut être partitionné en deux sous-ensembles C et I tels que
- C est une clique de G ;
- I est un ensemble stable de G.

On dénote alors le graphe G par $G = (C, I, E)$ (voir la figure 1.3).

Fig. 1.3 – Exemple d'un graphe biparti (à gauche) et d'un graphe split (à droite).

1.2.2.5 Graphes cercles

Un graphe $G = (V, E)$ est un graphe *cercle* si et seulement si G est le graphe d'intersection de cordes dans un cercle. Autrement dit, G est un graphe *cercle* si :

1. il existe une bijection f entre l'ensemble des sommets de V et un ensemble de cordes dans un cercle ; et
2. pour tout couple de sommets $u, v \in V$, l'arête $\{u, v\} \in E$ si et seulement si les cordes $f(u)$ et $f(v)$ s'intersectent.

Ces graphes sont parfois également appelés *graphes de cordes*. La figure 1.4 donne un exemple de graphe de cercle et de son modèle d'intersection.

1.2.2.6 Graphes cordaux, 4-cordaux et faiblement cordaux

Un graphe est *cordal* s'il ne contient pas de cycle induit de longueur supérieure ou égale à 4. Un graphe est 4-*cordal* si la longueur de son plus long cycle sans corde est au plus égal à 4. Un graphe G est *faiblement cordal* si G et son complémentaire $\overline{G}$ sont 4-cordaux.

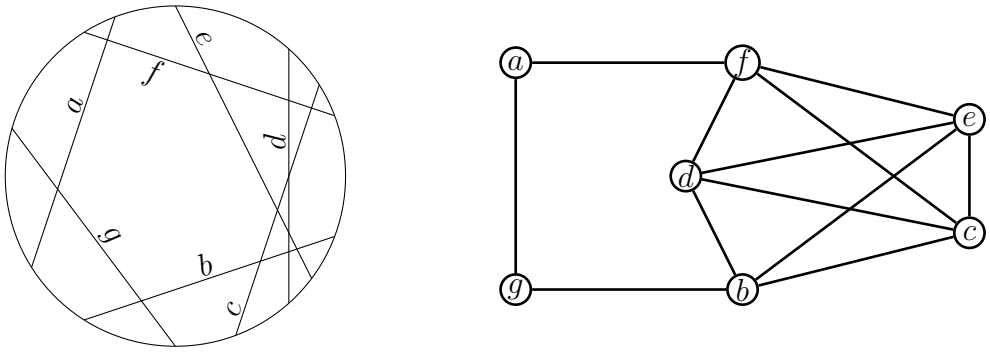


Fig. 1.4 – Exemple d'un graphe cercle et du modèle d'intersection de cordes correspondant.

La figure 1.5 donne quelques exemples de tels graphes. De façon générale, pour un entier k fixé, un graphe est dit k-cordal si la longueur du plus long cycle sans corde est au plus k.

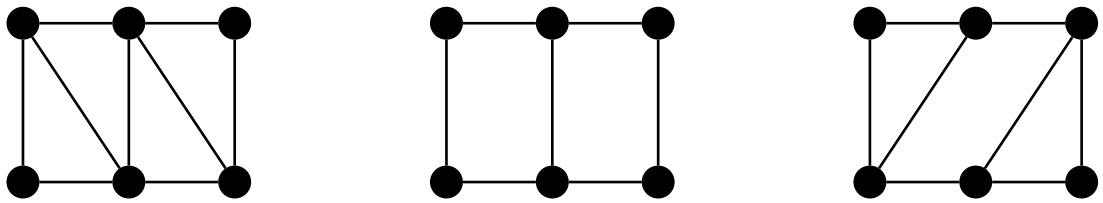


Fig. 1.5 – Exemple de graphe cordal, 4-cordal et faiblement cordal.

1.2.2.7 Autres classes de graphes

Nous listons et définissons ci-après quelques classes de graphes citées dans cette thèse. Nous référons le lecteur à [BLS99] pour davantage de précisions.

Graphes parfaits et fortement parfaits. Un graphe est dit *parfait* si pour tout sous-graphe induit, son nombre chromatique est égal à la taille de sa plus grande clique. Il est aujourd'hui bien connu que le complémentaire d'un graphe parfait est parfait.

Un graphe est *fortement parfait* si pour tout sous-graphe induit, il possède un ensemble stable qui intersecte toutes ses cliques maximales.

Graphes de comparabilité et de co-comparabilité. Un graphe $G = (V, E)$ est un graphe de *comparabilité* s'il existe une orientation F transitive et anti-symétrique des arêtes de G. Ainsi, pour toute arête $\{u, v\} \in E$, soit $(u, v) \in F$, soit $(v, u) \in F$; et si $(u, v) \in F$ et $(v, w) \in F$ alors $(u, w) \in F$.

Le graphe est dit de *co-comparabilité* si son complémentaire est de comparabilité.

Graphes sans astéroïde triple. Étant donné un graphe $G = (V, E)$, on dit que trois sommets $u, v, w \in V$ forment un *astéroïde triple* (abrégé par AT) si :

(i) $\{u, v, w\}$ est un ensemble stable ;
(ii) pour deux sommets quelconques parmi ces trois sommets, il existe toujours un chemin entre eux qui évite le voisinage du troisième sommet.

Les graphes *sans astéroïde triple* (sans AT) sont les graphes ne contenant aucun astéroïde triple.

Graphes planaires. Un graphe est *planaire* s'il existe un plongement de celui-ci dans le plan tel qu'aucune arête ne se croise.

Graphes d'intervalles et d'intervalles circulaires. Soit $\mathcal{I}$ une collection de n intervalles de la droite réelle et soit $G_{\mathcal{I}} = (V, E)$ son graphe d'intersection tel que :
(i) chaque intervalle $I_v \in \mathcal{I}$ est en bijection avec un sommet v de V ;
(ii) deux sommets $u \in V$ et $v \in V$ sont adjacents dans G si et seulement si les deux intervalles correspondants dans $\mathcal{I}$ ont une intersection non vide.

Un graphe est dit *d'intervalles* s'il est le graphe d'intersection d'une collection d'intervalles.

Les graphes *d'intervalles circulaires* sont définis de façon similaire, excepté que la collection d'intervalles de la droite réelle est maintenant une collection d'arcs de cercle.

Graphes de permutation. Soit $\pi = (\pi[1], \pi[2], \ldots, \pi[n])$ une permutation, de l'ensemble $V_n = \{1, 2, \ldots, n\}$. Le graphe de permutation $G_\pi = (V, E)$ associé à la permutation π est défini par $V = V_n$ et $\{u, v\} \in E$ si et seulement si $(u - v)(\pi^{-1}[u] - \pi^{-1}[v]) < 0$.

Un graphe G est dit de *permutation* s'il existe une permutation π telle que G soit isomorphe à G_π.

Cographes. Les *cographes* sont l'ensemble des graphes sans P_4 induit. Ils peuvent être construits en utilisant les règles suivantes :
(i) un sommet isolé est un cographe ;
(ii) le complémentaire d'un cographe est un cographe ;
(iii) étant donné deux cographes, leur union disjointe est un cographe

Graphes parfaitement ordonnés. Soit un graphe $G = (V, E)$ et un ordre linéaire, noté $\prec$, sur ses sommets. Un P_4 induit u, v, w, x tel que $u \prec v$ et $x \prec w$ est appelé une obstruction. Un graphe est dit *parfaitement ordonné* s'il existe un ordre linéaire de ses sommets ne contenant aucune obstruction.

Graphes en grille. Un graphe est dit en *grille* s'il est isomorphe au produit cartésien de deux chemins P_n et P_m.

Graphes de largeur arborescente bornée. La notion de largeur arborescente sera introduite au chapitre 4. L'ensemble des graphes pour lesquels la largeur arborescente d'un graphe peut être bornée par une constante sont les graphes de *largeur arborescente bornée*.

1.2.3 Quelques relations d'inclusion

Nous donnons ici quelques relations d'inclusion entre des classes de graphes usuelles. Les preuves de ces relations d'inclusions sont pour certaines faciles à voir (par exemple

un graphe cordal est aussi 4-cordal) ; d'autres, plus difficiles, peuvent se trouver dans la littérature (voir en particulier [BLS99]). La figure 1.6 présente quelques inclusions concernant des classes de graphes qui seront utilisées dans la suite. Les définitions des classes de graphes non données dans cette thèse sont présentées dans [BLS99]. En particulier, on a les inclusions fondamentales suivantes :

$$\begin{aligned}
\text{graphes bipartis} &\subset \text{graphes de comparabilité}\\
\text{graphes splits} &\subset \text{graphes cordaux}\\
\text{graphes d'intervalles} &\subset \text{graphes cordaux}\\
\text{graphes de permutation} &\subset \text{graphes cercles}\\
\text{graphes de permutation} &\subset \text{graphes de co-comparabilité}\\
\text{graphes de co-comparabilité} &\subset \text{graphes sans Astéroïde Triple}
\end{aligned}$$

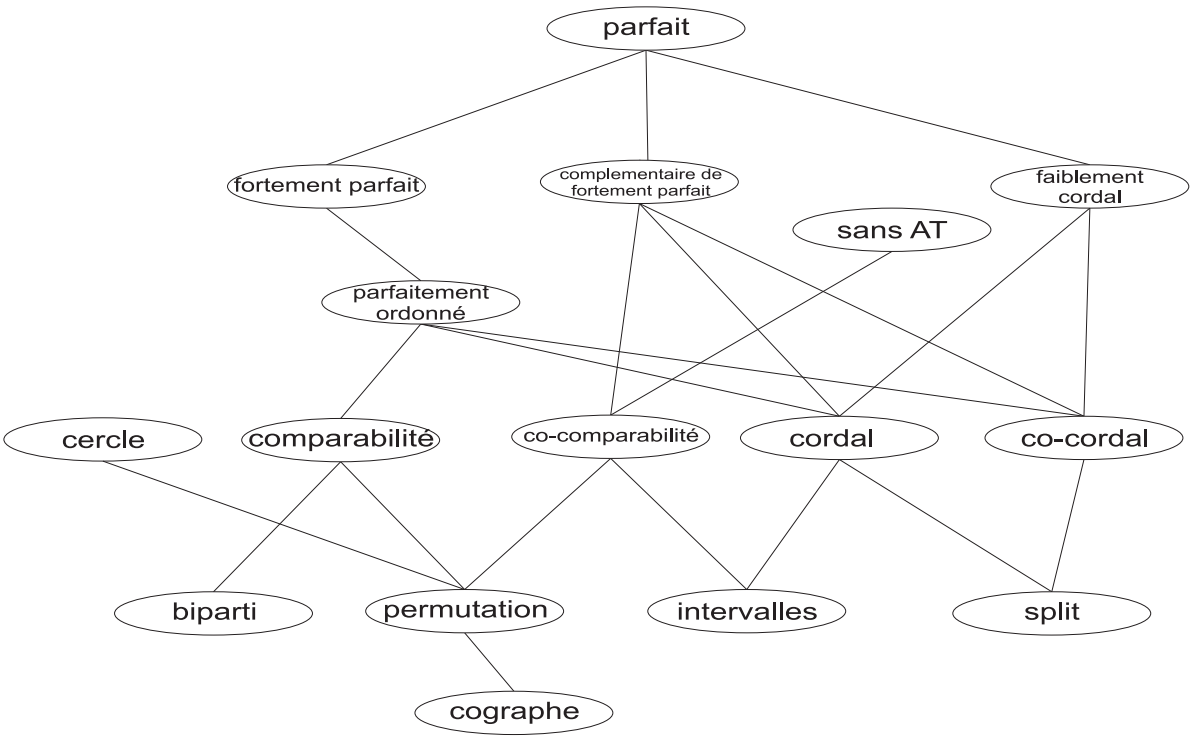


Fig. 1.6 – Quelques relations d'inclusion de classes de graphes.

1.3 Quelques problèmes de graphes

Il existe un grand nombre de problèmes sur les graphes. Certains sont « faciles » à résoudre (dans le sens où l'on connait un algorithme polynomial pour les résoudre) et d'autres sont « difficiles », typiquement NP-complets. Nous présentons maintenant quelques problèmes NP-complets fondamentaux et qui nous intéresseront dans le cadre de cette thèse.

1.3.1 Clique et Ensemble stable

Les problèmes de décision correspondant à ces deux problèmes sont les suivants :

Clique

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Existe-t-il $C \subseteq V$ tel que C est une clique de G de taille au moins k ?

Ensemble Stable (Independent Set)

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Existe-t-il $I \subseteq V$ tel que I est un ensemble stable de G de taille au moins k ?

Il est facile de voir qu'un graphe G a une clique de taille k si et seulement si son graphe complémentaire $\overline{G}$ a un ensemble stable de taille k. On note $\omega(G)$ la taille d'une plus grande clique de G, et $\alpha(G)$ la taille d'un plus grand ensemble stable. Les problèmes d'optimisation correspondant demandent de déterminer respectivement une clique et un ensemble stable de taille maximum du graphe G.

1.3.2 Coloration et partition en cliques

Un problème relié à Clique demande de trouver une partition des sommets d'un graphe en un nombre minimum de cliques. On note $\kappa(G)$ ce nombre minimum. Le problème de décision est donné par :

Partition en Cliques

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Est-il possible de partitionner V en au plus k cliques ? C'est-à-dire, existe-t-il au plus k cliques disjointes $C_1, C_2, \ldots, C_k$ dans G telles que $\cup_{i=1}^{k} C_i = V$?

Un autre problème bien connu est celui de la coloration d'un graphe :

Nombre Chromatique (Coloration)

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Est-il possible de colorier G avec au plus k couleurs ? C'est-à-dire, est-il possible de partitionner V en au plus k ensembles stables ?

L'objectif du problème est d'affecter une couleur à chaque sommet de sorte que deux sommets voisins ont toujours une couleur différente. Le problème d'optimisation correspondant demande de déterminer le plus petit nombre de couleurs nécessaires à la coloration d'un graphe G. Ce nombre est appelé *nombre chromatique* du graphe et est noté $\chi(G)$.

1.3.3 Domination et variantes

Le problème central que nous étudions dans cette thèse est le problème de la domination ainsi que quelques variantes et généralisations du problème. Voici la définition du problème Domination :

Domination (Dominating Set)

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Est-ce que G admet un ensemble dominant de taille au plus k ? C'est-à-dire, existe-t-il un sous-ensemble $D \subseteq V$ de cardinalité au plus k tel que pour tout $u \in V \setminus D$, le sommet u a au moins un voisin dans D ?

On note $\gamma(G)$ la taille d'un plus petit ensemble dominant de G. Le problème d'optimisation correspondant demande de déterminer un tel ensemble de cardinalité $\gamma(G)$. Le chapitre 3 donnera davantage d'informations sur ce problème central et en particulier on présentera les résultats connus pour une attaque par un algorithme de type exponentiel.

De nombreuses variantes du problème Domination ont été étudiées. Souvent le problème requiert des propriétés supplémentaires sur l'ensemble dominant ; pour en citer quelques-unes :

- l'ensemble dominant doit être connexe, c'est-à-dire que le sous-graphe induit par l'ensemble dominant doit être connexe ;
- il doit former un ensemble stable ;
- il doit être une clique ;
- il doit être total, c'est-à-dire que tout sommet du graphe (y compris un sommet appartenant à l'ensemble) doit avoir au moins un voisin dans l'ensemble.
- il doit être efficace, c.-à-d. chaque sommet qui n'appartient pas à l'ensemble doit avoir précisément un voisin dans l'ensemble.

Il existe aussi des variations du problème classique Domination qui autorisent un sommet à dominer plus que son voisinage et/ou attribuent des poids aux sommets. Par exemple, le problème de la domination romaine introduit dans [CDHH04] est défini de la façon suivante :

Domination Romaine

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Existe-t-il $V_1 \subseteq V$ et $V_2 \subseteq V$ tels que

1. chaque sommet $v \in V \setminus (V_1 \cup V_2)$ a au moins un voisin dans V_2 ; et
2. $|V_1| + 2|V_2| \leq k$?

Nous avons énuméré ici quelques variantes bien connues. D'autres problèmes autour de celui de la domination seront définis et feront l'objet d'études dans les prochains chapitres. Les livres de Haynes, Hedetniemi et Slater [HHS98a, HHS98b] offrent un panorama encore plus large puisque plus de 75 variantes y sont citées.

1.3.4 Le problème du voyageur de commerce

Le problème du voyageur de commerce est simple à formuler. Étant donné n villes, le coût du trajet entre deux villes quelconques, et une ville de départ, le problème demande de calculer un intinéraire qui visite chaque ville exactement une fois puis retourne à la ville d'origine de sorte à minimiser le coût total. Plus formellement, en utilisant des notations de graphes, le problème de décision est donné par :

Voyageur de Commerce (Travelling Salesman Problem)

entrée : Un graphe complet $G = (V, E)$, une fonction de pondération $d : E \to \mathbb{R}$ et un entier $k \in \mathbb{R}$.

question : Existe-t-il une séquence $(v_{i_1}, v_{i_2}, \ldots, v_{i_n})$ des n sommets, aussi appelée *tournée*, telle que $d(v_{i_1}, v_{i_n}) + \sum_{j=1}^{n-1} d(v_{i_j}, v_{i_{j+1}}) \leq k$?

L'objectif du problème d'optimisation est de minimiser la distance totale parcourue.

1.4 Attaquer un problème difficile

Lorsque l'on est confronté à un problème NP-complet il est impossible d'obtenir un algorithme qui s'exécute en temps polynomial pour le résoudre, sauf si P = NP. Néanmoins il existe plusieurs approches pour affronter ces problèmes. Pour cela on « relaxe » certaines conditions pour obtenir les attaques suivantes :

(i) considérer des graphes de taille modérée ;

(ii) considérer des graphes avec une *petite* solution ;

(iii) se restreindre à des classes de graphes ;

(iv) chercher une solution approchée ;

(v) concevoir des algorithmes randomisés ;

(vi) utiliser des heuristiques.

Les attaques (i), (ii) et (iii) relaxent la condition de résoudre le problème sur « n'importe quelle » entrée. Les attaques (iv) et (v) relaxent la condition de résoudre le problème « exactement » ; nous n'utilisons pas ces deux approches au cours de cette thèse et nous référons le lecteur aux ouvrages [ACGKMSP99, MR95, Vaz04] pour plus d'informations. L'attaque (vi) donne des algorithmes habituellement bons en pratique mais sans garantie sur leurs perfomances.

Dans cette thèse nous cherchons à résoudre exactement des problèmes NP-complets par des algorithmes dont le temps d'exécution est exponentiel. Au-delà de l'aspect algorithmique, important en soi, ces algorithmes trouveront une utilisation pratique au moins pour des entrées de taille modérée (et dépendant essentiellement de la base de l'exponentielle).

Dans la sous-section suivante, nous introduirons les algorithmes à paramètre fixé. Ce domaine de recherche est assez proche de celui des algorithmes exponentiels et c'est pour cela que nous en donnons une description.

1.4.1 Algorithmes à paramètre fixé

La théorie de la complexité à paramètre fixé s'applique à l'ensemble des problèmes pour lesquels l'instance d'entrée possède un paramètre. Par exemple, pour le problème de décision DOMINATION, l'entrée est constituée d'un graphe G et d'un entier k et le problème demande de déterminer si le graphe G admet un ensemble dominant de taille au plus k. Ce paramètre k est essentiel dans le domaine des algorithmes à paramètre fixé. Downey et Fellows ont constaté que ce paramètre peut contribuer de différentes façons à la complexité des problèmes [DF99]. L'objectif est de rendre les problèmes NP-complets *traitables* pour des paramètres fixés et petits. Plus formellement, un problème est *traitable à paramètre fixé* (en anglais FPT ou *fixed parameter tractable*), s'il peut être résolu en temps $f(k)|x|^c$ où c est une constante indépendante de k, $|x|$ représente la taille de l'entrée et la fonction calculable f est quelconque ; souvent $f(k)$ est une fonction exponentielle en k.

Prenons comme exemple le problème COUVERTURE DE SOMMETS défini par :

COUVERTURE DE SOMMETS (VERTEX COVER)

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Existe-t-il un ensemble $S \subseteq V$ avec $S \leq k$ et tel que pour toute arête de E, au moins l'une des extrémités appartient à S ?

On peut facilement résoudre ce problème de façon exacte en énumérant tous les sous-ensembles de sommets du graphe et en vérifiant si le sous-ensemble est bien une couverture de sommet de taille au plus k. On note alors que le nombre de sous-ensembles est 2^n, avec

$n = |V|$. Cet algorithme élémentaire n'est pas un algorithme à paramètre fixé puisque sa complexité en temps fait intervenir une fonction exponentielle qui dépend non pas du paramètre k mais de la taille de l'entrée.

Néanmoins, il existe pour ce problème un algorithme simple qui s'exécute en temps $\mathcal{O}(2^k|G|)$ dont nous allons donner une description. Soit un graphe $G = (V, E)$. L'idée de l'algorithme est de considérer une arête $\{u, v\}$ quelconque du graphe G. On note que toute couverture de sommet doit contenir au moins u ou au moins v. On construit alors un arbre binaire dont les deux fils de la racine sont étiquetés par u et par v. Dans la branche de l'arbre qui contient u, on construit le graphe $G_u = (V, E \setminus \{\{u, w\} : w \in N_G(u)\})$. On procède de façon similaire dans la branche contenant v : $G_v = (V, E \setminus \{\{v, w\} : w \in N_G(v)\})$. Ensuite, on réitère l'algorithme récursivement sur G_u et G_v jusqu'à ce que la hauteur de l'arbre ainsi construit soit égale à k ou que le graphe courant ne possède plus d'arête. Finalement, si, parmi tous les graphes correspondant à une feuille de l'arbre, l'un d'eux est sans arête, alors il existe bien une couverture de sommets de taille au plus k.

De manière informelle, on peut dire que l'idée des algorithmes à paramètre fixé est *de faire absorber l'intraitabilité des problèmes par le paramètre*. Néanmoins certains problèmes restent intraitables en utilisant cette approche. En effet, pour certains problèmes NP-difficiles on ne peut espèrer obtenir un algorithme à paramètre fixé, sauf si quelques relations, supposées très improbables, concernant des inclusions de classes de complexité, sont vérifiées. En particulier, pour le problème DOMINATION il n'existe pas d'algorithme à paramètre fixé, lorsque le paramètre est la taille de l'ensemble, sauf si la relation de complexité $FPT = W[2]$ est vérifiée. Pour des définitions précises sur les classes de complexité FPT, $W[1]$, $W[2]$, etc., nous référons également le lecteur aux livres [DF99, FG06, Nie06].

Dans l'espoir de mesurer la complexité des problèmes paramétrés, Downey et Fellows ont introduit une hiérarchie de complexité. Ils ont également établi des notions de réductibilité afin de montrer l'appartenance de problèmes à certaines classes de cette hiérarchie. Cette notion de réductibilité peut s'apparenter aux réductions polynomiales mis en œuvre pour montrer la NP-complétude de problèmes. La hiérarchie ainsi définie s'appelle la W-hiérarchie :

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[\mathrm{SAT}] \subseteq W[P].$$

Il est conjecturé que ces relations d'inclusions sont propres.

Pour décrire plus formellement ces classes, nous commençons par introduire les définitions qui suivent.

Une formule booléenne est dite 3-CNF si elle s'écrit sous la forme d'une conjonction de clauses, où chaque clause est une disjonction de trois littéraux. On considère une formule 3-CNF comme un circuit booléen composé d'une entrée pour chaque variable, d'une porte « et » (dénotée par $\wedge$), d'une porte « ou » (dénotée par $\vee$) pour chaque clause de la formule et d'une sortie unique. Une porte $\vee$ du circuit comporte exactement trois entrées (une

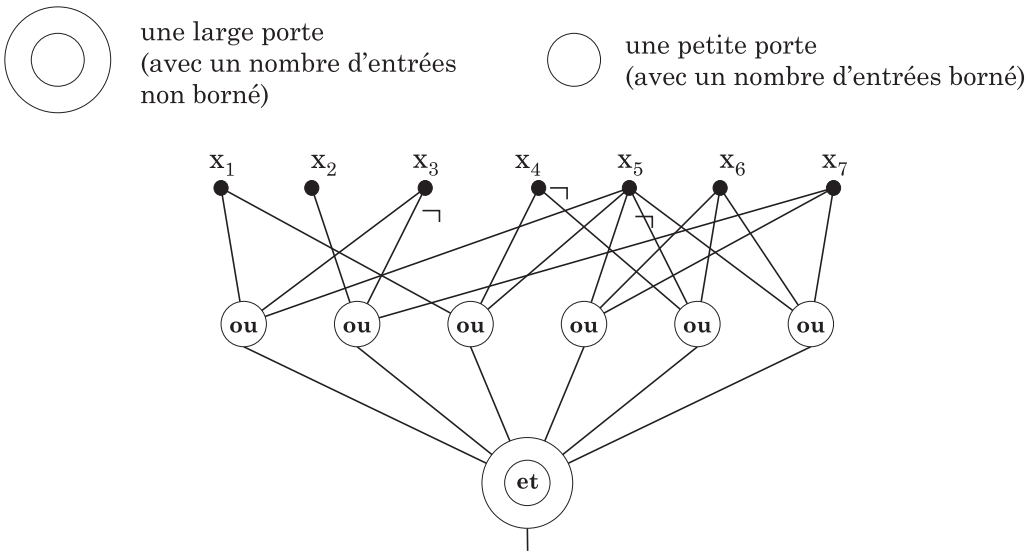


Fig. 1.7 – La formule 3-CNF $\varphi = (x_1 \vee x_3 \vee x_5) \wedge (x_2 \vee \neg x_3 \vee x_7) \wedge (x_1 \vee x_4 \vee x_5) \wedge (x_5 \vee x_6 \vee x_7) \wedge (\neg x_4 \vee \neg x_5 \vee x_6) \wedge (x_5 \vee x_6 \vee x_7)$ représentée par un circuit comportant une porte $\wedge$ et six portes $\vee$.

pour chacun des littéraux de la clause correspondante) et d'une sortie reliée à l'entrée de la porte $\wedge$. Une illustration est donnée à la figure 1.7.

De façon similaire, on peut définir une formule 4-CNF où chaque clause est la disjonction de quatre littéraux, ainsi que son circuit correspondant. Plus généralement, il est commode de considérer le modèle d'un *circuit de décision*. C'est un circuit avec une sortie unique constitué de portes dites « larges » et de portes dites « petites » sans restriction sur le nombre de sortie de ces portes. Les petites portes ont un nombre d'entrées borné (par quatre pour une formule 4-CNF), contrairement aux portes larges dont le nombre d'entrées excède cette borne pré-définie. Pour un tel circuit, on définit sa *profondeur* comme le nombre maximum de portes qu'il faut traverser pour un chemin allant depuis une variable d'entrée jusqu'à sa sortie. La *trame du circuit* est le nombre maximum de portes larges sur n'importe quel chemin allant des variables d'entrées à la sortie (voir la figure 1.8).

Soit $\mathcal{F} = \{C_1, \ldots, C_n, \ldots\}$ une famille de circuits de décision. On associe à $\mathcal{F}$ le langage paramétré suivant :

$$L_{\mathcal{F}} = \{\langle C_i, k\rangle \ : \ C_i \text{ a une affectation satisfaisante de poids } k\}$$

(Une affectation satisfaisante de poids k est une affectation des entrées du circuit tel que la sortie soit évaluée à vrai, et dont le nombre d'entrées évaluées à vrai est exactement k.)

Par exemple, si $\mathcal{F}$ est la famille de circuits booléens correspondant aux formules propositionnelles en forme 3-CNF, alors $L_{\mathcal{F}}$ correspond au langage qui représente le problème

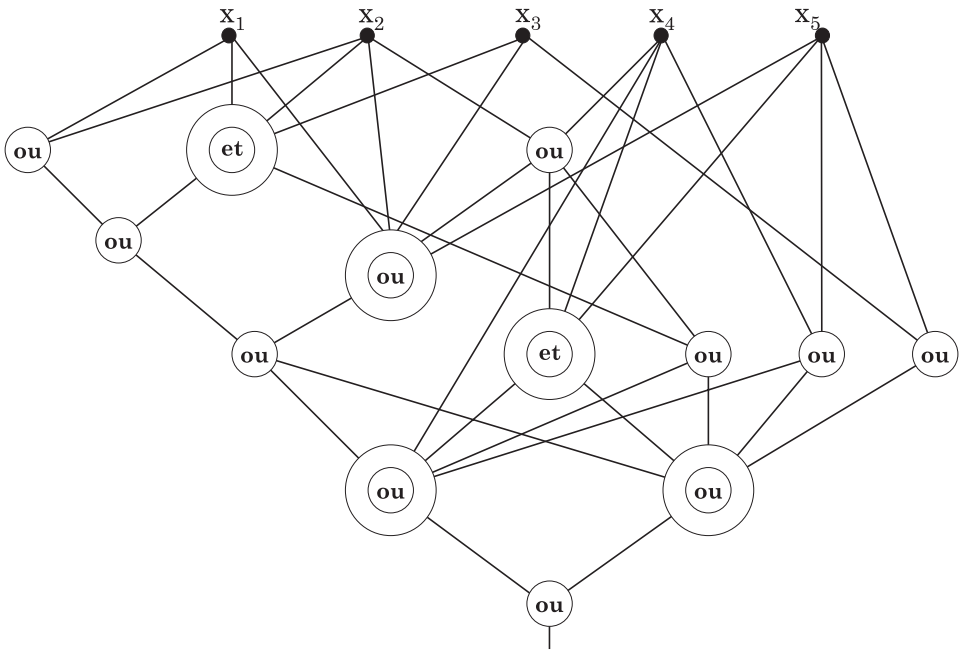


Fig. 1.8 – Un circuit de décision de trame 2 et de profondeur 5.

3-CNF Satisfaisabilité pondérée suivant :

3-CNF Satisfaisabilité pondérée

entrée : Une formule 3-CNF F (c.-à-d. une formule dont chaque clause ne possède pas plus de 3 littéraux) et un entier $k \in \mathbb{N}$ (le paramètre).

question : Existe-t-il une affectation satisfaisante pour F de poids exactement k ?

Une généralisation de la classe des circuits correspondant aux formules 3-CNF est donnée par :

Circuit Satisfaisable pondéré de Trame t et de Profondeur h

entrée : Un circuit de décision C ayant pour trame t et profondeur h, et un entier $k \in \mathbb{N}$ (le paramètre).

question : Existe-t-il une affectation satisfaisante de poids k pour C ?

Par convention, on désigne par $L_{\mathcal{F}(t,h)}$ le langage paramétré associé à la famille de circuits de décision dont la trame est t et la profondeur h.

Définition 1.4.1 (réduction paramétrée)**.** On dit que L se réduit à L' par une m-réduction paramétrée standard s'il existe deux fonctions f et g de $\mathbb{N}$ dans $\mathbb{N}$, et une fonction h de $\Sigma^* \times \mathbb{N}$ dans Σ^*, telle que $h(x,k)$ soit calculable en temps $g(k)|x|^c$, pour une certaine constante $c \geq 0$, et $\langle x,k\rangle \in L$ si et seulement si $\langle h(x,k), f(k)\rangle \in L'$.

Définition 1.4.2 ($W[t]$)**.** On définit un langage L comme étant dans la classe $W[t]$ si et seulement si L est paramètre fixé réductible à $L_{\mathcal{F}(t,h)}$ pour un certain h.

Tout comme pour le célèbre théorème de Cook montrant la NP-complétude du problème SAT, il existe un analogue pour la $W[1]$-complétude qui affirme que le problème n-CNF Satisfaisabilité pondérée est $W[1]$-complet pour n'importe quel $n \geq 2$ fixé [DF99]. Le problème, plus général, CNF Satisfaisabilité pondérée n'impose aucune restriction sur la taille des clauses. On a alors le théorème suivant :

Théorème 1.2 ([DF95a])**.** CNF Satisfaisabilité pondérée *est* $W[2]$ *complet.*

Il est possible de montrer qu'étant donné une formule en forme normale conjonctive et un entier k, un graphe G peut être construit de sorte que la formule a une affectation satisfaisante de poids k si et seulement si le graphe G a un ensemble dominant de taille $2k$. Le théorème suivant peut alors être établi :

Théorème 1.3 ([DF99])**.** Domination *est* $W[2]$ *complet si l'on considère la taille de la solution comme paramètre.*

La conséquence immédiate est que l'on ne peut pas espérer obtenir un algorithme à paramètre fixé pour le problème Domination puisque le problème est complet pour $W[2]$, lorsque le paramètre considéré est la taille de l'ensemble, sauf si $W[2] = \text{FPT}$.

Le problème Domination admet pour dual le problème Ensemble non bloquant défini ci-après. Dans [DFFPR06] et dans [Roo06] sont donnés deux algorithmes à paramètre fixé pour en calculer une solution.

Ensemble non bloquant

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Existe-t-il un sous-ensemble $N \subseteq V$ de taille au moins k tel que pour tout $v \in N$, v a au moins un voisin dans $V \setminus N$?

Finalement, si on dispose d'un algorithme FPT pour un problème d'optimisation, comme par exemple pour le problème de la couverture de sommets, alors on peut utiliser une recherche dichotomique pour déterminer une solution optimale.

Pour un traitement plus précis de ce vaste domaine de recherche, les livres [DF99, Nie06, FG06] font référence.

1.4.2 Algorithmes exponentiels

Si l'on désire résoudre un problème NP-complet de façon exacte, il n'est pas toujours possible d'utiliser l'une des approches précédentes, telles que les algorithmes d'approximations ou les algorithmes à paramètres fixés. De plus, même pour ces deux approches, il existe aussi des résultats *négatifs* comme l'attestent les théorèmes suivants :

Théorème 1.4 ([BH92])**.** *On ne peut pas approximer le problème* Clique *avec un facteur constant, sauf si $P = NP$; néanmoins le problème est $\mathcal{O}(|V|/\log^2|V|)$-approximable.*

Théorème 1.5 ([DF95b])**.** *Il est impossible d'obtenir un algorithme à paramètre fixé pour le problème* Clique*, sauf si $W[1] = FPT$. Le problème* Clique *est $W[1]$-complet.*

Dès lors, une approche qui reste possible est de résoudre le problème de façon exhaustive. Cette attaque brute, triviale mais simple, permet effectivement de trouver une solution exacte. Souvent cette approche n'a d'intérêt que si l'entrée considérée n'est pas trop grande puisque le temps d'exécution croît exponentiellement ou sous-exponentiellement en fonction de la taille de l'entrée. De toute manière, sous l'hypothèse P $\neq$ NP, on ne peut pas espérer obtenir un algorithme polynomial et seul un algorithme exponentiel ou sous-exponentiel pourrait résoudre exactement le problème.

Définition 1.4.3 ([Woe03])**.** Un problème peut être résolu en temps *sous-exponentiel*, s'il existe un algorithme qui le résout en temps $f(n)$ où $f(n) = \mathcal{O}(c^n)$ pour toute constante $c > 1$; autrement dit, si le logarithme de $f(n)$ dépend sous-linéairement de n.
La classe de complexité SUBEXP est l'ensemble des problèmes pour lesquels il existe un algorithme sous-expentiel qui les résout.

Exemple. *Un temps d'exécution en $\mathcal{O}^*(2^{5\sqrt{n}})$ est dit sous-exponentiel puisque* $\lim_{n\to\infty} \frac{\log(2^{5\sqrt{n}})}{n} = 0$.

Si on note P, SUBEXP et EXP l'ensemble des problèmes pour lesquels il existe respectivement un algorithme polynomial, sous-exponentiel et exponentiel pour les résoudre, on a la relation d'inclusion :

$$\text{P} \subseteq \text{SUBEXP} \subseteq \text{EXP}.$$

L'approche que nous avons entreprise dans cette thèse est d'obtenir de *bons* algorithmes exponentiels pour ces problèmes NP-complets. Précisément, nous cherchons à obtenir des algorithmes pour résoudre exactement de tels problèmes et dont le temps d'exécution est significativement meilleur que celui de l'algorithme trivial réalisant cette tâche par une recherche exhaustive naïve. Ce nouvel angle d'attaque a depuis quelques années motivé un certain nombre de chercheurs à concevoir et aussi à analyser de tels algorithmes. Il est manifeste qu'il est préférable d'avoir un algorithme dont le temps d'exécution est en $\mathcal{O}(1.1^n)$ qu'un algorithme en $\mathcal{O}(2^n)$ pour résoudre un certain problème. Dans un certain sens, on peut aussi se demander s'il n'est pas mieux d'avoir un algorithme en $\mathcal{O}(1.1^n)$ qu'un

algorithme en $\mathcal{O}(n^5)$; cela peut être vrai au moins pour des instances de taille modérée et en supposant que les facteurs non donnés dans la notation $\mathcal{O}$ sont du même ordre de grandeur. Néanmoins, notre objectif premier est de s'attacher à résoudre de façon exacte des problèmes des problèmes NP-complets, problèmes pour lesquels un algorithme polynomial est impossible, sauf si P = NP.

Puisque cette attaque est centrale dans ce travail de thèse, le chapitre 2 lui est consacré. Les chapitres suivants présenteront des applications à des problèmes de domination de graphes ainsi qu'à des variantes et généralisations de ce même problème.

1.5 Éléments de combinatoire

Au long de cette thèse, nous faisons parfois appel à des notions de combinatoire. Cette section se propose de rappeler quelques formules essentielles qui serviront à la fois à représenter des objets – tels que l'ensemble des sous-ensembles d'un ensemble – qu'à analyser le temps d'exécution de certains algorithmes – par exemple pour déterminer le temps nécessaire pour énumérer des sous-ensembles respectant une certaine propriété. Pour compléter cette section, nous pourrons nous référer à des ouvrages tels que [And03, Cam94, GKP03, Rio58, Ros06].

1.5.1 Ensembles et sous-ensembles

Un *ensemble* E est une collection d'objets appelés *éléments* de l'ensemble. Etant donné un élément e de E, on dit que e *appartient* à l'ensemble E, noté $e \in E$. Le nombre d'éléments dans E est appelé la *cardinalité* de l'ensemble et on la note $\mathrm{Card}(E) = |E|$. L'ensemble vide, dénoté par $\emptyset$, est l'ensemble ne contenant aucun élément ; sa cardinalité est donc égale à zéro.

Soit E' un ensemble tel que chaque élément de E' soit aussi un élément d'un ensemble E. On dit que E' est un *sous-ensemble* de E. L'ensemble vide est également un sous-ensemble de E.

L'ensemble des sous-ensembles d'un ensemble E, noté $\mathcal{P}(E)$, est aussi appelé ensemble des parties de E. On note également $\mathcal{P}(E)$ par 2^E. Pour un ensemble E de cardinalité finie, on a $\mathrm{Card}(\mathcal{P}(E)) = 2^{\mathrm{Card}(E)}$.

Soit Π une propriété et E un ensemble d'éléments. On dit que le sous-ensemble $E' \subseteq E$ est *maximal* (par inclusion) par rapport à la propriété Π, s'il respecte Π et si pour tout sous-ensemble $E'' \subseteq E$ tel que $E' \subset E''$, l'ensemble E'' ne respecte pas la propriété Π. Parmi tous les sous-ensembles maximaux, un sous-ensemble de plus grande taille possible est dit *maximum*. De même, $E' \subseteq E$ est *minimal* (par inclusion) par rapport à Π si E' respecte P et si pour tout sous-ensemble $E'' \subseteq E$ tel que $E'' \subset E'$, l'ensemble E'' ne respecte pas la propriété Π. Un ensemble minimal de plus petite cardinalité est dit *minimum*.

1.5.2 Partition d'un ensemble

Etant donné un ensemble E et un entier positif k, une *partition* en k sous-ensembles est une collection $\{E_1, E_2, \ldots, E_k\}$ de k sous-ensembles de E qui vérifie :

1. $\bigcup_{1 \leq i \leq k} E_i = E$;
2. pour tout $i, j \in \{1, \ldots, k\}$, $i \neq j$, $E_i \cap E_j = \emptyset$.

1.5.3 Permutations

Soit $E = \{e_1, e_2, \ldots, e_n\}$ un ensemble à n éléments. Une *permutation* de n éléments est un ordonnancement des objets de E.

Exemple. *Soit un ensemble* $S = \{s_1, s_2, s_3\}$*, alors il existe* 6 *façons d'ordonnancer ses* 3 *éléments :* (s_1, s_2, s_3)*,* (s_1, s_3, s_2)*,* (s_2, s_1, s_3)*,* (s_2, s_3, s_1)*,* (s_3, s_1, s_2) *et* (s_3, s_2, s_1)*.*

Il est commode de représenter un tel ordonnancement par une bijection de $\{1, 2, \ldots, n\}$ dans E.

Le nombre de permutations d'un ensemble contenant n éléments est noté $n!$. Par convention on a $0! = 1$. La valeur de l'expression $n!$ est donnée par la fonction factorielle définie par :

$$n! = \prod_{i=1}^{n} i = n \cdot (n-1) \cdot (n-2) \cdot \cdots \cdot 1.$$

La *formule de Stirling* donne une approximation de la fonction factorielle au voisinage de l'infini :

$$n! \approx \sqrt{2\pi n}(\frac{n}{e})^n.$$

1.5.4 Combinaisons

Soit $E = \{e_1, e_2, \ldots, e_n\}$ un ensemble à n éléments et soit un entier positif $k \leq n$. Une *combinaison* de k éléments parmi E est une sélection de k éléments sans tenir compte de leur ordre.

Exemple. *Soit un ensemble* $S = \{s_1, s_2, s_3\}$*, alors il existe* 3 *combinaisons de* 2 *éléments parmi* S *:* $\{s_1, s_2\}$*,* $\{s_1, s_3\}$*,* $\{s_2, s_3\}$

Le nombre de combinaisons de k éléments parmi un ensemble de cardinalité n est noté $\binom{n}{k}$ et on a la relation :

$$\binom{n}{k} = \frac{n \cdot (n-1) \cdot (n-2) \cdot \cdots \cdot (n-k+1)}{k!} = \frac{n!}{k!(n-k)!}.$$

On note que le nombre de combinaisons de k éléments parmi n est précisément le nombre de sous-ensembles de cardinalité k d'un ensemble à n éléments. En particulier, on

a $\sum_{k=0}^{n} \binom{n}{k} = 2^n$ où 2^n est le nombre de sous-ensembles de cardinalité quelconque d'un ensemble à n éléments.

Comme conséquence de la relation donnée précédemment, on en déduit immédiatement que $\binom{n}{k} = \binom{n}{n-k}$.

En utilisant l'approximation de Stirling de la section 1.5.3, il est possible de borner asymptotiquement le nombre de sous-ensembles de taille αn, $0 \leq \alpha \leq 1$. Cette borne supérieure asymptotique est particulièrement utile pour l'analyse du temps d'exécution des algorithmes nécessitant l'énumération de tels sous-ensembles.

$$\binom{n}{\alpha n} = \frac{n!}{(\alpha n)!(n-\alpha n)!} = \mathcal{O}\Big(\frac{(n/e)^n}{(\alpha n/e)^{\alpha n}(n(1-\alpha)/e)^{n(1-\alpha)}}\Big) = \mathcal{O}\Big(\Big(\frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}}\Big)^n\Big)$$

Enfin, la relation suivante est connue sous le nom de *formule du binôme* :

$$\sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k} = (a+b)^n.$$

Chapitre 2

Algorithmes exponentiels

Résoudre un problème NP-complet exactement et efficacement est l'un des défis les plus importants des quarante dernières années en informatique. Un algorithme polynomial pour résoudre un tel problème établirait immédiatement la relation P = NP. Malheureusement, à ce jour il n'existe aucun algorithme permettant d'accomplir cette tâche. De plus, de nombreux chercheurs conjecturent que P $\neq$ NP et donc qu'un tel algorithme est fortement improbable à obtenir. C'est pourquoi l'étude des algorithmes exponentiels est devenue un domaine de recherche conséquent où aussi bien la conception que l'analyse du temps d'exécution demandent d'importants efforts pour obtenir des algorithmes meilleurs que les simples algorithmes naïfs.

2.1 Motivations

Lorsqu'on souhaite résoudre exactement un problème NP-complet sur n'importe quelle instance, il n'est pas toujours possible d'utiliser les approches suivantes que nous avons précédemment listées à la section 1.4 :

(i) considérer des instances ayant une structure particulière ;
(ii) considérer des instances avec une *petite* solution ;
(iii) chercher une solution approchée ;
(iv) concevoir des algorithmes randomisés ;
(v) utiliser des heuristiques.

Par exemple, il existe des résultats « négatifs » comme le stipulent les théorèmes suivants :

Théorème 2.1 ([RS97, Fei98])**.** *Il n'existe pas d'algorithme calculant en temps polynomial une solution approchée du problème* Domination *avec un facteur constant, sauf si la relation $P = NP$ est vérifiée.*

Théorème 2.2 ([DF99])**.** *Le problème* Domination *n'admet pas d'algorithme à paramètre fixé, sauf si $W[2] = FPT$.*

Dès lors, une approche possible pour résoudre un problème de façon exacte est de le résoudre de manière exhaustive. On constate que cette attaque un peu brute et triviale fonctionne bien en pratique si l'entrée considérée n'est pas trop grande. La raison est que le temps d'exécution nécessaire à un algorithme pour résoudre un problème de façon exhaustive est exponentiel. Sous l'hypothèse P $\neq$ NP, on ne peut de toute façon pas espérer obtenir un algorithme polynomial. Néanmoins l'approche que nous avons entreprise dans cette thèse est d'obtenir un *bon* algorithme exponentiel ; nous cherchons à obtenir un algorithme pour résoudre exactement un problème dont le temps d'exécution au pire des cas est significativement meilleur que celui de l'algorithme naïf réalisant cette tâche par recherche exhaustive. Ce nouvel angle d'attaque a depuis quelques années motivé des chercheurs du domaine à concevoir, mais aussi à analyser, de tels algorithmes. Comme déjà indiqué, il est préférable d'avoir un algorithme dont le temps d'exécution est en $\mathcal{O}(1.1^n)$ qu'un algorithme en $\mathcal{O}(2^n)$ pour résoudre un certain problème. Bien sûr le temps d'exécution reste exponentiel en la taille de l'entrée, mais rappelons que notre objectif principal est de résoudre des problèmes pour lesquels un algorithme polynomial est improbable, sauf si P = NP.

Le tableau 2.1 donne une indication sur le temps d'exécution nécessaire à un algorithme en fonction de sa complexité et de la taille de son entrée. Ces temps ont été estimés en considérant comme modèle de calcul une machine capable d'effectuer dans l'ordre de 10^9 opérations par seconde. Les ordinateurs proposés aujourd'hui au grand public sont capables d'effectuer un tel nombre d'opérations : un processeur cadencé à 3GHz effectue trois milliards d'opérations par seconde.

Notons que bien qu'un algorithme polynomial dont le temps d'exécution est $\mathcal{O}(n^{10})$ soit assez rare en algorithmique, un tel algorithme est assez peu utile en pratique : pour traiter une entrée de taille 30 plusieurs jours sont nécessaires. On observe qu'un algorithme en 2^n est plus commode puisque pour une même entrée de taille 30, seulement quelques secondes sont nécessaires. Bien sûr un algorithme en 2^n voit son temps d'exécution croître de façon exponentielle et très vite il devient inutilisable en pratique : une entrée de taille 60 demande déjà plusieurs années d'exécution.

Un espoir est alors d'utiliser un ordinateur plus rapide. La célèbre « loi de Gordon Moore » indique que la puissance des ordinateurs devrait doubler tous les 18 mois. Et donc l'intuition serait qu'avec une machine plus performante, une plus grande entrée pourrait être traitée. Cela est vrai mais si l'on dispose d'un algorithme exponentiel, le gain apporté par un ordinateur deux fois plus rapide ne permet pas de traiter une entrée beaucoup plus grande comme l'indique la table 2.2. Si on dispose d'un algorithme en 2^n qui en une certaine unité de temps t permet de traiter une entrée de taille N_E, alors un ordinateur cent fois plus rapide ne permet que de traiter une entrée de taille $N_E + 6.64$. On ne gagne qu'un facteur additif ! Un algorithme polynomial permet quant à lui de gagner un facteur multiplicatif.

L'espoir fondé sur une augmentation de la puissance des machines est donc vain si l'on s'appuie sur la « loi de Moore » ne prévoyant pas plus qu'un doublement de la puissance. En réalité, nous sommes même un peu en dessous des prévisions de Moore puisque la puissance semble doubler par tranche de deux années.

Tab. 2.1 – Temps d'exécution en fonction de la complexité en temps d'un algorithme et de la taille de son entrée sur une machine effectuant 10^9 opérations par seconde. Un temps dit *instantané* signifie que le temps d'exécution requis est inférieur à la seconde.

complexité \ taille	20	30	40	50	60
n^2	instantané	instantané	instantané	instantané	instantané
n^5	instantané	instantané	instantané	instantané	instantané
n^{10}	$\approx$ 3 heures	$\approx$ 7 jours	$\approx$ 122 jours	$\approx$ 3 années	$\approx$ 19 années
1.2^n	instantané	instantané	instantané	instantané	instantané
1.3^n	instantané	instantané	instantané	instantané	instantané
1.4^n	instantané	instantané	instantané	instantané	$\approx$ 1 seconde
1.7^n	instantané	instantané	$\approx$ 2 secondes	$\approx$ 6 minutes	$\approx$ 19 heures
2^n	instantané	$\approx$ 1 seconde	$\approx$ 18 minutes	$\approx$ 13 jours	$\approx$ 37 années
3^n	$\approx$ 3 secondes	$\approx$ 2 jours	$\approx$ 386 années	$\approx 2 \cdot 10^7$ années	$\approx 10^{12}$ années

complexité \ taille	80	100	120
n^2	instantané	instantané	instantané
n^5	$\approx$ 3 secondes	$\approx$ 10 secondes	$\approx$ 25 secondes
n^{10}	$\approx$ 340 années	$\approx$ 3170 années	$\approx$ 19634 années
1.2^n	$\approx$ instantané	$\approx$ instantané	$\approx$ 3 secondes
1.3^n	$\approx$ 1 seconde	$\approx$ 4 minutes	$\approx$ 13 heures
1.4^n	$\approx$ 8 minutes	$\approx$ 5 jours	$\approx$ 11 années
1.7^n	$\approx$ 86 années	$\approx 3 \cdot 10^6$ années	$\approx 10^{11}$ années
2^n	$\approx 4 \cdot 10^7$ années	$\approx 4 \cdot 10^{14}$ années	$\approx 4 \cdot 10^{19}$ années
3^n	$\approx 4 \cdot 10^{21}$ années	$\approx 2 \cdot 10^{31}$ années	$\approx 6 \cdot 10^{40}$ années

Tab. 2.2 – Taille de l'entrée traitable en une unité de temps.

	ordinateur actuel	100 × plus rapide	1000 × plus rapide
n^2	N_A	$10N_A$	$31.6N_A$
n^5	N_B	$2.5N_B$	$3.98N_B$
n^{10}	N_C	$1.58N_C$	$1.99N_C$
1.41^n	N_D	$13.4 + N_D$	$20.1 + N_D$
2^n	N_E	$6.64 + N_E$	$9.96 + N_E$
3^n	N_F	$4.19 + N_F$	$6.28 + N_F$

Regardons de plus près la complexité des algorithmes. Supposons qu'au lieu d'avoir un algorithme en 2^n pour résoudre un certain problème, on dispose d'un algorithme en $\sqrt{2}^n \approx 1.41^n$ pour le résoudre. Alors, comme l'atteste la table 2.1, si avec l'algorithme en 2^n on était capable de traiter une entrée de taille N en une certaine unité de temps, alors avec l'algorithme en 1.41^n, on peut traiter une entrée de taille $2N$ dans cette même unité de temps. Il est donc particulièrement important de concentrer nos efforts sur l'obtention de *bons* algorithmes exponentiels, c'est-à-dire des algorithmes pour lesquels le temps d'exécution, bien qu'exponentiel, soit le meilleur possible. Pour cela il faut concentrer notre attention sur la base de l'exponentielle apparaissant dans le temps d'exécution. On note qu'une réduction de la base de l'exponentielle permet d'augmenter la taille de l'entrée que l'on peut résoudre en une certaine unité de temps par un facteur multiplication.

Dans la prochaine section, nous témoignons que cette idée a suscité d'abondantes publications depuis une trentaine d'années en donnant quelques références à des résultats importants et connus du domaine de recherche.

2.2 Principaux résultats connus

L'intérêt pour les algorithmes exponentiels date depuis les années soixante et soixante-dix. L'article de Held et Karp de 1962 [HK62] atteste déjà de la volonté d'obtenir des algorithmes exacts pour des problèmes difficiles. Au cours de ces dix dernières années cet intérêt ainsi que le nombre de publications dans le domaine n'a cessé de croître. Pour preuve, nous pouvons mentionner les cinq *états de l'art* [FGK05b, Iwa04, Sch05, Woe03, Woe04] qui ont été publiés et présentés à des conférences depuis 2003. Encore très récemment en septembre 2006, à Zürich, lors de la conférence ALGO 2006, Schöning a fait un exposé

invité intitulé *Moderately exponential algorithms* ; et c'est sous ce titre qu'une session de travail d'une semaine sera organisée au centre international de conférences et de recherches pour l'informatique (*International Conference and Research Center for Computer Science*, Dagstuhl, Allemagne) au cours de l'année 2008.

De façon historique, plusieurs problèmes réputés difficiles ont suscité un engouement depuis les années soixante et soixante-dix afin de proposer des algorithmes exponentiels pour les résoudre. Parmi eux, le problème SAT a fait l'objet d'intenses recherches par une communauté consacrée à l'étude de celui-ci. Dans [DP60, DLL62] sont données les premières approches visant à proposer un algorithme exponentiel.

Un peu plus tard, les problèmes Ensemble Stable et Nombre Chromatique ont eux aussi donné lieu à la publication de leurs premiers algorithmes exponentiels (voir [MM65, TT77] et [Chr71, Law76]). Ce n'est que très récemment, à la conférence FOCS 2006, qu'un algorithme s'exécutant en $\mathcal{O}^*(2^n)$ pour déterminer le nombre chromatique d'un graphe a été donné [BH06, Koi06].

De manière tout aussi attrayante, le problème Domination a fait naître depuis 2004 plusieurs articles. En moins de deux ans, pas moins de quatre algorithmes ont été proposé. Ce problème étant d'un grand intérêt au long de cette thèse, nous consacrerons le chapitre 3 à les présenter.

Avant de lister à la section 2.2.2 quelques-uns des principaux résultats connus, nous proposons de retracer un historique du problème Ensemble Stable.

2.2.1 Le problème Ensemble Stable

Le problème Ensemble Stable est un problème étudié depuis longtemps. En 1965, Moon et Moser ont montré dans [MM65] que le nombre de cliques maximales par inclusion est borné supérieurement par $3^{n/3} \approx 1.4423^n$. (La notion d'ensembles maximaux est rappelée à la section 1.5.1.) Cette borne est aussi valable pour le nombre d'ensembles stables maximaux d'un graphe $G = (V, E)$ puisque $S \subseteq V$ est un ensemble stable de G si et seulement si S est une clique du graphe complémentaire $\overline{G}$. Johnson, Yannakakis et Papadimitriou ont donné dans [JYP88] un algorithme à délai polynomial pour générer tous les ensembles stables maximaux d'un graphe. En combinant ces deux résultats, on obtient un algorithme en $\mathcal{O}^*(1.4423^n)$ pour générer tous les ensembles stables maximaux d'un graphe, et donc un algorithme pour déterminer un ensemble stable de taille maximum.

En 1977, Tarjan et Trojanowski [TT77] ont proposé un algorithme en $\mathcal{O}^*(2^{n/3}) = \mathcal{O}(1.2600^n)$ pour résoudre le problème Ensemble Stable. Cet algorithme de type *Brancher & Réduire* (cette technique sera expliquée plus loin dans ce chapitre) est une énumération d'études de cas permettant de résoudre le problème et d'établir aisément les récurrences nécessaires pour déterminer le temps d'exécution au pire des cas.

En 1986, Jian [Jia86] a établi un algorithme dont le temps d'exécution est $\mathcal{O}(1.2346^n)$. Dans la même année, Robson [Rob86] a publié un algorithme qui s'exécute en temps

$\mathcal{O}(1.2278^n)$. Cet algorithme considère un sommet du graphe et branche en fonction de son degré et de la structure de son voisinage. L'analyse des différents cas permet d'aboutir au temps annoncé. Cet article introduit également la technique *Mémorisation* (voir la section 2.3.1.3 pour une explication de cette technique) qui permet d'améliorer le temps d'exécution au détriment de la nécessité d'un espace exponentiel. Il obtient ainsi un temps de $\mathcal{O}(1.2108^n)$.

Dans un rapport technique de 2001 [Rob01], Robson présente un algorithme d'une dizaine de pages. Grâce à une étude de cas, il obtient un algorithme en $\mathcal{O}(1.1889^n)$.

Les années 90 voient aussi la publication de deux autres algorithmes pour Ensemble Stable. En 1990, Shindo et Tomita publient un algorithme en $\mathcal{O}(1.2737^n)$ [ST90], et en 1999, Beigel [Bei99] donne un algorithme en $\mathcal{O}(1.2227^n)$, sans en proposer une analyse du temps d'exécution.

Enfin, à la conférence SODA 2006, Fomin, Grandoni et Kratsch [FGK06a] donnent un algorithme simple et concis. Ils montrent, en utilisant la technique *Mesurer pour Conquérir* (voir la section 2.3.1.2), que son temps d'exécution est borné supérieurement par $\mathcal{O}(1.2210^n)$ et inférieurement par $\Omega(1.1219^n)$ au pire des cas.

Pour terminer cette section, précisons que quelques résultats ont également été établis pour des classes de graphes particulières. Dans [Bei99], Beigel donne un algorithme en $\mathcal{O}(1.0823^{|E|})$ et un algorithme en $\mathcal{O}(1.1713^n)$ pour les graphes de degré maximum égal à 4. De plus Razgon propose en 2006 dans [Raz06b] un algorithme dont le temps d'exécution est borné par $\mathcal{O}(1.1034^n)$ pour résoudre Ensemble Stable sur les graphes de degré maximum égal à 3.

2.2.2 Résultats pour divers problèmes

De nombreux autres problèmes ont fait l'objet de travaux et nous indiquons dans la table 2.3 les résultats connus pour quelques-uns d'entre eux.

2.3 Techniques principales de conception et d'analyse

Pour les algorithmes polynomiaux que nous pouvons trouver dans la littérature, bien souvent la principale technique utilisée tombe dans l'une des catégories suivantes :

- diviser pour régner ;
- stratégie gloutonne ;
- programmation dynamique.

Ces techniques, parfois combinées entre elles ou nécessitant des structures de données performantes et quelques pré-traitements des données, ont permis d'obtenir un grand nombre d'algorithmes polynomiaux pour résoudre des problèmes fameux.

Tab. 2.3 – Liste non exhaustive d'algorithmes exponentiels publiés.

problème	meilleur algorithme connu	référence
Circuit Hamiltonien	$\mathcal{O}(n^2 2^n)$	[HK62]
Voyageur de Commerce	$\mathcal{O}(n^2 2^n)$	[HK62]
Nombre Chromatique	$\mathcal{O}^*(2^n)$	[BH06, Koi06]
3-Coloration	$\mathcal{O}(1.3289^n)$	[BE05]
SAT	$\mathcal{O}^*(2^n)$	[DP60, DLL62]
3-SAT	$\mathcal{O}(1.4730^n)$	[BK04]
Coupe maximum	$\mathcal{O}(1.7315^n)$	[Wil05]
Ensemble de Sommets en Retour[1]	$\mathcal{O}(1.7548^n)$	[FGP06]
Largeur de Bande[2]	$\mathcal{O}^*(10^n)$	[Fei00]
Somme d'un sous-ensemble	$\mathcal{O}(1.4143^n)$	[Woe03]
Sac à dos Binaire	$\mathcal{O}(1.4143^n)$	[HS74]
Largeur Arborescente	$\mathcal{O}(1.8899^n)$	[Vil06]
Ensemble 3-intersectant[3]	$\mathcal{O}(1.6278^n)$	[Wah04, Wah07]

[1] en anglais, ce problème est appelé Feedback Vertex Set
[2] en anglais, ce problème est appelé Bandwidth
[3] en anglais, ce problème est appelé 3-hitting Set

Par exemple, en utilisant Diviser pour Régner il est possible de trier n nombres en temps $\mathcal{O}(n \log n)$ grâce à l'algorithme Tri Fusion donné par von Neumann et présenté dans [Knu97]. Avec cette technique, il est aussi possible de trouver une paire de points la plus proche dans le plan en temps $\mathcal{O}(n \log n)$ [SH75].

L'algorithme de Dijkstra, basé sur la stratégie gloutonne et utilisant des tas de Fibonacci, permet le calcul du plus court chemin entre deux sommets d'un graphe en temps $\mathcal{O}(m + n \log n)$ [Dij59]. Les algorithmes gloutons de Kruskal et de Prim permettent de calculer un arbre couvrant minimum d'un graphe en temps $\mathcal{O}(m \log n)$ grâce aux tas binaires [Kru56, Pri57]. L'algorithme de Huffman permet de construire un codage préfixe optimal en temps $\mathcal{O}(n \log n)$, avec n la taille de l'alphabet, pour compresser des données [Huf52]. Là encore, l'algorithme est basé à chaque étape sur une stratégie gloutonne.

D'autres problèmes tels que l'alignement de séquence, problème particulièrement utile à la biologie, ou encore la recherche d'une plus longue sous-séquence commune à deux textes possèdent des algorithmes basés sur le paradigme de la programmation dynamique et dont le temps d'exécution est $\mathcal{O}(mn)$, où m et n sont les longueurs des deux séquences (voir par exemple [CHL01]). Également, il est possible de déterminer une triangulation optimale d'un polygone à n sommets en temps $\mathcal{O}(n^3)$ en utilisant ce paradigme de conception d'algorithme [W-J94].

Pour mentionner des résultats de l'auteur, nous avons établi en 2005 [LKLP05] que le problème Domination Romaine peut être résolu en temps linéaire sur les graphes d'intervalles, en utilisant essentiellement la programmation dynamique combinée avec des pré-traitements sur l'entrée, répondant ainsi à une question ouverte donnée dans l'article [CDHH04].

Ces diverses techniques ont clairement fait preuve de leur capacité à résoudre des problèmes non triviaux. Elles sont aujourd'hui enseignées dans de nombreux cours d'algorithmique et les problèmes sus-cités servent souvent à illustrer leur mise en œuvre et leurs applications (voir par exemple les ouvrages [CLRS02, KT06]).

De même, dans le domaine des algorithmes exponentiels, nous retrouvons des techniques de conception permettant d'obtenir des algorithmes meilleurs que les algorithmes naïfs. Nous nous proposons dans la suite d'expliquer ces principales techniques :

- *Brancher & Réduire*, à la section 2.3.1 ;
- *Programmation Dynamique*, à la section 2.3.2 ;
- *Mémorisation*, à la section 2.3.1.3 ;
- *Trier et Chercher*, à la section 2.3.3 ;
- *Inclusion-Exclusion*, à la section 2.3.4.

Dans l'état de l'art proposé dans [Woe03] par Woeginger sont également listées plusieurs techniques usuellement utilisées. L'auteur donne différents exemples permettant d'illustrer l'idée principale de chacune d'elles.

Les paradigmes comme *Brancher & Réduire* ou *Programmation Dynamique* ont fait leur apparition dès les débuts de l'intérêt pour les algorithmes exponentiels (voir respectivement [TT77] et [HK62]). *Mémorisation* est une méthode introduite dans [Rob86]. La technique

que nous appelons *Trier et Chercher* est dû principalement à Horowitz et Sahni [HS74] (voir aussi l'article de Schroeppel et Shamir [SS81]). Enfin, la technique *Inclusion-Exclusion*, bien que déjà utilisée dans [Kar82], a été redécouverte indépendamment dans deux récents articles de FOCS 2006 [BH06, Koi06] pour obtenir le meilleur algorithme connu pour Nombre Chromatique. Cet algorithme demande un temps $\mathcal{O}^*(2^n)$.

2.3.1 Brancher & Réduire

Le paradigme *Brancher & Réduire* est particulièrement utilisé pour concevoir des algorithmes exponentiels. On le retrouve sous d'autres dénominations : *algorithme à arbre de recherche* (*search tree algorithm*), *algorithme à retour en arrière* (*backtracking algorithm*). Le terme d'arbre de recherche fait référence au fait qu'une exécution peut être assimilée à un arbre : étant donnée une instance à résoudre, l'algorithme utilise des appels récursifs sur des instances plus petites pour calculer une solution optimale ; on associe alors à une instance un nœud dans l'arbre, et pour chaque appel récursif nécessaire pour résoudre cette instance (qu'on appelle aussi sous-problème), on associe des fils dans l'arbre de recherche. Cette modélisation d'une exécution sous la forme d'un arbre de recherche est particulièrement utile pour analyser la complexité temporelle d'un algorithme. Elle dépend directement du nombre de sous-problèmes à résoudre. Celui-ci correspond au nombre de nœuds dans l'arbre de recherche. Nous aurons l'occasion de détailler l'analyse de tels algorithmes dans la section 2.3.1.2, ainsi que dans les prochains chapitres. En particulier, au chapitre 5 nous présenterons un algorithme de type *Brancher & Réduire* pour résoudre le problème de la clique dominante minimum dans un graphe.

2.3.1.1 Conception d'algorithmes

Dans un algorithme de type *Brancher & Réduire* sont essentiellement utilisés deux types de règles :

– des règles de réduction, et
– des règles de branchement.

Les règles de réduction ont pour objectif de réduire la taille de l'instance et de simplifier l'instance. Après avoir appliqué une ou plusieurs règles de réduction sur l'instance, un appel récursif unique peut alors, éventuellement, être appliqué sur l'instance modifiée. Ces règles demandent généralement un temps d'exécution polynomial pour être appliquées, en excluant le temps passé dans l'appel récursif.

Les règles de branchement résolvent le problème en cherchant à résoudre récursivement des instances plus petites. Habituellement c'est à ce moment que l'algorithme essaie plusieurs choix, par l'intermédiaire de sous-problèmes à résoudre, pour déterminer une solution. On note aussi que l'algorithme peut comporter des règles d'arrêt qui permettent de stopper la récursion. Ces règles d'arrêt sont exécutées lorsque l'algorithme a trouvé une solution candidate pour être une solution optimale, ou qu'il a déterminé que l'instance courante n'admet pas de solution.

De manière informelle, on peut dire que la force de l'algorithme réside dans la capacité de ses règles de réduction à simplifier (et à réduire) le plus possible l'instance courante à chaque étape. Le cœur de l'algorithme étant les règles de branchement qui permettent de tester différentes possibilités que l'on ne peut décider autrement que par un branchement.

De nombreux algorithmes utilisent cette technique pour résoudre des problèmes NP-difficiles ; parmi eux, DOMINATION, ENSEMBLE STABLE, ENSEMBLE DE SOMMETS EN RETOUR (en anglais, FEEDBACK VERTEX SET), CLIQUE DOMINANTE ou encore SAT [FGK05a, Rob86, FGK06a, Fei00, Raz06a, FGP06, CGA05, DLL62, BK04].

Afin d'illustrer la technique *Brancher & Réduire*, nous donnons un algorithme simple, basé sur ce paradigme, pour résoudre le problème ENSEMBLE STABLE. Cet algorithme, bien qu'il ne soit pas le meilleur connu, est facile à implémenter et en outre permet d'énumérer tous les ensembles stables maximaux par inclusion d'un graphe.

Algorithme EnsStable($G = (V, E)$,S)
Entrées: un graphe $G = (V, E)$ et un ensemble S de sommets non adjacents.
Sortie : l'union de S et d'un ensemble stable de taille maximum de G .
  si *V n'est pas vide* **alors**
    Soit v un sommet de degré minimum de G
    Pour chaque sommet $u \in N[v]$, appeler récursivement l'algorithme EnsStable($G - N[u]$,$S \cup \{u\}$), puis retourner le plus grand ensemble trouvé
  sinon
    retourner S

L'objectif ici n'est pas de donner une preuve complète de la validité de cet algorithme. Néanmoins, on constate que si un ensemble S est un ensemble stable maximal par inclusion d'un graphe G, alors pour tout sommet $u \in V$, $N[u] \cap S \neq \emptyset$. Etant donné un graphe G, il suffit d'un appel à **EnsStable(G,$\emptyset$)** pour calculer un ensemble stable de taille maximum de G. On peut noter que l'algorithme **EnsStable** calcule tous les ensembles stables maximaux du graphe puisque pour chaque sommet v du graphe, on essaye d'ajouter v ou au moins un sommet de son voisinage à l'ensemble stable S. De plus, on sait que le nombre d'ensembles stables maximaux par inclusion d'un graphe est borné par $3^{n/3}$ [MM65]. Il s'ensuit que **EnsStable** a un temps d'exécution qui doit être au moins de $\mathcal{O}(1.4423^n)$. Le théorème 2.3 donné à la section suivante propose de montrer que le temps d'exécution de l'algorithme **EnsStable** est au pire des cas borné par $\mathcal{O}(1.4423^n)$.

2.3.1.2 Analyse du temps d'exécution

De façon générale, supposons que le problème à résoudre soit de taille n et que l'algorithme utilise une règle de branchement demandant de résoudre récursivement k sous-problèmes de taille plus petite pour résoudre le problème. Supposons de plus que ce bran-

chement diminue la taille n par t_i dans la i-ième branche, $1 \leq i \leq k$; on dit que la règle a un vecteur de branchement $(t_1, t_2, \ldots, t_k)$. Si l'on désigne par $T(n)$ le temps requis pour résoudre un problème de taille n, et si le temps nécessaire à l'algorithme en excluant les appels récursifs est polynomial, alors on obtient la récurrence sur le temps d'exécution donnée par

$$T(n) = T(n - t_1) + T(n - t_2) + \cdots + T(n - t_k) + \text{poly}(n).$$

La solution à cette récurrence est de la forme $\mathcal{O}(\text{poly}(n)\alpha^n)$. En utilisant la méthode par *substitution* pour sa résolution [CLRS02], on en déduit que α est la plus grande racine positive de l'équation

$$1 = \frac{1}{\alpha^{t_1}} + \frac{1}{\alpha^{t_2}} + \cdots + \frac{1}{\alpha^{t_k}}.$$

La solution d'une telle équation peut ensuite être calculée à l'aide de méthodes d'analyse numérique standard telles que la *méthode de Newton* [PFT92]. Typiquement, un algorithme de type Brancher & Réduire peut disposer de plusieurs règles de branchement. Le temps d'exécution au pire des cas est alors borné par $\mathcal{O}(\text{poly}(n)(\alpha_{\max})^n)$, où $\alpha_{\max}$ est la plus grande de toutes les solutions aux équations correspondant à ces règles. Pour davantage d'informations sur ce type d'analyse, on pourra se référer par exemple à [Cam94, KL99, SF98].

Autrement dit, pour analyser le temps d'exécution de l'algorithme **EnsStable**, nous pouvons aussi considérer l'arbre récursif correspondant à une exécution de l'algorithme sur une certaine entrée, puis borner le nombre maximum de sous-problèmes récursivement résolus, c'est-à-dire le nombre maximum de nœuds au pire des cas.

Un nœud de l'arbre de recherche correspond à une exécution de l'algorithme sans les appels récursifs. L'algorithme **EnsStable** demande un temps d'exécution polynomial si l'on ignore ces appels récursifs. En effet, le temps alors nécessaire à l'algorithme correspond au test vérifiant si l'ensemble des sommets est vide, au choix d'un sommet de degré minimum dans le graphe et aux éventuels calculs des paramètres d'appels pour les appels récursifs. Le nombre de nœuds dans l'arbre est finalement donné par les récurrences correspondant aux branchements réalisés, comme indiqué dans la preuve du théorème suivant, et expliqué précédemment de façon générale.

Théorème 2.3. *L'algorithme* ***EnsStable(G,∅)*** *s'exécute en temps* $\mathcal{O}(1.4423^n)$.

Démonstration. Notons $T(n)$ le plus grand nombre de nœuds dans l'arbre récursif pour résoudre le problème sur un graphe $G = (V, E)$ à n sommets, en utilisant **EnsStable**. Si le graphe est non vide, alors l'algorithme branche sur chaque sommet $u \in N[v]$ où v est un sommet de degré minimum dans G. Ainsi, pour tout sommet u, on a la relation $d(u) \geq d(v)$. Soit u_i, $1 \leq i \leq d(v)$ les voisins de v. Puisque l'on résout récursivement le problème en résolvant les sous-problèmes ajoutant chaque sommet de $N[v]$ à la solution (on dit que l'algorithme *branche* sur un sommet), on obtient la récurrence

$$T(n) \leq 1 + T(n - (d(v) + 1)) + \sum_{i=1}^{d(v)} T(n - (d(u_i) + 1)).$$

En effet, lors de l'ajout du sommet v à la solution S, l'ensemble de sommets $N[v]$ est supprimé de G et l'on résout récursivement le problème sur $G \setminus N[v]$ qui possède $n - |N[v]| = n - (d(v) + 1)$ sommets. Lorsqu'on ajoute un sommet $u_i \in N(v)$ on résout alors un sous-problème ayant $n - |N[u_i]| = n - (d(u_i) + 1)$ sommets. Puisque pour tout i, $1 \leq i \leq d(v)$, on a $d(u_i) \geq d(v)$, et que la fonction $T(n)$ est monotone, cela implique que $T(n - (d(u_i) + 1)) \leq T(n - (d(v) + 1))$, pour tout i tel que $1 \leq i \leq d(v)$. Ainsi, on obtient

$$\begin{aligned} T(n) &\leq 1 + T(n - (d(v) + 1)) + \sum_{i=1}^{d(v)} T(n - (d(v) + 1)) \\ &\leq 1 + (d(v) + 1)T(n - (d(v) + 1)) \end{aligned}$$

Le polynôme caractéristique de cette récurrence est $x^{d(v)+1} = d(v) + 1$. La solution de la récurrence est donnée par la plus grande racine positive de ce polynôme. Comme la fonction $f(t) = t^{1/t}$ est maximale pour $t = e \approx 2.718$ (pour $t = 3$ si $f : \mathbb{N} \to \mathbb{R}$), on en déduit que la solution de la récurrence est $T(n) = \mathcal{O}^*(3^{n/3})$. □

La démonstration que nous venons de donner a permis d'établir que le temps d'exécution de l'algorithme **EnsStable** est borné par $\mathcal{O}^*(3^{n/3})$. Elle permet également de montrer que le nombre d'ensembles stables maximaux par inclusion est borné par $\mathcal{O}^*(3^{n/3})$. De plus, le nombre d'ensembles stables maximaux d'un graphe peut vraiment atteindre cette borne pour certains graphes : dans un K_3 (un K_3 est un graphe complet à trois sommets) il y a trois ensembles stables maximaux et donc un graphe composé d'une collection disjointe de K_3 contient $3^{n/3}$ ensembles stables maximaux (voir la figure 2.1).

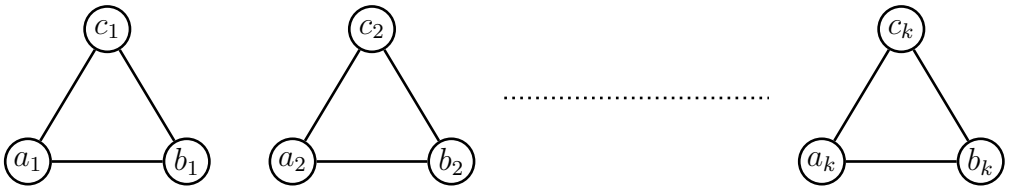


Fig. 2.1 – Exemple d'un graphe à $n = 3k$ sommets contenant $3^k = 3^{n/3}$ ensembles stables maximaux.

Mesurer pour Conquérir. Nous venons de montrer que l'algorithme **EnsStable** demande un temps $\mathcal{O}^*(3^{n/3})$ au pire des cas et qu'il existe des graphes pour lesquels ce temps est atteint. Malheureusement, contrairement aux algorithmes polynomiaux et à l'algorithme **EnsStable**, le temps d'exécution d'algorithmes construits en utilisant le paradigme Brancher & Réduire est souvent surestimé. En particulier, si nous considérons un algorithme comportant plusieurs règles de branchement, le temps d'exécution au pire des cas peut être borné par la plus grande solution correspondant aux différentes récurrences

obtenues de l'analyse de chacune des règles de réduction. Néanmoins, cela n'implique pas nécessairement que l'algorithme va constamment utiliser cette règle de branchement et nous aurons donc probablement surestimé le temps d'exécution au pire des cas, sauf s'il existe réellement des instances pour lesquelles l'algorithme n'applique que cette règle ou des règles impliquant un même temps d'exécution. Il est donc possible que des règles de branchement ne soient que rarement utilisées alors qu'avec l'analyse standard elles sont directement responsables du temps d'exécution au pire des cas de l'algorithme. C'est pourquoi une attention particulière a été portée à l'amélioration de la précision des analyses du temps d'exécution.

Déjà Robson s'était intéressé à ce problème pour analyser un algorithme résolvant le problème Ensemble Stable [Rob86]. Malgré le fait que les graphes réguliers provoquent des récurrences plus *coûteuses* par son algorithme, il avait alors remarqué que ces graphes ne contribuaient pas significativement au temps d'exécution au pire des cas puisqu'un tel graphe apparaît au plus une fois dans chaque chemin partant de la racine dans l'arbre de recherche.

Une technique proposée par Chen *et al.* dans [CKX05] consiste à analyser le comportement d'un algorithme, pour résoudre le problème Couverture de Sommets sur des graphes de degré au plus 3, de façon « amortie ». L'idée principale est de balancer des branchements coûteux avec des branchements moins coûteux apparaissant sur un même chemin de l'arbre de recherche.

Une autre tentative pour mieux borner le temps d'exécution est l'utilisation d'une mesure *non standard* sur la taille de l'instance. Typiquement, ce type de mesure nécessite la prise en compte de plusieurs paramètres dans l'analyse. Dans une analyse *standard*, comme celle que nous avons proposée dans la preuve du thèorème 2.3, la mesure utilisée sur la taille de l'instance représente le nombre de sommets de l'instance courante. Pour chaque appel récursif, on compte le nombre de sommets supprimés et on obtient ainsi une borne supérieure sur la taille de l'instance pour laquelle l'appel récursif est effectué. L'idée principale de la technique *Mesurer pour Conquérir* suggère de choisir une fonction de mesure différente qui prend mieux en compte les changements induits sur une instance lors d'un branchement. Par exemple, lorsqu'un sommet est supprimé dans un graphe, non seulement on obtient un graphe de taille plus petite, mais les voisins de ce sommet voient leurs degrés décroître. Cela peut avoir des effets bénéfiques pour une exécution future de l'algorithme, c'est-à-dire lors d'un prochain appel récursif. Pour mieux mesurer ces changements, il est avantageux de choisir une mesure plus appropriée qui permet de borner plus finement le temps d'exécution d'un algorithme Brancher & Réduire. En particulier, au lieu de compter le nombre de sommets du graphe, la mesure pourrait dénombrer les sommets en fonction de leur degré et pondérer leur importance selon leur rôle joué dans l'algorithme.

Dans [Epp06], Eppstein explique comment analyser des récurrences avec plusieurs paramètres. Il montre qu'il est suffisant de considérer une récurrence ne dépendant que d'une seule variable égale à une combinaison pondérée des différents paramètres pour déterminer le comportement asymptotique d'une récurrence à plusieurs paramètres.

Eppstein a utilisé cette technique pour établir un algorithme en $\mathcal{O}(1.2600^n)$ pour le

problème Voyageur de Commerce sur les graphes de degré au plus 3 [Epp07]. Fomin *et al.* l'utilisent aussi, sous le nom *Mesurer pour Conquérir*, pour proposer notamment des algorithmes pour Domination et Ensemble Stable en temps $\mathcal{O}(1.5263^n)$ et $\mathcal{O}(1.2210^n)$ respectivement.

Au chapitre 5, nous donnerons une analyse utilisant une mesure non standard pour borner le temps d'exécution d'un algorithme permettant de résoudre le problème Clique Dominante de taille minimum. Cela nous permettra d'illustrer concrètement la mise en oeuvre d'une telle technique d'analyse.

Les algorithmes exponentiels de type Brancher & Réduire que l'on trouve dans la littérature pourraient être classés en deux catégories : « algorithme long et analyse du temps d'exécution facilitée » et « algorithme court et analyse du temps d'exécution complexe ». Cette deuxième catégorie d'algorithmes est plutôt récente et l'introduction de la technique d'analyse *Mesurer pour Conquérir* combinée au paradigme Brancher & Réduire a favorisé leur apparition. Cela permet d'obtenir notamment des algorithmes relativement simples à implémenter dont les performances au pire des cas sont mieux bornées par une analyse plus précise.

2.3.1.3 Mémorisation

Il est parfois possible de réduire le temps d'exécution d'algorithmes basés sur le paradigme Brancher & Réduire. Pour obtenir une telle amélioration, un espace de stockage exponentiel est alors indispensable. Cette technique, due à Robson [Rob86] et appelée *Mémorisation*, demande de mémoriser dans une base de données les solutions de tous les sous-problèmes résolus lors des appels récursifs. Ensuite, lorsqu'un même sous-problème nécessite d'être résolu une nouvelle fois, le résultat déjà calculé est simplement récupéré de la base de données. On évite ainsi de nouveaux calculs coûteux en temps, puisque maintenant la seule opération coûteuse est celle demandant de retrouver le sous-problème et sa solution dans la base de données. Cette base de données peut être implémentée de sorte que la requête est effectuée en temps polynomial (en fait en temps logarithmique en le nombre d'entrées dans la base) [Knu97].

La technique *Mémorisation* a été introduite dans le domaine des algorithmes exponentiels et utilisée par Robson [Rob86] pour obtenir un algorithme en temps $\mathcal{O}(1.2109^n)$ et espace exponentiel pour résoudre le problème Ensemble Stable, contre $\mathcal{O}(1.2278^n)$ et espace polynomial. Cette même technique a également été utilisée par Fomin *et al.* pour obtenir l'algorithme connu le plus rapide en $\mathcal{O}(1.5137^n)$ et espace exponentiel pour résoudre Domination, alors que leur algorithme demandant seulement un espace polynomial avait pour temps d'exécution $\mathcal{O}(1.5263^n)$. Il a aussi été fait usage de *Mémorisation* dans le domaine des algorithmes à paramètres fixés pour améliorer des temps d'exécution d'algorithmes pour Couverture de Sommets (voir par exemple [NR03, CG05]). Nous utiliserons la technique pour établir à la section 5.6 un algorithme qui calcule une clique dominante de taille minimum en temps $\mathcal{O}(1.3234^n)$ et espace exponentiel. Ce résultat sera établi grâce à un nouvel algorithme que nous proposerons pour résoudre ce même problème

en temps $\mathcal{O}(1.3387^n)$ et espace polynomial. Puisque pour établir le temps d'exécution de l'algorithme en espace polynomial, nous utiliserons *Mesurer pour Conquérir* ; l'étude d'une mesure non standard nous obligera donc à adapter la technique originelle de Robson.

Pour illustrer sans attendre cette technique nous reproduisons l'exemple simple donné dans [FGK05b]. On demande de résoudre le problème Couverture de Sommets défini à la section 1.4.1. Pour cela, Fomin *et al.* considèrent l'algorithme `MVC` (pour *Minimum Vertex Cover*) reproduit ci-après.

Algorithme `MVC`(G)
Entrée: un graphe $G = (V, E)$**.**
Sortie : une couverture de sommets minimum de G**.**

```
si le graphe est vide alors retourner ∅
sinon si il existe un sommet v tel que d(v) = 0 alors
    retourner MVC(G \ {v})
sinon si il existe un sommet v tel que d(v) = 1 alors
    Soit w le voisin de v
    retourner {w} ∪ MVC(G \ {v, w})
sinon
    Choisir un sommet v de degré maximum
    si d(v) = 2 alors
        Résoudre le problème en temps polynomial (G est une collection
        de cycles et de chemins) et retourner la plus petite couverture de
        sommets
    sinon
        S1 ← {v} ∪ MVC(G \ {v})
        S2 ← N(v) ∪ MVC(G \ N[v])
        retourner le plus petit des ensembles S1 ou S2
```

Théorème 2.4. *L'algorithme* `MVC` *trouve une couverture de sommets de plus petite taille d'un graphe en temps* $\mathcal{O}(1.3803^n)$ *et espace polynomial.*

Démonstration. La preuve de validité de `MVC` est particulièrement facile et s'appuie sur la propriété fondamentale d'une couverture de sommets, à savoir que pour toute arête du graphe au moins l'une de ses extrémités appartient à la couverture. Ceci signifie en particulier que si l'on exclut un sommet de la couverture, alors tous ses voisins doivent y appartenir.

Notons maintenant $T(n)$ le nombre de feuilles au pire des cas dans l'arbre de recherche correspondant à une exécution de `MVC` sur un graphe a n sommets. La récurrence correspondant à la règle de branchement est donnée par $T(n) \leq T(n-1) + T(n-d(v)-1)$ et correspond au fait d'ajouter le sommet v à la solution ou d'y ajouter l'ensemble de ses $d(v)$ voisins. La solution de cette récurrence est obtenue en résolvant l'équation caractéristique

$x^{d(v)+1} = 1 + x^{d(v)}$ dont la plus grande racine est obtenue lorsqu'on branche sur un sommet v de degré 3. Dans ce cas, soit qu'on ajoute le sommet v à la couverture, auquel cas on supprime v de G, soit qu'on ajoute ses trois voisins à la couverture, auquel cas on supprime $N[v]$. On obtient alors la récurrence

$$T(n) \leq T(n-1) + T(n-4)$$

dont la solution satisfait $T(n) < 1.3803^n$. Comme chaque exécution de `MVC`, hormis les éventuels appels récursifs, demande un temps polynomial, le temps d'exécution au pire des cas est donc $\mathcal{O}(1.3803^n)$. □

Regardons maintenant comment appliquer *mémorisation* à cet algorithme, et appelons `exp-MVC` la nouvelle version de l'algorithme nécessitant un espace exponentiel. Avant de résoudre un nouveau sous-problème G', l'algorithme `exp-MVC` vérifie dans une base de données si une solution pour G' a déjà été précédemment calculée. Si une telle solution existe, elle est simplement récupérée de la base. Sinon `exp-MVC` résout le sous-problème G' puis stocke dans la base, pour l'entrée G', le résultat renvoyé par `exp-MVC`(G'). Comme G a $\mathcal{O}(2^n)$ sous-graphes induits, la base peut être implémentée de sorte que chaque requête demande un temps polynomial en n.

On cherche à présent à évaluer le temps d'exécution de `exp-MVC`. Soit $T_h(n)$, $h \leq n$, le nombre de sous-problèmes dont l'entrée est un sous-graphe à h sommets résolus pour calculer la solution d'un graphe à n sommets. On observe qu'il existe $\binom{n}{h}$ sous-graphes induits de G avec h sommets. Ceci implique donc que $T_h(n) \leq \binom{n}{h}$. De plus, d'après l'analyse de `MVC`, et comme l'ensemble des sous-problèmes qui sont des sous-graphes de taille h sont au maximum à la profondeur $n-h$ dans l'arbre de recherche, on en déduit que $T_h(n) \leq 1.3803^{n-h}$. Ainsi, on a

$$T_h(n) \leq \min\left\{\binom{n}{h}, 1.3803^{n-h}\right\}.$$

En utilisant l'approximation de Stirling, et en balançant les deux termes, on obtient que pour chaque valeur de h, $T_h(n) \leq 1.3803^{n-\alpha n} < 1.3426^n$, où $\alpha > 0.0865$ satisfait

$$1.3803^{1-\alpha} = \frac{1}{\alpha^\alpha (1-\alpha)^{1-\alpha}}.$$

Par conséquent, le temps d'exécution de `exp-MVC` est borné par $\mathcal{O}(1.3426^n)$.

2.3.2 Programmation dynamique

La *programmation dynamique* est une technique classique de conception d'algorithmes pour résoudre des problèmes en temps polynomial. Elle est traditionnellement enseignée dans les cours d'algorithmique et fait l'objet de chapitres dans des ouvrages tels que [CLRS02, KT06]. L'idée générale est de résoudre un problème en utilisant des solutions à des sous-problèmes précédemment résolus. Pour ce faire, la programmation dynamique

applique une approche dite « du bas vers le haut », c'est-à-dire qu'on commence par résoudre les sous-problèmes les plus petits, et donc les plus faciles, pour ensuite résoudre des problèmes de plus en plus grands, jusqu'à finalement déterminer une solution du problème initial. Pour obtenir un *bon* temps d'exécution, chaque sous-problème n'est résolu qu'au plus une fois, et une fois la solution d'un sous-problème calculée, elle est conservée dans une structure de données. Les algorithmes polynomiaux obtenus en employant cette technique sont nombreux et nous en avons déjà cité quelques-uns en début de section ; nous verrons ci-après que la technique peut aussi être utilisée pour résoudre des problèmes NP-difficiles et ainsi obtenir des algorithmes exponentiels. Bien souvent, comme la programmtion dynamique nécessite de stocker les solutions de tous les sous-problèmes résolus, il est également nécessaire de disposer d'un espace exponentiel.

L'un des tous premiers algorithmes exponentiels calculant la solution d'un tel problème est dû à Held et Karp. Dans leur article de 1962 [HK62], ils utilisent cette technique pour résoudre le problème du voyageur de commerce en temps $\mathcal{O}(n^2 2^n)$ sur un graphe à n sommets. On rapelle ci-dessous la définition de ce problème.

Voyageur de Commerce (Travelling Salesman Problem)

entrée : Un graphe complet $G = (V, E)$, une fonction de pondération $d : E \to \mathbb{R}$ et un entier $k \in \mathbb{R}$.

question : Existe-t-il une séquence $(v_{i_1}, v_{i_2}, \ldots, v_{i_n})$ des n sommets, aussi appelée *tournée*, telle que $d(v_{i_1}, v_{i_n}) + \sum_{j=1}^{n-1} d(v_{i_j}, v_{i_{j+1}}) \leq k$?

Le problème d'optimisation classique est celui d'un représentant de commerce qui doit parcourir n villes puis retourner à son point de départ. Une distance $d(v_i, v_j)$ est donnée entre chaque ville v_i et v_j. L'objectif est de déterminer dans quel ordre le représentant doit visiter chacune des villes de sorte à minimiser la distance totale parcourue.

L'algorithme trivial permettant de résoudre ce problème demande un temps en $\mathcal{O}^*(n!)$. Il consiste à énumérer toutes les permutations possibles des n villes et pour chacune d'elle à calculer en temps polynomial la distance à parcourir. Finalement une solution est une tournée dont la distance est minimum parmi toutes les permutations explorées. Comme l'algorithme proposé par Held et Karp est à la fois simple, particulièrement joli et le meilleur connu, nous le ré-expliquons en quelques lignes.

Pour tout sous-ensemble $X \subseteq \{v_2, \ldots, v_n\}$ et pour toute ville $v_i \in X$ on calcule la valeur $\mathrm{Opt}[X, i]$ représentant la longueur de la plus courte tournée qui commence à la ville v_1, visite toutes les villes de $X \setminus v_i$ (dans un ordre arbitraire à déterminer) et s'arrête à la ville v_i.

Les valeurs de $\mathrm{Opt}[X, i]$ sont calculées en regardant les ensembles X par ordre de cardinalité croissante de la façon suivante :

$$\mathrm{Opt}[\{v_i\}, i] = d(v_1, v_i);$$

$$\mathrm{Opt}[X, i] = \min_{j \in X \setminus \{v_i\}} \mathrm{Opt}[X \setminus \{i\}, j] + d(v_j, v_i).$$

Finalement la solution optimale est donnée par :

$$\min_{2 \leq j \leq n} \mathrm{Opt}[\{v_2, \ldots, v_n\}, j] + d(v_j, v_1).$$

Le temps d'exécution de cet algorithme est $\mathcal{O}(n^2 2^n)$ puisqu'on calcule $\mathrm{Opt}[X, i]$ pour 2^n sous-ensembles de X, n valeurs de i, et étant donné un ensemble X et un entier i, il faut un temps $\mathcal{O}(n)$ pour calculer la valeur de $\mathrm{Opt}[X, i]$.

Également en utilisant la programmation dynamique, Lawler a proposé en 1976 un algorithme exponentiel pour déterminer en temps $\mathcal{O}(2.4423^n)$ le *nombre chromatique* d'un graphe G à n sommets. Le nombre chromatique, noté $\chi(G)$, d'un graphe G est le plus petit nombre de sous-ensembles stables en lequel on peut partitionner les sommets de G.

Nombre Chromatique

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Peut-on colorier G avec au plus k couleurs ? C'est-à-dire, est-il possible de partitionner V en au plus k ensembles stables ?

L'algorithme utilise plusieurs résultats bien connus. Le premier est que pour tout graphe G, il existe une coloration optimale pour laquelle une classe de couleur, c'est-à-dire un ensemble de sommets coloriés par une même couleur, induit un ensemble stable maximal par inclusion. Le deuxième résultat utilisé provient de l'article de Moon et Moser [MM65] qui indique qu'un graphe à n sommets contient au plus $3^{n/3}$ ensembles stables maximaux par inclusion. De plus, il existe des algorithmes à délai polynomial qui permettent d'énumérer ces ensembles comme par exemple celui de Johnson *et al.* [JYP88].

Étant donné un graphe $G = (V, E)$, l'algorithme de Lawler calcule pour chaque sous-ensemble $X \subseteq V$ la valeur de $\mathrm{Opt}[X]$ égale à $\chi(G[X])$. En regardant les ensembles X par ordre de cardinalité croissante, cette valeur est calculée de la façon suivante :

$$\begin{aligned} \mathrm{Opt}[\emptyset] &= 0; \\ \mathrm{Opt}[X] &= 1 + \min\{\mathrm{Opt}[X \setminus I] \text{ tel que } I \text{ est un} \\ &\qquad \text{ensemble stable maximal de } G[X]\}. \end{aligned}$$

Pour analyser le temps d'exécution de cet algorithme on observe que pour un ensemble $X \subseteq V$ donné, le temps nécessaire au calcul de $\mathrm{Opt}[X]$ est dominé par le temps demandé pour générer les ensembles stables maximaux de $G[X]$. Ainsi, on obtient le temps d'exécution par la formule

$$\sum_{k=0}^{n} \binom{n}{k} 3^{k/3} = (1 + 3^{1/3})^n = 2.4423^n.$$

2.3.3 Trier et Chercher

Trier et Chercher est une technique de conception assez peu employée dans le domaine des algorithmes exponentiels. De ce fait, la technique est très méconnue. Elle date des années soixante-dix, lorsque Horowitz et Sahni [HS74] ont proposé un algorithme pour améliorer la compléxité de l'algorithme trivial de $\mathcal{O}^*(2^n)$ à $\mathcal{O}^*(2^{n/2})$ pour résoudre le problème du sac à dos binaire, aussi appelé sac à dos discret, et défini ci-après. En 1981, Schroeppel et Shamir, toujours avec cette même technique, ont montré qu'un espace en $\mathcal{O}^*(2^{n/4})$ était suffisant pour résoudre en temps $\mathcal{O}^*(2^{n/2})$ ce problème.

Sac à dos Binaire (Binary Knapsack)

entrée : Un ensemble $O = \{o_1, \ldots, o_n\}$ de n objets, chacun ayant une valeur $v(o_i)$ et un poids $w(o_i)$, $1 \leq i \leq n$, et un entier positif W.

question : Trouver un ensemble $O' \subseteq O$ tel que $\sum_{o \in O'} w(o) \leq W$ et dont la valeur totale définie par $\sum_{o \in O'} v(o)$ soit la plus grande possible.

La technique peut se décrire de façon très générale. Supposons que l'on ait un certain problème Π à résoudre sur une instance I de taille n, et que I soit un ensemble (d'entiers, de sommets, ...). Une des idées fondamentales de *Trier et Chercher* est de diviser le problème en deux sous-problèmes, comme le ferait une approche *Diviser pour Régner* classique. Contrairement à *Diviser pour Régner* qui chercherait à résoudre récursivement les sous-problèmes en appliquant à nouveau le même paradigme jusqu'à obtenir des problèmes de taille très petite, l'approche *Trier et Chercher* ne divise qu'une seule fois pour obtenir deux problèmes I_1 et I_2 de taille $n/2$. Ensuite, on énumère tous les sous-ensembles $S \subseteq I_1$. À chaque sous-ensemble S est associé un vecteur de taille n dont le calcul se fait en temps polynomial et est fonction du problème considéré. Les vecteurs ainsi obtenus en temps $\mathcal{O}^*(2^{n/2})$ sont stockés dans une table T_1. On répète ce même processus pour tous les sous-ensembles de I_2 et l'on stocke les vecteurs correspondant dans une table T_2. De plus, les vecteurs de T_2 sont triés par ordre lexicographique en temps $\mathcal{O}(n2^{n/2}) = \mathcal{O}^*(2^{n/2})$ en utilisant un tri par base (en anglais, *Radix Sort*) ou même en temps $\mathcal{O}(2^{n/2} \log 2^{n/2}) = \mathcal{O}^*(2^{n/2})$ en utilisant l'algorithme classique Tri Fusion. Cette étape est importante pour la suite de l'exécution de l'algorithme. Finalement, pour chaque vecteur $\vec{v_1}$ de la table T_1, on cherche s'il existe un vecteur $\vec{v_2}$ dans T_2 tel que la somme de $\vec{v_1}$ et $\vec{v_2}$ permette d'obtenir un vecteur objectif $\vec{o}$. La recherche d'un tel vecteur $\vec{v_2} \in T_2$, s'il existe, peut se faire en temps $\mathcal{O}(n)$ fois la longueur des vecteurs grâce à une recherche dichotomique [Knu97, p. 409 ff.].

Au chapitre 6 nous décrirons un algorithme basé sur cette approche pour résoudre une généralisation du problème de la domination. Nous expliquons à présent l'emploi de cette technique pour résoudre le problème Somme d'un sous-ensemble.

Somme d'un sous-ensemble (Subset Sum)

entrée : Un ensemble fini $A \subseteq \mathbb{N}$ d'entiers positifs et un entier positif W.
question : Existe-t-il un sous-ensemble $A' \subseteq A$ tel que $\sum_{a \in A'} a = W$?

Soit $A = \{a_1, a_2, \ldots, a_n\}$ et W une instance du problème Somme d'un sous-ensemble avec $|A| = n$. Soit A_1 et A_2 une partition en deux sous-ensembles de A tels que $|A_1| = |A_2| = n/2$ [1]. Pour chaque sous-ensemble $S \subseteq A_i$, $i \in \{1, 2\}$, on calcule la valeur $w_S = \sum_{a \in S} a$ que l'on stocke dans la table T_i avec un vecteur $\vec{v_S}$ de longueur n dont la k-ième composante est égale à 1 si a_k appartient à S, égale à 0 sinon. Après avoir calculé tous les vecteurs de T_1 et de T_2, les vecteurs de T_2 sont triés par valeur w_S croissante. Finalement, le problème Somme d'un sous-ensemble admet une solution si et seulement s'il existe un vecteur $\vec{v_{S_1}} \in T_1$ et un vecteur $\vec{v_{S_2}} \in T_2$ tels que $w_{S_1} + w_{S_2} = W$. Comme expliqué précédemment, puisque les vecteurs de T_2 sont triés, étant donné un vecteur $\vec{v_{S_1}}$ de T_1, il est facile de vérifier en temps polynomial s'il existe un vecteur $\vec{v_{S_2}}$ dans T_2 tel que $w_{S_2} = W - w_{S_1}$. En effectuant ce test polynomial pour les $2^{n/2}$ vecteurs de T_1, une solution au problème est trouvée ou l'algorithme retourne que l'instance n'admet pas de solution. Le temps d'exécution total et l'espace requis par cet algorithme sont bornés par $\mathcal{O}^*(2^{n/2})$.

2.3.4 Inclusion et Exclusion

Le principe d'inclusion-exclusion est une méthode connue en mathématiques qui indique le nombre d'éléments présents dans l'union d'un nombre fini d'ensembles. Le principe est expliqué dans de nombreux livres de combinatoire et de mathématiques discrétes [And03, Cam94, Rio58, Ros06]. Si l'on a seulement deux ensembles A_1 et A_2, on obtient $|A_1 \cup A_2| = |A_1| + |A_2| - |A_1 \cap A_2|$. Pour trois ensembles, la formule devient $|A_1 \cup A_2 \cup A_3| = |A_1| + |A_2| + |A_3| - |A_1 \cap A_2| - |A_1 \cap A_3| - |A_2 \cap A_3| + |A_1 \cap A_2 \cap A_3|$. De façon générale, pour un nombre n d'ensembles on a la relation :

$$\begin{aligned} |A_1 \cup A_2 \cup \cdots \cup A_n| \;=\; & \sum_{1 \le i \le n} |A_i| - \sum_{1 \le i < j \le n} |A_i \cap A_j| \\ & + \sum_{1 \le i < j < k \le n} |A_i \cap A_j \cap A_k| \\ & - \cdots + (-1)^{n+1} |A_1 \cap A_2 \cap \ldots A_n| \end{aligned}$$

L'idée générale de cette formule a été reprise très récemment et indépendamment par deux groupes d'auteurs, Björklund et Husfeldt [BH06] et Koivisto [Koi06], afin d'obtenir des algorithmes exponentiels pour résoudre des problèmes de partitionnement fondamentaux, notamment pour le problème Nombre Chromatique qui demande de partitionner les

[1] Si n est impair, alors on considère A_1 et A_2 tels que $|A_1| = \lfloor n/2 \rfloor$ et $|A_2| = \lceil n/2 \rceil$; dans ce cas, la technique décrite s'applique également et permet aussi d'obtenir un algorithme dont le temps d'exécution est borné par $\mathcal{O}^*(2^{n/2})$.

sommets d'un graphe en un nombre minimum d'ensembles stables. Nous présentons ci-après le fonctionnement de cette technique pour ce problème en se basant sur l'article [BHK07] qui combine les résultats des deux articles [BH06] et [Koi06].

Soit $G = (V, E)$ un graphe. Tout d'abord, il est montré dans [BHK07] qu'étant donné un ensemble V à n éléments et une famille $\mathcal{F}$ de sous-ensembles de V, l'ensemble V peut être couvert par k ensembles de $\mathcal{F}$ si et seulement si

$$\sum_{X \subseteq V} (-1)^{|X|} a(X)^k > 0$$

où $a(X)$ représente le nombre d'ensembles de $\mathcal{F}$ qui n'intersectent pas X. Pour le cas de la coloration, la famille $\mathcal{F}$ est l'ensemble des ensembles stables (non nécessairement maximaux) du graphe. Ensuite, les auteurs montrent comment il est possible de calculer la valeur $a(X)$ pour tout X de façon efficace. Pour cela, ils calculent $a(X)$ en considérant les ensembles X par ordre de cardinalité décroissante et en utilisant la récurrence :

$$a(X) = a(X \cup \{v\}) + a(X \cup N[v]) + 1$$

pour un sommet quelconque $v \notin X$, avec $a(V) = 0$. La formule est justifiée par le fait que $a(X \cup \{v\})$ dénombre les ensembles ne contenant pas v et qui n'intersectent pas X ; $a(X \cup N[v])$ dénombre les ensembles contenant v et qui n'intersectent pas X. Finalement, on ajoute l'ensemble contenant uniquement v. Ainsi, toutes les valeurs $a(X)$ sont calculées en temps et espace $\mathcal{O}^*(2^n)$ et la formule est évaluée en temps $\mathcal{O}^*(2^n)$ puisqu'elle demande de considérer tous les sous-ensembles de V. Finalement, pour déterminer le nombre chromatique, une recherche dichotomique sur la valeur de k est effectuée, et la formule est calculée au plus $\log n$ fois.

2.4 Conclusion

Les techniques de conception et d'analyse précédemment présentées ont abouti à la résolution de divers problèmes NP-difficiles. Les exemples que nous nous sommes attachés à donner illustrent cette utilité. Dans les prochains chapitres, nous utiliserons également quelques-unes de ces techniques et développerons de nouvelles approches pour la résolution de problème.

Au chapitre 4, nous proposerons une approche générale, basée sur le degré maximum et sur la largeur arborescente d'un graphe, afin d'obtenir des algorithmes exponentiels pour résoudre Domination sur diverses classes de graphes. Nous proposerons également un algorithme basé sur le paradigme de la programmation dynamique, avec une étape de prétraitement, qui aboutira à un algorithme pour Domination sur les graphes possédant un grand ensemble stable.

Au chapitre 5, nous donnerons un algorithme de type Brancher & Réduire pour déterminer une clique dominante de taille minimum. L'analyse de son temps d'exécution

nécessitera une fonction de mesure non standard sur la taille de l'instance et la technique Mesurer pour Conquérir. Des bornes inférieures sur le temps d'exécution au pire des cas seront établies pour différents algorithmes. De plus, nous utiliserons Mémorisation afin d'obtenir un algorithme asymptotiquement plus rapide, au détriment de la nécessité d'un espace exponentiel.

Au chapitre 6, nous développerons une nouvelle approche : une combinaison de la technique habituelle de branchement avec un astucieux système de rechargement. Cette technique, que nous bâptisons *Brancher & Recharger* sera très utile pour établir un algorithme pour une généralisation de la domination, la (σ, ϱ)-domination. De plus, la technique Trier et Chercher nous permettra de résoudre ce même problème pour quelques ensembles σ et ϱ.

Au chapitre 7, nous proposerons un algorithme Brancher & Réduire pour résoudre un problème de domination partielle et un algorithme basé sur la programmation dynamique pour résoudre un problème de domination avec des puissances variables.

Chapitre 3

Problème de la domination et état de l'art

3.1 Présentation du problème

Le problème Domination, appelé en anglais Dominating Set, est un problème de graphes classique et bien connu. Il est déjà recensé en 1979 dans la longue liste des problèmes NP-complets du livre de Garey et Johnson [GJ79] sous la référence [GT2]. Commençons par donner la définition du problème sous la forme d'un problème de décision :

Domination (Dominating Set)

entrée : Un graphe $G = (V, E)$ et un entier positif $k \leq |V|$.

question : Existe-t-il un ensemble dominant de taille au plus k pour le graphe G, c'est-à-dire un sous-ensemble de sommets $D \subseteq V$ avec $|D| \leq k$ et tel que pour tout sommet $u \in V \setminus D$, il existe un sommet $v \in D$ pour lequel l'arête $\{u, v\}$ appartient à E ?

Une façon plus courte de définir le problème consiste simplement à demander si le graphe $G = (V, E)$ donné en entrée admet un sous-ensemble $D \subseteq V$ tel que $N[D] = V$, et $|D| \leq k$ pour un k donné. Le problème d'optimisation qui correspond à Domination demande simplement de déterminer un ensemble dominant de plus petite taille pour un graphe donné. Dans la suite, s'il n'y a pas d'ambiguïté nous désignerons par Domination indifféremment le problème de décision ou le problème d'optimisation qui lui correspond. On note $\gamma(G)$ la taille d'un ensemble dominant minimum de G. Un exemple est proposé à la figure 3.1.

La preuve de NP-complétude du problème Domination est basée sur le problème de Couverture de Sommets défini ci-après, et se sert d'un remplacement local. (Voir [GJ79] pour une description de ce type de preuve. Une preuve détaillée est donnée dans

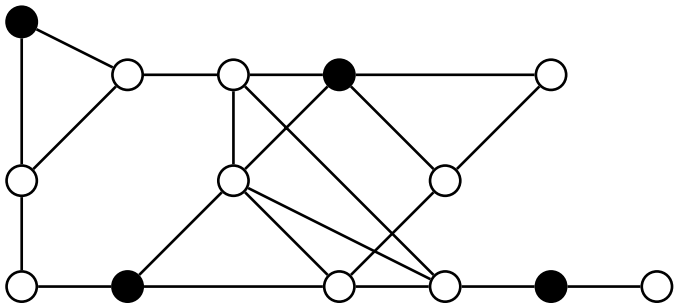


Fig. 3.1 – Exemple de graphe dont les sommets coloriés en noir forment un ensemble dominant minimum de taille 4.

[Lie03] et demande, pour une entrée du problème Couverture de Sommets, de remplacer chaque arête du graphe par un triangle.)

Couverture de Sommets (Vertex Cover)

entrée : Un graphe $G = (V, E)$ et un entier positif $k \leq |V|$.

question : Existe-t-il une couverture de sommets de taille au plus k pour le graphe G, c'est-à-dire, un sous-ensemble de sommets $C \subseteq V$ avec $|C| \leq k$ et tel que pour toute arête $\{u, v\} \in E$ au moins une des extrémités appartient à C ?

Le problème de la domination est également connu pour être NP-complet sur de nombreuses classes de graphes ; certaines d'entre elles seront d'ailleurs étudiées dans ce chapitre ainsi que dans le chapitre 4 où nous proposerons des algorithmes exacts et exponentiels pour Domination sur les graphes cordaux, les graphes cercles, les graphes denses, les graphes faiblement cordaux et les graphes bipartis. La table 3.1 indique la complexité de Domination pour quelques classes de graphes.

On peut facilement résoudre le problème en temps $\mathcal{O}^*(2^n)$. Pour ce faire, on énumère les 2^n sous-ensembles de sommets. Pour chacun d'eux on vérifie en temps $\mathcal{O}(n+m)$ s'il est dominant, puis on retourne un ensemble dominant de plus petite cardinalité.

D'un point de vue de la complexité paramétrée, Domination est $W[2]$-complet [DF99]. Il est donc très improbable que ce problème puisse admettre un algorithme à paramètre fixé pour le résoudre, à moins que $W[2] = \text{FPT}$, ce qui est considéré comme une relation fortement improbable.

Concernant l'approximation d'une solution optimale, Johnson a montré en 1974 qu'une solution pour Domination est approchable avec un facteur $1 + \log n$ où n est le nombre de sommets du graphe [Joh74]. Ceci signifie qu'étant donné un graphe G, l'algorithme de [Joh74] est capable de déterminer un ensemble dominant D tel que $|D|/\gamma(G) \leq 1 + \log n$.

Tab. 3.1 – Quelques résultats de complexité pour le problème DOMINATION

classe de graphe	complexité	référence
planaire	NP-c	[GJ79]
parfait	NP-c	cordal $\subseteq$ parfait
cordal	NP-c	[BJ82]
faiblement cordal	NP-c	cordal $\subseteq$ faiblement cordal
4-cordal	NP-c	cordal $\subseteq$ 4-cordal
split	NP-c	[Ber84]
biparti	NP-c	[Ber84]
comparabilité	NP-c	[Ber84]
co-comparabilité	P	[KS93]
cograph e	P	cograph e $\subseteq$ permutation
intervalles	P	[Cha98, HT91]
intervalles circulaires	P	[Cha98, HT91]
permutation	P	[FK85]
cercle	NP-c	[Kei93]
sans AT	P	[Kra00]
largeur arborescente bornée	P	[AP89, Tel94]
arbre	P	arbre $\subseteq$ largeur arborescente bornée
degré borné par 3	NP-c	[GJ79]
grille	non connue	

En 1997, Raz et Safra [RS97] ainsi que Feige [Fei98] ont montré que le problème n'est pas approchable avec un facteur $c \log n$, pour une constante $c < 1$ positive, à moins que NP $\subseteq$ DTIME($n^{\mathcal{O}(\log \log n)}$). Par conséquent, il semble improbable d'obtenir un algorithme polynomial donnant une solution à un facteur constant près d'une solution optimale.

Pour un état de l'art détaillé sur la théorie de la domination, nous référons le lecteur aux livres [HHS98a, HHS98b] de Haynes, Hedetniemi et Slater.

Différentes variantes du problème de la domination ont été intensivement étudiées. Pour un ensemble dominant D d'un graphe il peut être requis que D vérifie également quelques propriétés supplémentaires. Par exemple que $G[D]$ n'a pas de sommet isolé, que $G[D]$ soit connexe, que D soit un ensemble stable ou une clique, etc. Pour toutes ces variantes de l'ensemble dominant, le problème correspondant est typiquement NP-difficile et est difficile à approcher (voir en particulier [HHS98b] et [CC04]).

3.2 L'algorithme de Fomin-Kratsch-Woeginger

Le premier algorithme brisant la barrière $\Omega(2^n)$ pour déterminer un ensemble dominant de taille minimum d'un graphe à n sommets est dû à Fomin, Kratsch et Woeginger. Dans l'article [FKW04], les trois auteurs présentent un algorithme non trivial en $\mathcal{O}(1.9379^n)$. Ce premier article en initiera d'autres par la suite. En outre, les classes des graphes splits, bipartis et de degré maximum 3 y sont également considérées. Un autre résultat important donné dans cet article et reformulé dans le théorème 3.1, est l'improbabilité d'obtenir un algorithme sous-exponentiel pour DOMINATION. La notion de temps d'exécution sous-exponentiel est rappelée à la section 1.4.3. La classe SNP est l'ensemble des propriétés exprimables par $\exists S \forall x_1 \ldots \forall x_k \varphi(S, G, x_1, \ldots, x_k)$ où φ est une expression logique du premier ordre sans quantifieur, invoquant les variables x_i, l'entrée G et l'ensemble S (voir [Pap94]).

Théorème 3.1 ([FKW04] et [IPZ01])**.** *Si le problème* DOMINATION *peut être résolu en temps sous-exponentiel, alors les classes de complexité SNP et SUBSEXP satisfont SNP* $\subseteq$ *SUBSEXP.*
Cela est considéré comme une relation très improbable en théorie de la complexité.

L'algorithme proposé dans [FKW04] pour résoudre DOMINATION sur un graphe quelconque s'appuie principalement sur un résultat de Reed donnant une borne supérieure sur la taille d'un ensemble dominant minimum :

Théorème 3.2 ([Ree96])**.** *Tout graphe à n sommets de degré minimum au moins trois possède un ensemble dominant de taille au plus* $3n/8$.

L'idée de l'algorithme de Fomin *et al.* est de traiter récursivement (par l'intermédiaire de branchements et de réductions) tous les sommets de degré 0, 1 et 2 du graphe, puis d'énumérer tous les sous-ensembles de sommets de taille au plus $3n/8$ pour chacun des graphes résultants. L'article montre qu'il faut un temps $\mathcal{O}(1.9379^t)$ pour énumérer les $\binom{t}{3t/8}$ sous-ensembles d'un ensemble contenant t éléments, ce qui aboutit au temps d'exécution annoncé.

Théorème 3.3 ([FKW04]). *Un ensemble dominant de taille minimum d'un graphe à n sommets peut être déterminé en temps $\mathcal{O}^*(1.9379^n)$ (et espace polynomial).*

3.3 L'algorithme de Randerath-Schiermeyer

Randerath et Schiermeyer ont proposé dans [RS04] un algorithme en $\mathcal{O}(1.8899^n)$ pour résoudre DOMINATION. Ils donnent également dans cet article des algorithmes exponentiels non triviaux pour résoudre des problèmes tels que 3-coloration, ensemble dominant connexe minimum, ensemble stable dominant minimum, ensemble d'arête dominante minimum, ou encore couplage maximal minimum.

Leur algorithme pour résoudre le problème DOMINATION commence par traiter récursivement tous les sommets de degré 0 et 1 du graphe donné en entrée. De plus, ils travaillent sur un graphe connexe. Ceci est justifié par le fait qu'un ensemble dominant minimum d'un graphe G est égal à l'union des ensembles dominant minimum pour chacune des composantes connexes de G. Ce graphe connexe a pour degré maximum au moins trois, sinon, puisqu'il n'existe plus de sommets de degré 0 et 1 dans le graphe, si le degré maximum est égal à 2 alors le graphe connexe est un cycle et il est alors facile d'en déterminer un ensemble dominant de taille minimum en temps polynomial. L'algorithme de [RS04] est reproduit ci-après sous une forme simplifiée.

Les auteurs montrent ensuite que le temps d'exécution est borné par $\mathcal{O}(1.8899^n)$ et prouvent la validité de l'algorithme **MinDom**. En particulier, ils montrent que pour tout ensemble S, les sommets de T_S ont au plus deux voisins dans R_S, et que le graphe induit par R_S a pour degré maximum 1. Enfin ils établissent qu'il est suffisant de calculer un couplage de G_S pour déterminer un ensemble dominant minimum contenant S.

Théorème 3.4 ([RS04]). *Un ensemble dominant de taille minimum d'un graphe à n sommets peut être déterminé en temps $\mathcal{O}(1.8899^n)$ (et espace polynomial).*

3.4 L'algorithme de Fomin-Grandoni-Kratsch

On présente dans cette section le meilleur algorithme publié à ce jour pour résoudre DOMINATION. Cet algorithme simple, dû à Fomin, Grandoni et Kratsch [FGK05a] et présenté à ICALP 2005, est de type *Brancher & Réduire* et utilise la technique *Mesurer pour Conquérir* pour l'analyse de son temps d'exécution au pire des cas. L'algorithme proposé par Fomin *et al.* ne cherche pas à déterminer directement un ensemble dominant de taille minimum d'un graphe, mais résout le problème NP-difficile de la couverture d'ensemble minimum dont l'énoncé est donné ci-après. C'est déjà cette approche qu'avait utilisé Grandoni [Gra04, Gra06] pour obtenir un algorithme en $\mathcal{O}(1.9053^n)$ (en $\mathcal{O}(1.8026^n)$ au coût d'un espace exponentiel), sans utiliser une analyse aussi fine que le permettra par la suite la technique *Mesurer pour Conquérir*.

COUVERTURE D'ENSEMBLES (SET COVER)

Algorithme MinDom(G)
Entrée: un graphe $G = (V, E)$ **tel que** $\delta(G) \geq 2$ **et** $\Delta(G) \geq 3$**.**
Sortie : un ensemble dominant minimum de G**.**

Générer tous les sous-ensembles $S \subset V$ **de taille inférieure à** $\lceil n/3 \rceil$
si *l'un des sous-ensembles est un ensemble dominant* **alors**
  retourner le plus petit ensemble dominant trouvé
sinon
  Garder uniquement les sous-ensembles S **avec** $|N[S]| \geq 3|S|$ **qui sont maximaux au regard de la propriété** $|N[S \cup \{v\}]| < 3(|S| + 1)$**,** $\forall v \in V - S$
  pour tous les *ensembles* S *restants* **faire**
    $T_S \leftarrow N(S) - S$
    $R_S \leftarrow V - (S \cup T_S)(= V - N[S])$
    $E_S \leftarrow \emptyset$
    tant que $\exists v \in T_S$ *et* $v_1, v_2 \in R_S$ *tels que* $\{v, v_1\}, \{v, v_2\} \in E$ *et* v_1*,* v_2 *appartiennent à des composantes connexes différentes de* $G[R_S]$ **faire**
      $E_S \leftarrow E_S \cup \{v_1, v_2\}$
    Soit $G_S = (R_S, E_S)$
  pour tous les *graphes* G_S **faire**
    $\alpha_0(G_S) \leftarrow$ **taille maximum d'un couplage de** G_S
    $D_{G_S} \leftarrow$ **un ensemble dominant minimum de** G_S **de cardinalité** $|R_S| - \alpha_0(G_S)$ **qui est déterminé en temps polynomial grâce au couplage et à la structure du graphe** $G[R_S]$
  Soit $D = S \cup D_{G_S}$ **le plus petit ensemble parmi tous les ensembles** S
  retourner D

entrée : Un univers $\mathcal{U}$, une collection $\mathcal{S}$ de sous-ensembles de $\mathcal{U}$ et un entier $k \in \mathbb{N}$.
question : Existe-t-il $\mathcal{C} \subseteq \mathcal{S}$ tel que $\mathcal{U} = \bigcup_{C \in \mathcal{C}} C$ et $|\mathcal{C}| \leq k$?

Nous reproduisons ci-après l'algorithme **msc** de [FGK05a]. On note que cet algorithme est sensiblement le même que celui donné dans [Gra06]).

Algorithme msc($\mathcal{S}$)
Entrée: une collection d'ensembles $\mathcal{S}$ sur un univers $\mathcal{U}(\mathcal{S})$.
Sortie : la taille minimum d'une couverture d'ensembles.
  si $|\mathcal{S}| = 0$ **alors**
    retourner 0
  si $\exists S, R \in \mathcal{S}$ *tels que* $S \subseteq R$ **alors**
    retourner msc($\mathcal{S} \setminus S$)
  si $\exists u \in \mathcal{U}(\mathcal{S})$ *tel qu'il existe un unique* $S \in \mathcal{S}$ *tel que* $u \in S$ **alors**
    retourner 1 + msc(del($S, \mathcal{S}$))
  Choisir $S \in \mathcal{S}$ **de cardinalité maximum**
  si $|S| = 2$ **alors**
    Utiliser une réduction au problème du couplage maximum pour trouver une solution en temps polynomial
  sinon retourner $\min\{$**msc**$(\mathcal{S} \setminus S), 1 +$ **msc**$($**del**$(S, \mathcal{S}))\}$

L'opération **del** qui y est utilisée est définie par :

$$\text{del}(S, \mathcal{S}) = \{Z \ : \ Z = R \setminus S \neq \emptyset, \ R \in \mathcal{S}\}.$$

L'algorithme **msc** retourne la taille d'une couverture d'ensembles de taille minimum. Il est très facile de le modifier pour qu'il donne une couverture d'ensembles de taille minimum. Regardons comment il est possible d'utiliser cet algorithme pour résoudre Domination. La réduction polynomiale suivante peut être utilisée : étant donné un graphe $G = (V, E)$ pour lequel on souhaite calculer un ensemble dominant minimum, on construit l'instance de couverture d'ensembles $\mathcal{U} = V$, $\mathcal{S} = \{N[v] \ : \ v \in V\}$. Une couverture d'ensembles pour cette instance permet d'obtenir immédiatement un ensemble dominant minimum pour G. Comme la taille de l'instance ainsi construite est $|\mathcal{U}| + |\mathcal{S}| = 2|V|$, on en déduit que si l'on dispose d'un algorithme en $\mathcal{O}^*(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ pour résoudre Couverture d'ensembles alors on résout Domination en temps $\mathcal{O}^*(\alpha^{2n})$. Fomin *et al.* [FGK05a] ont montré que l'algorithme **msc** s'exécute en temps $\mathcal{O}(1.2355^{|\mathcal{U}|+|\mathcal{S}|})$ au pire des cas en utilisant un espace polynomial, et en $\mathcal{O}(1.2303^{|\mathcal{U}|+|\mathcal{S}|})$ en nécessitant un espace exponentiel.

Théorème 3.5 ([FGK05a])**.** *Un ensemble dominant de taille minimum d'un graphe à n sommets peut être déterminé en temps $\mathcal{O}(1.5263^n)$ et espace polynomial, ou en temps $\mathcal{O}(1.5137^n)$ et espace exponentiel.*

Pour parvenir à un tel temps d'exécution, les auteurs ont utilisé une technique qu'ils ont dénommée *Mesurer pour Conquérir*. Une analyse standard établirait seulement un temps d'exécution au pire des cas en $1.3803^{|\mathcal{U}|+|\mathcal{S}|}$ pour **msc** ; et permettrait donc de résoudre DOMINATION en temps $\mathcal{O}(1.9053^n)$ au pire des cas (voir [Gra06] et [FGK05a] pour une analyse standard de l'algorithme).

L'idée principale de Mesurer pour Conquérir est d'utiliser une mesure non standard (différente de $|\mathcal{U}| + |\mathcal{S}|$) puis une analyse par cas du temps d'exécution. Le choix d'une mesure différente de celle classiquement utilisée dans une analyse dite standard est motivé par le fait que deux instances de même mesure (dans le cas standard) ne correspondent pas nécessairement à deux problèmes de même difficulté. Par exemple, si une instance contient beaucoup d'ensembles avec un seul élément, ou si l'instance contient beaucoup d'éléments de fréquence 1 (la fréquence d'un élément est le nombre d'ensembles le contenant), alors il devient plus facile de la résoudre. Il convient donc, à chaque branchement, de mieux mesurer l'instance qualitativement, grâce à une mesure plus fine sur sa taille et sa difficulté, et quantitativement grâce à la distinction de sous-cas lors de l'analyse, on peut mieux mesurer la variation qu'un branchement induit sur la mesure de l'instance courante. Il faut noter que le prix à payer pour utiliser une telle technique est une analyse plus fastidieuse et parfois plus difficile.

Analyser l'algorithme **msc** par le biais de *Mesurer pour Conquérir* suggère d'affecter des poids différents aux ensembles de $\mathcal{S}$ et aux éléments de $\mathcal{U}$ en fonction de leur importance. Pour un ensemble $S \in \mathcal{S}$ ce sera au rapport de sa cardinalité, alors que pour un élément $u \in \mathcal{U}$ le poids dépendra de la fréquence de l'élément. On définit ainsi une nouvelle mesure sur la taille d'une instance par :

$$\mu(\mathcal{S},\mathcal{U}) = \sum_{i\geq 1} w_i n_i + \sum_{j\geq 1} v_j m_j$$

où $w_i, v_j \in [0,1]$ sont des poids, n_i est le nombre d'ensembles $S \in \mathcal{S}$ de cardinalité i et m_j est le nombre d'élements $u \in \mathcal{U}$ de fréquence j. On note que $\mu(\mathcal{S},\mathcal{U}) \leq |\mathcal{S}| + |\mathcal{U}|$.

Une étape importante demande de déterminer précisemment les poids w_i et v_j utilisés dans la mesure de sorte à minimiser le temps d'exécution. Pour cela, deux approches sont habituellements utilisées. La première est basée sur la notion de programmation et d'analyse de fonctions quasi-convexes. Cette technique a été donnée et démontrée par Eppstein dans [Epp06]. Elle requiert d'importants calculs mathématiques puisqu'elle fait appel à des calculs de descente de gradients pour déterminer la valeur minimum d'une fonction quasi-convexe. La seconde approche, plus simple, est basée sur une recherche locale randomisée [FGK05b]. La méthode de Newton est ensuite utilisée pour déterminer les solutions des récurrences. Dans ces deux approches, l'emploi d'un ordinateur est souhaitable puisqu'une puissance de calcul importante est nécessaire pour déterminer les poids. On note néanmoins, qu'étant donné les récurrences et les valeurs des poids, il est relativement aisé de vérifier que la solution annoncée est correcte.

Pour être complet, citons les récents travaux de van Rooij. Il a montré dans [Roo06]

que les résultats établis par Fomin *et al.* peuvent être légèrement améliorés. Pour cela, il propose une analyse plus détaillée de l'algorithme **msc** et considère davantage de règles de réductions. Il établit alors un algorithme en $\mathcal{O}(1.5138^n)$ au pire des cas (et espace polynomial) pour trouver un ensemble dominant minimum d'un graphe à n sommets. D'après nos connaissances, cet algorithme n'a pas fait l'objet de publication et dans la suite nous considérons l'algorithme de Fomin *et al.* comme le meilleur connu.

Aussi, comme indiqué dans [Roo06], l'algorithme **msc** proposé par Fomin *et al.* [FGK05a] permet de résoudre les problèmes de domination donnés à la figure 3.2 (voir la page suivante) en temps $\mathcal{O}(1.2355^{2n})$ ($\mathcal{O}(1.2355^{3n})$ pour le problème Ensemble Dominant Global) en utilisant des transformations simples vers le problèmes Couverture d'ensembles.

Nous ajoutons que le problème $W[2]$-difficile ([DF99]) Domination Rouge et Bleue peut aussi être résolu par l'algorithme **msc**, en temps $\mathcal{O}(1.2355^n)$, grâce à une réduction à Couverture d'ensembles.

Domination Rouge et Bleue

entrée : Un graphe $G = (V, E)$ et une partition de V en deux ensembles R et B.

question : Chercher un ensemble $D \subseteq R$ de taille minimum tel que pour tout $v \in B$, il existe $u \in D$ avec $v \in N(u)$.

3.5 Résultats connexes

Dans cette dernière section, on cite quelques résultats fortement reliés au problème de la domination.

Tout d'abord, un certain nombre de travaux ont été effectués pour obtenir des algorithmes plus rapides que dans le cas général pour quelques classes de graphes. Dans [FKW04] sont proposés des algorithmes pour les graphes splits, bipartis et de degré maximum trois respectivement en $\mathcal{O}(1.4143^n)$, $\mathcal{O}(1.7321^n)$ et $\mathcal{O}(1.5144^n)$. Puisque Fomin et Høie [FH06] donnent un algorithme en $\mathcal{O}(1.2010^n)$ pour résoudre Domination sur les gaphes de degré maximum trois, et que le meilleur algorithme connu pour résoudre Domination s'exécute en $\mathcal{O}(1.5263^n)$, seul l'algorithme en $\mathcal{O}(1.4143^n)$ pour les graphes splits reste plus rapide que les meilleurs algorithmes connus. Néanmoins, comme nous le notons à la section 4.10, en combinant les idées de [FKW04] et le résultat de [FGK05a], il est possible d'obtenir un algorithme pour les graphes splits en temps $\mathcal{O}(1.2355^n)$ et espace polynomial, ou $\mathcal{O}(1.2303^n)$ et espace exponentiel.

Concernant le nombre d'ensembles dominants minimaux (par inclusion) qu'un graphe à n sommets peut contenir, Fomin *et al.* [FGPS05] ont montré en utilisant une approche algorithmique et la technique *Mesurer pour Conquérir* que ce nombre est au plus 1.7697^n

Ensemble Dominant Orienté

entrée : Un graphe orienté $G = (V, A)$.

question : Chercher un ensemble $D \subseteq V$ de taille minimum tel que pour tout $v \in V \setminus D$, il existe $u \in D$ avec $(u, v) \in A$.

Ensemble Dominant Total

entrée : Un graphe $G = (V, E)$.

question : Chercher un ensemble $D \subseteq V$ de taille minimum tel que pour tout $v \in V$, il existe $u \in D$ avec $v \in N(u)$.

Ensemble Dominant k-Distant

entrée : Un graphe $G = (V, E)$.

question : Chercher un ensemble $D \subseteq V$ de taille minimum tel que pour tout $v \in V \setminus D$, il existe $u \in D$ avec $d(u, v) \leq k$.

Ensemble Dominant Fort (resp. Ensemble Dominant Faible)

entrée : Un graphe $G = (V, E)$.

question : Chercher un ensemble $D \subseteq V$ de taille minimum tel que pour tout $v \in V \setminus D$, il existe $u \in D$ avec $v \in N(u)$ et $d(u) \geq d(v)$ (respectivement $d(u) \leq d(v)$).

Ensemble Dominant Global

entrée : Un graphe $G = (V, E)$.

question : Chercher un ensemble $D \subseteq V$ de taille minimum tel que D est un ensemble dominant pour G et pour son complémentaire $\overline{G}$.

Fig. 3.2 – Définition de quelques problèmes pour lesquels il existe une transformation simple vers le problème Couverture d'ensembles [Roo06].

et que tous ces ensembles dominants peuvent être listés en temps $\mathcal{O}(1.7697^n)$. De plus, Kratsch a donné un graphe pour lequel le nombre d'ensembles dominants minimaux est $15^{n/6} > 1.5704^n$ [Kra05]. Ce graphe est constitué de copies disjointes d'octahedrons – voir la figure 3.3.

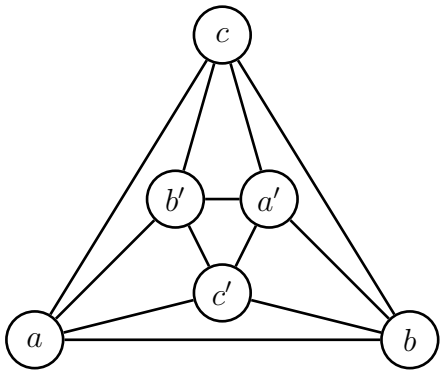


Fig. 3.3 – Un octahedron ayant pour ensemble de sommets $\{a, b, c, a', b', c'\}$ et possèdant 15 ensembles dominants minimaux : $\{a, a'\}$, $\{a, b\}$, $\{a, c\}$, $\{a, b'\}$, $\{a, c'\}$, $\{a', b\}$, $\{a', c\}$, $\{a', b'\}$, $\{a', c'\}$, $\{b, c\}$, $\{b, b'\}$, $\{b, c'\}$, $\{c, b'\}$, $\{c, c'\}$ et $\{b', c'\}$.

Fomin *et al.* conjecturent que le graphe proposé par Kratsch est celui avec le nombre maximum d'ensembles dominants maximal.

Dans [FGPS05], il est également donné un algorithme s'exécutant en $\mathcal{O}(1.5780^n)$ pour résoudre la version pondérée de DOMINATION.

Le problème NOMBRE DOMINANT (en anglais DOMATIC NUMBER) demande de trouver le nombre maximum de sous-ensembles en lequel les sommets d'un graphe peuvent être partitionnés de sorte que chaque ensemble soit un ensemble dominant du graphe. Les auteurs de [FGPS05] donnent un algorithme pour résoudre ce problème en temps $\mathcal{O}(2.8805^n)$. En utilisant la technique Inclusion-Exclusion, Björklund et Husfeld ainsi que Koivisto montrent qu'il est possible de résoudre ce même problème en temps $\mathcal{O}^*(2^n)$ et espace exponentiel [BH06, Koi06]. De plus, Björklund et Husfeld proposent un algorithme en temps $\mathcal{O}(2.8718^n)$ ne nécessitant qu'un espace polynomial.

Finalement, dans l'article [FS07], Fomin et Stepanov montrent qu'il est possible de compter en temps $\mathcal{O}(1.5535^n)$ les ensembles dominants pondérés de coût minimum d'un graphe.

Chapitre 4

Ensemble dominant sur quelques classes de graphes

Le chapitre 3 nous a permis d'exposer un des problèmes de graphes parmi les plus importants : le problème DOMINATION. Ce problème, listé comme étant NP-complet par Garey et Johnson [GJ79], reste NP-complet pour de nombreuses classes de graphes (voir la Table 3.1).

Le meilleur algorithme publié à ce jour pour résoudre DOMINATION demande un temps d'exécution en $\mathcal{O}(1.5263^n)$ et espace polynomial ou $\mathcal{O}(1.5137^n)$ et espace exponentiel. Cet algorithme, présenté dans la section 3.4, est dû à Fomin, Grandoni et Kratsch [FGK05a] et est basé sur une réduction au problème COUVERTURE D'ENSEMBLES. Il est naturel de se demander s'il est possible d'obtenir un meilleur algorithme exponentiel lorsqu'on se restreint à des classes de graphes où le problème reste NP-complet. Cet angle d'attaque a déjà eu beaucoup de succès depuis les années 1980 afin de trouver des algorithmes polynomiaux sur certaines classes de graphes pour résoudre des problèmes en général NP-complets. Bien souvent la connaissance de quelques propriétés structurelles sur les graphes considérés aide à exhiber de tels algorithmes. Par exemple, le problème DOMINATION admet un algorithme polynomial si l'entrée est un graphe d'intervalles [Cha98], un graphe sans AT [Kra00] ou même un graphe de co-comparabilité [KS93]. C'est cette approche qui motive l'étude d'algorithmes exponentiels pour quelques classes de graphes. Pour des classes de graphes pour lesquelles le problème reste NP-complet, on se demande alors s'il est possible d'obtenir un algorithme plus rapide que dans le cas général. Là aussi, on cherche à exploiter les connaissances apportées par la restriction à une famille de graphes particuliers.

En 2004, Fomin, Kratsch et Woeginger [FKW04] se sont déjà intéréssés à des algorithmes pour résoudre DOMINATION sur quelques classes de graphes particulières. Comme déjà indiqué à la section 3.5, les auteurs de [FKW04], proposent un algorithme en temps $\mathcal{O}(1.9379^n)$ pour le cas général, et des algorithmes en temps $\mathcal{O}(1.4143^n)$, $\mathcal{O}(1.7321^n)$ et $\mathcal{O}(1.5144^n)$ pour résoudre DOMINATION respectivement sur les graphes splits, bipartis et de degré au plus 3 ; les algorithmes pour les graphes splits et bipartis demandant un espace exponentiel. Le résultat pour les graphes de degré au plus 3 a ensuite été amélioré, puisqu'en 2006 Fomin et Høie ont obtenu un algorithme en $\mathcal{O}(1.2010^n)$ et espace exponentiel

pour résoudre DOMINATION sur cette classe. Enfin très récemment, Dorn a présenté à la conférence WG 2007, un algorithme qui résout DOMINATION sur les graphes planaires en temps $\mathcal{O}^*(3^{tw})$ [Dor07a], où tw représente la largeur arborescente du graphe. Nous définirons cette notion à la prochaine section. Dorn montre dans [Dor07b] que cela implique un algorithme sous-exponentiel en temps $\mathcal{O}^*(15.8895^{\sqrt{n}})$ sur cette même classe.

Dans ce chapitre, nous allons proposer une approche générale qui nous permettra d'obtenir les résultats présentés à la Table 4.1.

Tab. 4.1 – Algorithmes présentés dans ce chapitre pour résoudre DOMINATION.

classe de graphe	temps d'exécution	
graphes c-denses	$\mathcal{O}(1.2303^{n(1+\sqrt{1-2c})})$	(Section 4.3)
graphes cordaux	$\mathcal{O}(1.4173^n)$	(Section 4.4)
graphes 4-cordaux	$\mathcal{O}(1.4913^n)$	(Section 4.5)
graphes faiblement cordaux	$\mathcal{O}(1.4842^n)$	(Section 4.6)
graphes cercles	$\mathcal{O}(1.4956^n)$	(Section 4.7)
graphes bipartis	$\mathcal{O}(1.4143^n)$	(Section 4.8)
graphes quelconques	$\mathcal{O}(1.7088^n)$	(Section 4.8)

Dans la suite de ce chapitre, nous développons et utilisons deux stratégies générales. La première que nous nommerons « beaucoup de sommets de grand degré » est reliée à l'algorithme de Fomin *et al.* pour résoudre le problème COUVERTURE D'ENSEMBLES [FGK05a]. En fait elle exploite le meilleur algorithme connu pour résoudre COUVERTURE D'ENSEMBLES. Elle nous servira directement pour obtenir un algorithme pour DOMINATION sur les graphes c-denses. La seconde stratégie sera basée sur la largeur arborescente d'un graphe. Lorsque celle-ci sera grande nous ferons appel à l'approche « beaucoup de sommets de grand degré », sinon nous appliquerons un algorithme utilisant une décomposition arborescente pour résoure DOMINATION. Cette dernière approche permettra d'obtenir des algorithmes pour les graphes cordaux, cercles, 4-cordaux et faiblement cordaux.

Afin d'appliquer cette dernière stratégie, il nous faudra au préalable établir une borne (linéaire) supérieure de la largeur arborescente d'un graphe en fonction de son degré maximum. L'obtention de telles bornes est intéressante en soi. De plus l'approche que nous développons pourrait s'appliquer à d'autres classes de graphes et à d'autres problèmes, tel que par exemple le problème ENSEMBLE STABLE DOMINANT qui demande de trouver un ensemble dominant de plus petite taille qui soit aussi un ensemble stable.

L'approche que nous utiliserons pour obtenir un algorithme pour les graphes bipartis sera quant à elle très différente et nécessitera l'utilisation d'un pré-traitement sur une partie du graphe combiné avec de la Programmation Dynamique.

4.1 Préliminaires

Parmi les outils que nous utiliserons dans la suite et qui n'ont pas été présentés dans les chapitres précédents se trouvent les décompositions arborescentes et la largeur arborescente d'un graphe. Ces notions ont été introduites par Robertson et Seymour [RS86] et sont rappelées ci-après. Nous les illustrons par l'exemple de la figure 4.1.

Définition 4.1.1 (décomposition arborescente)**.** Soit $G = (V, E)$ un graphe. Une *décomposition arborescente* de G est une paire $(\{X_i : i \in I\}, T)$ où chaque ensemble X_i, $i \in I$, est un sous-ensemble de V, et T est un arbre ayant pour nœuds les éléments de I tel que les propriétés suivantes sont vérifiées :

1. $\cup_{i \in I} X_i = V$;
2. $\forall \{u, v\} \in E$, $\exists i \in I$ tel que $\{u, v\} \subseteq X_i$;
3. $\forall i, j, k \in I$, si j est sur un chemin de i à k dans T alors $X_i \cap X_k \subseteq X_j$.

Les ensembles X_i, $i \in I$, sont habituellement appelés *sacs* de la décomposition arborescente. La largeur d'une décomposition arborescente est égale à $\max_{i \in I} |X_i| - 1$.

Définition 4.1.2 (Largeur arborescente)**.** La *largeur arborescente* (en anglais *treewidth*) d'un graphe G est la largeur minimum parmi toutes les décompositions arborescentes de G. On la note $tw(G)$. Une décomposition arborescente est dite *optimale* si sa largeur est précisément égale à $tw(G)$.

Remarque. *Etant donné un graphe $G = (V, E)$, il existe toujours une décomposition arborescente du graphe. En effet, un arbre possédant un unique nœud avec tous les sommets V de G donne une décomposition arborescente triviale de G et de largeur $|V| - 1$.*

Définition 4.1.3 (Décomposition arborescente commode)**.** Une *décomposition arborescente commode* $(\{X_i : i \in I\}, T)$ est une décomposition arborescente qui satisfait les propriétés suivantes :

1. chaque nœud dans T possède au plus deux fils ;
2. Si un nœud i a deux fils j et k, alors $X_i = X_j = X_k$ (on appelle le nœud i un nœud *joint*) ;
3. Si un nœud i a un seul fils j, alors l'une des deux conditions suivantes est vérifiée :
   (a) $|X_i| = |X_j| + 1$ et $X_j \subset X_i$ (i est appelé *nœud insérant*) ;
   (b) $|X_i| = |X_j| - 1$ et $X_i \subset X_j$ (i est appelé *nœud oubliant*).

Lemme 4.1 ([Klo94])**.** *Pour toute constante k, étant donné une décomposition arborescente de largeur k avec N nœuds pour un graphe G, il est possible de trouver une décomposition arborescente commode de G de largeur k avec au plus $4N$ nœuds en temps $\mathcal{O}(n)$, où n est le nombre de sommets de G.*

Le lemme 4.2 donne deux résultats bien connus sur la largeur arborescente de graphes.

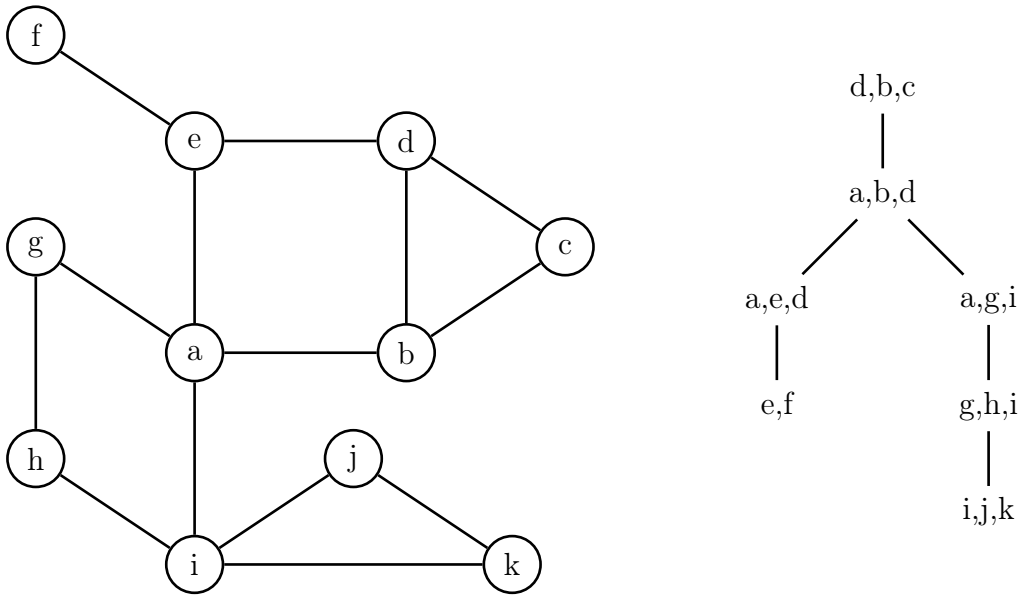


Fig. 4.1 – Exemple d'un graphe G et d'une décomposition arborescente optimale de G de largeur 2.

Lemme 4.2 ([BM93])**.** *La largeur arborescente d'un arbre est égale à 1. De plus, les graphes de largeur arborescente* 1 *sont précisément les forêts. Une clique à n sommets a pour largeur arborescente $n-1$.*

L'étude des décompositions arborescentes de graphes a fait l'objet d'intenses travaux. Nous référons le lecteur à [Bod93, Bod98, Bod05, Heg06, Klo93] pour une vue plus large de ces outils.

Le théorème suivant nous servira dans la suite de ce chapitre. Il donne un algorithme qui résout Domination en fonction de la largeur arborescente d'un graphe. Les idées fondamentales utilisées dans cet algorithme seront données dans la preuve du théorème 4.14 (page 80).

Théorème 4.3 ([ABFKN02, AN02])**.** *Étant donné un graphe G et une décomposition arborescente de largeur au plus ℓ du graphe, il existe un algorithme qui résout le problème* Domination *en temps $4^{\ell}n^{\mathcal{O}(1)}$.*

4.2 Approche générale

Les algorithmes des sections 4.3 à 4.7 résolvent le problème Domination en exploitant deux propriétés du graphe G donné en entrée :

- G a beaucoup de sommets de grand degré, c'est-à-dire $|\{v \in V : \ d(v) \geq t-2\}| \geq t$ pour un certain entier $t > 0$
  (voir la section 4.2.1 et le théorème 4.4) ;

– il existe une constante positive r vérifiant $tw(H) \leq r \cdot \Delta(H)$ pour tous les sous-graphes induits H de G, et de plus il existe un algorithme calculant en temps polynomial une décomposition arborescente de H de largeur au plus $r \cdot \Delta(H)$. (En fait, le temps d'exécution ne doit pas excéder $\mathcal{O}(4^{r \cdot \Delta(H)} n^{\mathcal{O}(1)}) = \mathcal{O}^*(4^{r \cdot \Delta(H)})$.)
(voir la section 4.2.2 et le théorème 4.6).

Les méthodes décrites ci-après utilisent et combinent ces deux propriétés pour établir des algorithmes exponentiels résolvant DOMINATION sur diverses classes de graphes pour lesquelles DOMINATION est NP-complet. Les bornes supérieures sur le temps d'exécution des algorithmes obtenus sont inférieures à celle du meilleur algorithme connu et publié pour les graphes en général, c'est-à-dire meilleurs que $\mathcal{O}(1.5137^n)$.

4.2.1 Beaucoup de sommets de grand degré

Supposons qu'on dispose d'un algorithme qui résout le problème DOMINATION sur un graphe quelconque et qui est basé sur un algorithme résolvant le problème COUVERTURE D'ENSEMBLES en temps $\mathcal{O}^*(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ (où $\mathcal{U}$ est l'univers et $\mathcal{S}$ un ensemble de sous-ensembles de l'univers). Nous montrons au théorème 4.4 qu'un graphe possédant suffisamment de sommets de grand degré permet d'obtenir un algorithme plus rapide que $\mathcal{O}^*(\alpha^{2n})$. En particulier, à la section 3.4 nous avons présenté le meilleur algorithme connu pour résoudre DOMINATION et cet algorithme est justement basé sur un algorithme qui résout COUVERTURE D'ENSEMBLES en temps $\mathcal{O}(1.2303^{|\mathcal{U}|+|\mathcal{S}|})$, c'est-à-dire pour lequel $\alpha = 1.2303$.

Théorème 4.4. *Supposons qu'on dispose d'un algorithme qui s'exécute en temps $\mathcal{O}^*(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ pour calculer une couverture d'ensembles minimum de $(\mathcal{U}, \mathcal{S})$.*

Soit une fonction $t : \mathbb{N} \to \mathbb{R}^+$. Alors il existe un algorithme pour résoudre le problème DOMINATION *en temps $\mathcal{O}^*(\alpha^{2n-t(n)})$ sur n'importe quel graphe $G = (V, E)$ vérifiant $|\{v \in V : \ d(v) \geq t(n) - 2\}| \geq t(n)$, avec $n = |V|$.*

Démonstration. Soit $G = (V, E)$ un graphe vérifiant $|\{v \in V : \ d(v) \geq t(n) - 2\}| \geq t(n)$. Posons $t = t(n) \geq 0$ et $T = \{v \in V : \ d(v) \geq t - 2\}$. On a donc $|T| \geq t$. Pour chaque ensemble dominant minimum D de G, soit au moins un sommet de T appartient à D, soit aucun sommet de T n'est dans D ($T \cap D = \emptyset$).

Nous nous servons de cette observation pour déterminer un ensemble dominant minimum de G en utilisant les branchements en deux sortes de sous-problèmes : le sous-problème « $v \in D$ » (branchement effectué pour tout $v \in T$), et le sous-problème « $T \cap D = \emptyset$ ». Dans ces deux cas, nous utilisons ensuite un algorithme en temps $\mathcal{O}^*(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ pour résoudre le problème COUVERTURE D'ENSEMBLES.

Rappelons la réduction qui transforme une instance $G = (V, E)$ du problème DOMINATION en une instance $(\mathcal{U}, \mathcal{S})$ de COUVERTURE D'ENSEMBLES : l'univers est défini par $\mathcal{U} = V$ et l'ensemble $\mathcal{S}$ par $\{N[u] : \ u \in V\}$; ainsi $|\mathcal{U}| + |\mathcal{S}| = 2n$. Cette réduction standard est également celle utilisée par Fomin *et al.* dans [FGK05a].

Par conséquent, si on supprime x éléments dans $\mathcal{U}$ et y sous-ensembles dans S de l'instance de COUVERTURE D'ENSEMBLES, le temps d'exécution pour résoudre un tel sous-problème devient $\mathcal{O}^*(\alpha^{2n-(x+y)})$.

Regardons les deux types de sous-problèmes obtenus lors d'un branchement. Pour tout sommet $v \in T$, on prend le sommet v dans l'ensemble dominant minimum et on exécute ensuite l'algorithme en temps $\mathcal{O}^*(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ pour résoudre Couverture d'ensembles sur une instance de taille au plus $2n-(d(v)+1)-1 \leq 2n-t$. En effet, on supprime de l'univers $\mathcal{U}$ les éléments de $N[v]$ et on supprime de l'ensemble $\mathcal{S}$ l'ensemble correspondant à v, c'est-à-dire l'ensemble $N[v]$. Et on branche dans le sous-problème « éliminer les sommets de T » : dans ce cas on obtient une instance pour Couverture d'ensembles de taille au plus $2n - |T| = 2n - t$ puisque pour tout $v \in T$ on supprime de $\mathcal{S}$ les ensembles correspondant à chacun des sommets v de T. □

En combinant le théorème que nous venons d'établir avec l'algorithme pour Couverture d'ensembles de [FGK05a] (voir section 3.4), nous obtenons le corollaire suivant :

Corollaire 4.5. *Soit une fonction $t : \mathbb{N} \to \mathbb{R}^+$. Alors il existe un algorithme qui s'exécute en temps $\mathcal{O}(1.2303^{2n-t(n)})$ et qui résout* Domination *sur tout graphe G qui vérifie $|\{v \in V : \ d(v) \geq t(n) - 2\}| \geq t(n)$.*

4.2.2 Approche basée sur la largeur arborescente

La technique que nous introduisons maintenant est basée sur la largeur arborescente d'un graphe. La notion de largeur arborescente a déjà été utilisée précédemment pour concevoir des algorithmes exponentiels. L'état de l'art donné dans [FGK05b] liste quelques unes des utilisations. Néanmoins, l'approche que nous donnons ici semble nouvelle et pourrait s'étendre à d'autres problèmes.

Supposons l'existence d'une classe de graphes pour laquelle $tw(G) \leq r\Delta(G)$ pour tout graphe de la classe, avec r une constante dépendant de la classe considérée. L'idée que nous allons développer est que de tels graphes ont beaucoup de sommets de grand degré, ou sinon leur degré maximum est petit, et dans ce cas leur largeur arborescente est petite. Dans le premier cas, on utilise l'algorithme décrit juste précédemment à la section 4.2.1. Dans le second cas, on fait appel à l'algorithme d'Alber *et al.* [ABFKN02] qui détermine un ensemble dominant minimum en temps $4^\ell n^{\mathcal{O}(1)}$ (voir le théorème 4.3). Afin de balancer le temps d'exécution de ces deux approches, un paramètre que nous appelons λ sera choisi. Sa valeur est précisée dans l'énoncé du théorème 4.6.

Théorème 4.6. *Supposons qu'on dispose d'un algorithme qui s'exécute en temps $\mathcal{O}^*(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ pour calculer une couverture d'ensembles minimum de $(\mathcal{U}, \mathcal{S})$.*

Soit r une constante positive. Soit $\mathcal{G}$ une classe de graphes héréditaire telle que $tw(G) \leq r \cdot \Delta(G)$ pour tous les graphes $G \in \mathcal{G}$. Supposons de plus qu'il existe un algorithme qui pour n'importe quel graphe $G \in \mathcal{G}$ calcule en temps polynomial une décomposition arborescente de largeur au plus $r \cdot \Delta(G)$ en temps polynomial.

Alors il existe un algorithme qui résout Domination *en temps*

$$\max(\alpha^{2n-\lambda n/r}, 4^{(r+1)\lambda n/r})n^{\mathcal{O}(1)}$$

pour tout graphe de $\mathcal{G}$, avec $\lambda = \lambda(r, \alpha) = \frac{2}{\frac{1+d}{r}+d}$ et $d = 1/\log_4(\alpha)$.

Algorithme Dom-GrandDeg-PetitTw(un graphe $G = (V, E)$)
Entrée: un graphe G satisfaisant les conditions du théorème 4.6.
Sortie : le nombre dominant $\gamma(G)$ de G.

$\lambda \leftarrow \lambda(r, \alpha)$ `/* la valeur de` λ `est donnée par le théorème 4.6 */`
$X \leftarrow \{u \in V : d(u) \geq \lambda n/r\}$
si $|X| \geq \lambda n/r$ **alors**
  utiliser l'algorithme du théorème 4.4 et retourner le résultat
sinon
  utiliser l'algorithme du théorème 4.3 et retourner le résultat

Démonstration. La procédure décrite dans cette preuve est également donnée par l'algorithme **Dom-GrandDeg-PetitTw**. Tout d'abord, l'algorithme construit l'ensemble X contenant tous les sommets du graphe G ayant un degré plus grand que $\lambda n/r$.

Par définition de X, pour tout $v \in X$, on a $d(v) > \lambda n/r$. Ainsi, si $|X| \geq \lambda n/r$, alors on applique l'algorithme du théorème 4.4 et donc un ensemble dominant de cardinalité minimum peut être trouvé en temps $\alpha^{2n-\lambda n/r} n^{\mathcal{O}(1)}$.

Sinon on a les relations $|X| < \lambda n/r$ et $\Delta(G - X) < \lambda n/r$. Le graphe $G - X$ appartient à la classe héréditaire $\mathcal{G}$ puisque c'est un sous-graphe induit de G et donc la relation $tw(G - X) \leq r\Delta(G - X)$ est vérifiée. Il s'ensuit que $tw(G - X) < r\lambda n/r = \lambda n$. On sait que si l'on ajoute un sommet à un graphe, alors sa largeur arborescente augmente d'au plus 1, puisqu'il suffit d'ajouter ce sommet à tous les ensembles correspondant à un nœud dans l'arbre. D'où $tw(G) \leq tw(G - X) + |X| < \lambda n + \lambda n/r = (r + 1)\lambda n/r$. L'algorithme calcule alors une décomposition arborescente de largeur au plus $(r + 1)\lambda n/r$ en temps polynomial, puis il utilise l'algorithme du théorème 4.3. Ainsi, dans ce cas, un ensemble dominant minimum peut être calculé en temps $4^{tw(G)} n^{\mathcal{O}(1)} = 4^{(r+1)\lambda n/r} n^{\mathcal{O}(1)}$.

Finalement, un ensemble dominant de taille minimum de G peut être calculé en temps $\max(\alpha^{2n-\lambda n/r}, 4^{(r+1)\lambda n/r}) n^{\mathcal{O}(1)}$. □

Là aussi, si on combine ce théorème avec l'algorithme pour Couverture d'ensembles de [FGK05a] (voir section 3.4), nous obtenons le corollaire suivant :

Corollaire 4.7. *Il existe un algorithme qui résout le problème* Domination *en temps* $\max(1.2303^{2n-\lambda n/r}, 4^{(r+1)\lambda n/r}) n^{\mathcal{O}(1)}$ *sur n'importe quel graphe G vérifiant $tw(G) \leq r\Delta(G)$, avec r une constante, $\lambda = \lambda(r) = \frac{2}{\frac{1+d}{r}+d}$ et $d = 1/\log_4(1.2303)$.*

La table 4.2 donne le temps d'exécution de l'algorithme pour différentes valeurs de la constante r. Bien sûr, tout algorithme permettant de résoudre Couverture d'ensembles plus rapidement que l'algorithme proposé par Fomin *et al.* dans [FGK05a] (actuellement le meilleur connu) permettrait immédiatement d'améliorer le temps d'exécution de notre algorithme **Dom-GrandDeg-PetitTw**.

Dans les sections suivantes nous montrons comment les approches générales que nous venons d'établir peuvent s'appliquer aux graphes denses (section 4.3), aux graphes cordaux

Tab. 4.2 – Temps d'exécution de l'algorithme du corollaire 4.7 pour quelques valeurs de r.

valeur de r	1.5	2	2.5
temps d'exécution	$\mathcal{O}(1.4787^n)$	$\mathcal{O}(1.4842^n)$	$\mathcal{O}(1.4882^n)$
valeur de r	3	4	5
temps d'exécution	$\mathcal{O}(1.4913^n)$	$\mathcal{O}(1.4956^n)$	$\mathcal{O}(1.4985^n)$

(section 4.4), aux graphes 4-cordaux (section 4.5), aux graphes faiblement cordaux (section 4.6) et aux graphes cercles (section 4.7). Nous donnons aussi explicitement quelques algorithmes calculant une décomposition arborescente de largeur au plus $r\Delta(G)$ et nous donnerons la valeur de la constante r dépendant des classes de graphes considérées.

4.3 Graphes c-denses

De nombreux problèmes restent NP-complets lorsqu'on se restreint à des graphes ayant un grand nombre d'arêtes. Par exemple Schiermeyer montre dans [Sch95] que les problèmes Ensemble Stable, Circuit Hamiltonien et Chemin Hamiltonien restent NP-complets sur ce type de graphes.

Nous commencerons par montrer que le problème Domination l'est également pour les graphes c-denses. Nous donnerons ensuite un algorithme exponentiel qui résout ce problème sur cette classe de graphes. L'algorithme illustrera l'utilisation de l'approche « beaucoup de sommets de grand degré » que nous avons présenté à la section précédente.

Débutons par une définition des graphes denses, la notion de graphes c-denses.

Définition 4.3.1. Un graphe $G = (V, E)$ est appelé *c-dense*, ou simplement *dense* s'il n'y a pas d'ambiguïté, si $|E| \geq cn^2$ où c est une constante vérifiant $0 < c < 1/2$.

Une façon commode de montrer qu'un problème de graphes est NP-complet pour les graphes c-denses, pour n'importe quelle constante c avec $0 < c < 1/2$, est de construire un graphe G' en ajoutant une clique C suffisamment grande comme une nouvelle composante connexe du graphe original G. Le graphe G' ainsi obtenu est un graphe c-dense. Cette simple réduction peut être utilisée pour montrer que de nombreux problèmes de graphes NP-complets restent difficiles à résoudre sur la classe des graphes c-denses. Parmi ces problèmes, il y a le problème Ensemble Stable (pour lequel $\alpha(G') = \alpha(G) + 1$), Partition en Cliques ($\kappa(G') = \kappa(G) + 1$), Couverture de Sommets ($\tau(G') = \tau(G) + |C| - 1$), Ensemble de Sommets en Retour (en anglais, Feedback Vertex Set ; $\mathrm{FVS}(G) = \mathrm{FVS}(G') + |C| - 2$) ou encore Complétion Minimale (en anglais, Minimum Fill-In ; $\mathrm{MFI}(G) = \mathrm{MFI}(G')$).

Le théorème suivant montre que le problème DOMINATION est également NP-complet lorsqu'on se restreint aux graphes c-denses.

Théorème 4.8. *Pour n'importe quelle constante c avec $0 < c < 1/2$, le problème de décider si un graphe c-dense admet un ensemble dominant de taille au plus k est NP-complet, même si le graphe c-dense est aussi un graphe split.*

Démonstration. Soit c une constante telle que $0 < c < 1/2$. Le problème DOMINATION appartient à NP et donc le problème DOMINATION sur les graphes c-denses est aussi dans NP.

Dans [Ber84] il est montré que le problème qui demande de déterminer si un graphe split a un ensemble dominant de taille au plus k est NP-complet. Nous allons donner une réduction polynomiale du problème DOMINATION sur les graphes splits vers le problème DOMINATION sur les graphes c-denses splits.

Soit k un entier et soit $G = (I \cup C, E)$ un graphe split où I et C forment une partition des sommets de G tels que I induit un ensemble stable et C induit une clique dans G. Tout d'abord, on construit un graphe c-dense $G' = (V', E')$ avec $E' \geq c \cdot |V'|^2$. Ce graphe $G' = (V', E')$ est obtenu du graphe $G = (C \cup I, E)$ en ajoutant une clique C' de taille $|C'| = \lceil (1 + 4c|I \cup C| + \sqrt{1 + 8c|I \cup C|(1 + |I \cup C|)})/(2 - 4c) \rceil$ à G et en ajoutant toutes les arêtes avec une extrémité dans C et l'autre dans C'. La taille de C' est donnée par la résolution de l'inéquation $|C'| \cdot (|C'| - 1)/2 > c \cdot (|I \cup C| + |C'|)^2$.

Ceci garantit que G' est un graphe split avec une partition des sommets de V' en un ensemble stable I et une clique $C \cup C'$. De plus, le nombre d'arêtes de G' est plus grand que $c(|I \cup C| + |C'|)^2$ et donc G' est bien un graphe c-dense.

On montre maintenant que le graphe G a un ensemble dominant de taille au plus k si et seulement si le graphe G' a un ensemble dominant de taille au plus k.

Tout d'abord, supposons que G' a un ensemble dominant D de taille au plus k. Puisque $N_{G'}[x'] \subseteq N_{G'}[x]$ pour tout $x' \in C'$ et pour tout $x \in C$, nous pouvons remplacer chaque sommet de C' qui appartient à D par un sommet de C. De cette façon on obtient un ensemble dominant $D' \subseteq I \cup C$ de G' tel que $|D'| \leq k$. Par conséquent D' est aussi un ensemble dominant de G.

Inversement, supposons que D est un ensemble dominant de G de taille au plus k. Si D contient au moins un sommet de C, alors D est aussi un ensemble dominant de G' car chaque sommet de C' est adjacent à tous les sommets de C. Sinon, D ne contient aucun sommet de C et donc chaque sommet de C a au moins un voisin dans $D \cap I$. Dans ce second cas, on remplace un sommet $s \in D \cap I$ quelconque par l'un de ses voisins $t \in C$ et l'on considère l'ensemble $D' = (D \setminus \{s\}) \cup \{t\}$. Ensuite, D' est un ensemble dominant de G puisque $N_G[s] \subseteq N_G[t]$. De plus, comme D' contient un sommet de C, D' est aussi un ensemble dominant de G'. Finalement, dans chacun des cas, G' a un ensemble dominant de taille au plus k.

En conclusion, le problème qui demande de décider si un graphe c-dense qui est également un graphe split a un ensemble dominant de taille au plus k est un problème de décision NP-complet. □

Nous venons de montrer que même lorsqu'on se restreint aux graphes contenant beaucoup d'arêtes, comme c'est le cas pour les graphes c-denses, le problème DOMINATION reste NP-complet. C'est pourquoi nous nous intéressons pour déterminer un algorithme exact et exponentiel permettant de résoudre ce problème plus rapidement que dans le cas général, en se servant du fait que le graphe a un grand nombre d'arêtes.

L'idée principale de notre algorithme pour déterminer un ensemble dominant de taille minimum demande de considérer un grand sous-ensemble de sommets de grand degré. Le lemme suivant nous indique que si un graphe a suffisamment d'arêtes alors il possède aussi un grand sous-ensemble de sommets de grand degré.

Lemme 4.9. *Pour deux entiers t et t' fixés, $1 \leq t \leq n$, $1 \leq t' \leq n-1$, n'importe quel graphe $G = (V, E)$ avec $|E| \geq 1 + \dfrac{(t-1)(n-1) + (n-t+1)(t'-1)}{2}$ possède un sous-ensemble $T \subseteq V$ qui vérifie*

(i) $|T| \geq t$,

(ii) pour tout $v \in T$, $d(v) \geq t'$.

Démonstration. Soit t et t' tels que $1 \leq t \leq n$ et $1 \leq t' \leq n-1$. Soit un graphe $G = (V, E)$ pour lequel il n'existe pas d'ensemble T respectant les propriétés énoncées dans le lemme. Alors pour tout sous-ensemble $T \subseteq V$ de taille au moins t, il existe un sommet $v \in T$ dont le degré est strictement inférieur à t'. Ainsi un tel graphe a au plus $k = (k_1 + k_2)/2$ arêtes où : $k_1 = (t-1)(n-1)$ (ce qui correspond à $t-1$ sommets de degré $n-1$) et $k_2 = (n-t+1)(t'-1)$ (ce qui correspond à $n-(t-1)$ de degré $t'-1$). On observe que si l'un des $n-(t-1)$ sommets a un degré supérieur à $t'-1$ alors le graphe possède un sous-ensemble T avec les propriétés requises. On obtient donc une contradiction. □

Le lemme que nous venons de montrer va nous permettre d'aboutir au lemme suivant :

Lemme 4.10. *Tout graphe c-dense $G = (V, E)$ possède un ensemble $T \subseteq V$ satisfaisant*

(i) $|T| \geq \left\lfloor n - \dfrac{\sqrt{9 - 4n + 4n^2 - 8cn^2} - 3}{2} \right\rfloor$,

(ii) pour tout $v \in T$, $d(v) \geq \left\lfloor n - \dfrac{\sqrt{9 - 4n + 4n^2 - 8cn^2} - 3}{2} \right\rfloor - 2$.

Démonstration. Pour démontrer le lemme, on utilise le lemme 4.9 avec $t' = t - 2$. Comme le graphe G est dense, on a $|E| \geq cn^2$. En utilisant l'inégalité $1 + ((t-1)(n-1) + (n - t + 1)(t-3))/2 \leq cn^2$ on obtient que pour un graphe dense, pour tout entier t tel que $t \leq n + \frac{3-\sqrt{9-4n+4n^2-8cn^2}}{2}$, le graphe admet un ensemble de taille au moins t dont tous les sommets ont pour degré au moins $t-2$. □

Finalement, en utilisant l'approche « beaucoup de sommets de grand degré » présentée à la section 4.2.1, on obtient un algorithme pour résoudre DOMINATION sur les graphes c-denses dont le temps d'exécution est précisé par le théorème suivant.

Théorème 4.11. *Pour tout c avec $0 < c < 1/2$, il existe un algorithme qui s'exécute en temps $\mathcal{O}(1.2303^{(1+\sqrt{1-2c})n})$ pour résoudre le problème* Domination *sur les graphes c-denses.*

Démonstration. En combinant le théorème 4.4, le corollaire 4.5 et le lemme 4.10, on obtient un algorithme pour résoudre le problème de l'ensemble dominant de taille minimum en temps

$$\begin{aligned}
1.2303^{2n-\left\lfloor n-\frac{\sqrt{9-4n+4n^2-8cn^2}-3}{2}\right\rfloor} &\leq 1.2303^{2n-(n-\frac{\sqrt{9-4n+4n^2-8cn^2}-3}{2}-1)} \\
&= 1.2303^{n+\frac{\sqrt{9-4n+4n^2-8cn^2}-1}{2}} \\
&\leq 1.2303^{n+\frac{\sqrt{9+4n^2(1-2c)}}{2}} \\
&\leq 1.2303^{n+\frac{3+2n\sqrt{1-2c}}{2}} \\
&\leq 1.2303^{n(1+\sqrt{1-2c})} \cdot 1.2303^{\frac{3}{2}} \\
= \mathcal{O}(1.2303^{n(1+\sqrt{1-2c})}).
\end{aligned}$$

□

Notons que pour tout $c \geq 0.043$ on obtient un temps d'exécution meilleur que $\mathcal{O}(1.5^n)$.

Tab. 4.3 – Temps d'exécution de l'algorithme du théorème 4.11 pour quelques valeurs de c avec $0 \leq c \leq 1/2$.

valeur de c	0.1	0.2	0.3	0.4	0.5
temps d'exécution	$\mathcal{O}(1.4809^n)$	$\mathcal{O}(1.4446^n)$	$\mathcal{O}(1.4027^n)$	$\mathcal{O}(1.3498^n)$	$\mathcal{O}(1.2303^n)$

Naturellement, plus le graphe est dense, meilleur sera le temps d'exécution au pire des cas (voir la table 4.3).

4.4 Graphes cordaux

Nous présentons dans cette section un algorithme exponentiel pour le problème Domination sur la classe des graphes cordaux. Nous utilisons l'approche basée sur la largeur arborescente avec un algorithme pour résoudre Domination sur les graphes cordaux utilisant un arbre de cliques (en anglais, *clique tree*). Nous commençons par donner quelques définitions sur cette classe de graphes.

Un graphe est dit *cordal* s'il ne possède pas de cycle sans corde de longueur plus grande que trois. Les graphes cordaux sont des graphes très étudiés et ont leur propre chapitre dans la monographie de Golumbic [Gol80]. Les arbres et les graphes complets

sont des exemples simples de graphes cordaux ; parmi les sous-classes étudiées de la classe des graphes cordaux, on peut citer les graphes splits, fortement cordaux et les graphes d'intersection de chemins dans un arbre (an anglais, *undirected path graphs*).

Dans la suite, nous allons utiliser l'arbre de cliques des graphes cordaux et nous le regarderons comme une décomposition arborescente du graphe. La définition d'un arbre de cliques est rappelée par le lemme 4.13. Nous commençons tout d'abord par donner quelques définitions et résultats.

Lemme 4.12 (voir [Bod05])**.** *Soit un graphe $G = (V, E)$ et une décomposition arborescente $(\{X_i : i \in I\}, T)$ de G. Si un ensemble $C \subseteq V$ est une clique de G alors il existe un $i \in I$ tel que $C \subseteq X_i$.*

Par conséquent, ce lemme permet d'établir immédiatement une borne inférieure sur la largeur arborescente d'un graphe : si C est une clique d'un graphe G quelconque alors $tw(G) \geq |C| - 1$ et donc $tw(G) \geq \omega(G) - 1$.

Lemme 4.13 (voir [BLS99, Gol80])**.** *Si $G = (V, E)$ est un graphe cordal alors il existe une décomposition arborescente $(\{X_i : i \in I\}, T)$ de G telle qu'il y a une bijection entre les cliques maximales de G et les nœuds de l'arbre T. Une telle décomposition arborescente est appelée* arbre de cliques.

On note que si G est un graphe cordal, alors son arbre de cliques est une décomposition arborescente de largeur $\omega(G) - 1$. Ainsi un arbre de cliques est une décomposition optimale d'un graphe cordal (voir la figure 4.2).

Pour un graphe quelconque, le nombre de cliques maximales peut être exponentiel (au plus $3^{n/3}$ comme l'ont montré Moon et Moser dans [MM65]). Pour un graphe cordal, ce nombre est borné par le nombre de sommets et il est possible de contruire un arbre de cliques en temps $\mathcal{O}(n + m)$, avec n le nombre de sommets du graphe et m son nombre d'arêtes (voir [Gol80] ou [Spi03, chapitre 15]).

Au lemme suivant nous donnons un algorithme pour DOMINATION sur ces graphes.

Lemme 4.14. *Il existe un algorithme qui calcule en temps $3^{tw(G)}n^{\mathcal{O}(1)} = 3^{\omega(G)}n^{\mathcal{O}(1)}$ un ensemble dominant de taille minimum d'un graphe cordal G.*

Démonstration. L'algorithme d'Alber *et al.* proposé dans [ABFKN02, AN02] résout le problème DOMINATION en temps $4^{\ell}N$ en utilisant une décomposition arborescente commode d'un graphe G, où N est le nombre de nœuds dans l'arbre de décomposition de largeur ℓ. Une description complète de l'algorithme d'Alber *et al.* est également donnée dans la monographie de Niedermeier [Nie06]. Pour calculer un ensemble dominant d'un graphe G, l'algorithme parcourt l'arbre de décomposition depuis ses feuilles jusqu'à la racine en utilisant la programmation dynamique. L'idée principale est d'affecter trois *couleurs* différentes à l'ensemble des sommets correspondant à un nœud de l'arbre :

- *noire* (noté 1), signifiant que le sommet appartient à l'ensemble dominant ;
- *blanche* (noté 0), signifiant que le sommet est déjà dominé ;
- *grise* (noté $\hat{0}$), signifiant que le sommet n'est pas encore dominé.

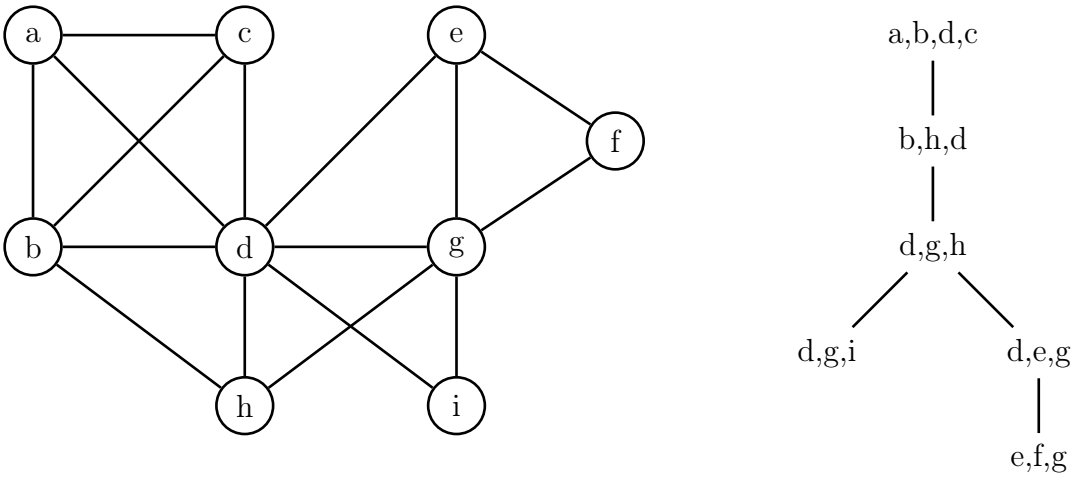


Fig. 4.2 – Un graphe cordal et son arbre de cliques.

Supposons maintenant que le graphe G soit cordal. Un arbre de cliques de G, noté T, peut être construit en temps linéaire [Bla93]. En utilisant le lemme 4.1, une décomposition arborescente commode T' de G peut être calculée à partir de la décomposition arborescente optimale T en temps $\mathcal{O}(n)$. De plus, cette décompostion arborescente commode T' est optimale et contient au plus $4n$ nœuds. Comme G est cordal, chaque ensemble de sommets correspondant à un nœud de T' induit une clique de G. Par conséquent, aucun de ces ensembles ne peut avoir à la fois un sommet de couleur noire et un sommet de couleur grise puisque tous les sommets d'un même ensemble sont voisins. De par cette restriction, il y a au plus $2^{|X|}$ colorations possibles pour un ensemble X correspondant à un nœud, contre $3^{|X|}$ pour un graphe quelconque.

En reprenant l'algorithme d'Alber *et al.*, nous montrons que le temps d'exécution de l'algorithme pour des graphes cordaux peut être réduit à $3^{tw(G)}n^{\mathcal{O}(1)}$ avec pour seules modifications essentielles, l'utilisation d'un arbre de cliques et une restriction sur les couleurs autorisées pour les sommets d'un ensemble de la décomposition interdisant simultanément les couleurs noire et grise.

Un vecteur $c = (c_1, \ldots, c_t, \ldots, c_{n_i}) \in \{1, 0, \hat{0}\}^{n_i}$ est appelé une *coloration* d'un nœud X_i de la décomposition, où $n_i = |X_i|$. Soit $X_i = (x_1, \ldots, x_t, \ldots, x_{n_i})$ un nœud, la *couleur* associée à x_t par la coloration c est donnée par la valeur de c_t. On rappelle que chaque nœud de la décomposition d'un graphe cordal est une clique, par conséquent un tel vecteur c ne peut contenir à la fois les couleurs 1 et $\hat{0}$. Pour un nœud X_i, on a donc au plus $2 \cdot 2^{n_i}$ colorations possibles : il y a 2^{n_i} colorations avec des 0 ou des $\hat{0}$ et 2^{n_i} colorations avec des 0 ou des 1. Une coloration c respectant cette condition est dite *valide*.

Pour chaque nœud X_i de la décomposition, on utilise une application $\mathcal{A}_i : \{1, 0, \hat{0}\}^{n_i} \to \mathbb{N} \cup \{+\infty\}$. Pour une coloration c, la valeur de $\mathcal{A}_i(c)$ représente le nombre de sommets nécessaires pour un ensemble dominant minimum du graphe visité jusqu'à ce stade dans la décomposition, et tel que la couleur affectée au sommet x_t soit c_t. Pour une coloration

$c \in \{1, 0, \hat{0}\}^{n_i}$ et une couleur $d \in \{1, 0, \hat{0}\}$, on note $\#_d(c)$ la quantité $|\{t$ tel que $1 \leq t \leq n_i$ et $c_t = d\}|$.

Sur l'ensemble de couleurs $\{1, 0, \hat{0}\}$, on définit l'ordre partiel $\prec$ tel que $\hat{0} \prec 0$ et $d \prec d$ pour tout $d \in \{1, 0, \hat{0}\}$. Pour deux colorations c et c', on pose $c \prec c'$ si et seulement si $c_t \prec c'_t$ pour tout t. Une application $\mathcal{A}$ est dite *monotone* de l'ensemble partiellement ordonné $(\{1, 0, \hat{0}\}^{n_i}, \prec)$ vers $(\mathbb{N} \cup \{+\infty\}, \leq)$ si pour deux colorations c et c', $c \prec c'$ implique $\mathcal{A}(c) \leq \mathcal{A}(c')$.

La preuve de validité de l'algorithme est une conséquence immédiate de la validité de l'algorithme de [ABFKN02, AN02]. En particulier, l'algorithme d'Alber *et al.* a comme invariant la monotonie des applications $\mathcal{A}_i$. Nous rappelons dans la suite le fonctionnement de l'algorithme et montrons que son temps d'exécution est borné par $3^{tw(G)} n^{\mathcal{O}(1)}$.

Initialisation. Pour chaque feuille X_i de la décomposition, on définit $\mathcal{A}_i$, pour toute coloration $c \in \{1, 0, \hat{0}\}^{n_i}$ valide, par :

$$\mathcal{A}_i(c) \leftarrow \begin{cases} +\infty & \text{si } c \text{ contient des 0 sans contenir de 1,} \\ \#_1(c) & \text{sinon.} \end{cases}$$

L'évaluation de $\mathcal{A}_i$ est réalisée en temps $\mathcal{O}^*(2^{n_i})$ puisqu'il n'existe que 2^{n_i+1} colorations valides.

nœud oubliant. Soit X_i le nœud courant et X_j son unique fils tel que $X_j = X_i \cup \{x\}$. On définit $\mathcal{A}_i$, pour toute coloration $c \in \{1, 0, \hat{0}\}^{n_i}$ valide, par :

$$\mathcal{A}_i(c) \leftarrow \min_{d \in \{1,0\}} \mathcal{A}_j(c \times \{d\})$$

Là aussi, l'évaluation de $\mathcal{A}_i$ est réalisée en temps $\mathcal{O}^*(2^{n_i})$ puisqu'il n'existe que 2^{n_i+1} colorations valides, et pour chacune d'elles deux colorations ($d \in \{1, 0\}$) de X_j sont regardées.

nœud insérant. Soit X_i le nœud courant et X_j son unique fils tel que $X_i = X_j \cup \{x\}$.

Soit $\phi : \{1, 0, \hat{0}\}^{n_i} \to \{1, 0, \hat{0}\}^{n_i}$ telle que $\phi(c) = (c'_1, \ldots, c'_t, \ldots, c'_{n_i})$ où

$$c'_t = \begin{cases} \hat{0} & \text{si } x_t \in N(x) \text{ et } c_t = 0, \\ c_t & \text{sinon.} \end{cases}$$

On définit $\mathcal{A}_i$, pour toute coloration $c \in \{1, 0, \hat{0}\}^{n_j}$ de X_j, par :

$$\begin{aligned} \mathcal{A}_i(c \times \{1\}) &\leftarrow \mathcal{A}_j(\phi(c)) + 1 \\ \mathcal{A}_i(c \times \{0\}) &\leftarrow \begin{cases} \mathcal{A}_j(c) & \text{si } x \text{ a un voisin } x_t \text{ avec } c_t = 1, \\ +\infty & \text{sinon.} \end{cases} \\ \mathcal{A}_i(c \times \{\hat{0}\}) &\leftarrow \mathcal{A}_j(c) \end{aligned}$$

On rapelle qu'on ne considère que des colorations valides pour X_i. Cela signifie en particulier que $\mathcal{A}_i(c \times \{1\})$ n'est défini que pour des colorations c de X_j pour lesquelles $\#_{\hat{0}}(c) = 0$. De même, $\mathcal{A}_i(c \times \{\hat{0}\})$ n'existe que si $\#_1(c) = 0$.

L'évaluation de $\mathcal{A}_i$ demande un temps $\mathcal{O}^*(2^{n_i})$. En effet, il y a 2^{n_i+1} colorations c valides pour X_j et pour chacune d'elle on calcule la coloration $c \times \{d\}$, avec $d \in \{1, 0, \hat{0}\}$ de sorte que la coloration obtenue reste une coloration valide.

nœud joint. Soit X_i le nœud courant et X_j, X_k ses deux fils tels que $X_i = X_j = X_k$. Soit c une coloration de X_i. Soit c' et c'' une coloration, respectivement, de X_j et de X_k. On dit que c' et c'' *divisent* c si, pour tout t :

1. si $c_t \in \{1, \hat{0}\}$ alors $c'_t = c''_t = c_t$;
2. si $c_t = 0$ alors $c'_t, c''_t \in \{0, \hat{0}\}$ et $c'_t \neq c''_t$.

On définit $\mathcal{A}_i$, pour toute coloration $c \in \{1, 0, \hat{0}\}^{n_i}$ valide, par :

$$\mathcal{A}_i(c) \leftarrow \min\{\mathcal{A}_j(c') + \mathcal{A}_k(c'') - \#_1(c) \text{ tel que } c' \text{ et } c'' \text{ divisent } c\}$$

Pour une coloration valide c de X_i donnée, avec $z = \#_0(c)$, il existe 2^z paires de coloration c' et c'' qui divisent c. De plus, il existe $2 \cdot \binom{n_i}{z}$ façons de colorier c avec $z = \#_0(c)$ puisque si c contient z « 0 » alors les $n_i - z$ couleurs restantes sont soit toutes des « 1 », soit toutes des « $\hat{0}$ » (1 et $\hat{0}$ ne peuvent apparaître dans une même coloration c). Par conséquent, le nombre de paires (c', c'') tel qu'il y a une coloration $c \in \{1, 0, \hat{0}\}^{n_i}$ valide pour laquelle c' et c'' divisent c est borné par

$$\sum_{z=0}^{n_i} 2 \cdot \binom{n_i}{z} 2^z = 2 \cdot 3^{n_i}.$$

Ainsi, le calcul de $\mathcal{A}_i$ nécessite un temps $\mathcal{O}^*(3^{n_i})$.

Obtention de la solution. Finalement, si r est la racine de l'arbre de décomposition, le nombre dominant du graphe G est donné par

$$\gamma(G) = \min\{A_r(c) \text{ tel que } c \in \{1, 0\}^{n_r}\}.$$

De plus, comme le remarque Alber *et al.*, un ensemble dominant de taille $\gamma(G)$ peut aussi être construit sans augmenter le temps d'exécution de l'algorithme. □

Nous introduisons le corollaire suivant comme conséquence du théorème 4.4.

Corollaire 4.15. *Il existe un algorithme qui prend en entrée un graphe G possédant une clique C, et qui résout le problème* Domination *en temps $\mathcal{O}(1.2303^{2n-|C|})$.*

Démonstration. On note que chaque sommet de C a pour degré au moins $|C| - 1$. □

L'idée principale de notre algorithme pour les graphes cordaux est que, soit le graphe a une grande largeur arborescente et il a donc nécessairement une grande clique et nous appliquons alors le corollaire 4.15, soit la largeur arborescente du graphe n'est pas trop grande et nous utilisons le lemme 4.14. Nous obtenons le théorème suivant et indiquons dans sa preuve quand il faut privilégier l'une ou l'autre des approches.

Théorème 4.16. *Il existe un algorithme qui résout en temps* $\mathcal{O}(1.4173^n)$ *le problème* DOMINATION *sur les graphes cordaux.*

Démonstration. Soit G un graphe cordal. On calcule en temps linéaire la largeur arborescente de G. Si sa largeur arborescente est telle que $tw(G) < 0.3174n$ alors par le lemme 4.14, on résout le problème DOMINATION en temps $\mathcal{O}(3^{0.3174n}) = \mathcal{O}(1.4173^n)$. Sinon, on a la relation $tw(G) \geq 0.3174n$ et en utilisant le corollaire 4.15 on obtient un ensemble dominant de plus petite taille en temps $\mathcal{O}(1.2303^{2n-0.3174n}) = \mathcal{O}(1.4173^n)$. □

4.5 Graphes 4-cordaux

Intéressons nous à présent aux graphes 4-cordaux. La *cordalité* d'un graphe est la longueur de son plus long cycle sans corde. Un graphe est dit *4-cordal* si sa cordalité est au plus égale à 4. Les graphes 4-cordaux sont une super-classe de la classe des graphes cordaux. Nous montrons dans cette section que, pour tout graphe 4-cordal G, sa largeur arborescente est au plus $3\Delta(G)$.

Un sous-ensemble de sommets $S \subset V$ est un *séparateur* si $G - S$ est non connexe. Étant donné deux sommets u et v, on dit que S est un *u, v-séparateur* si les sommets u et v appartiennent à des composantes connexes différentes dans $G - S$; on dit alors que S *sépare* u et v. Un u, v-séparateur S est *minimal* s'il n'existe pas de sous-ensemble de S qui sépare u et v. Un ensemble S est un *séparateur minimal* de G s'il existe deux sommets u et v pour lesquels S est un u, v-séparateur minimal. Une composante connexe C de $G - S$ telle que $N(C) = S$ est une *composante pleine associée à* S.

Lemme 4.17 ([Gol80])**.** *Un ensemble de sommets S d'un graphe G est un séparateur minimal de G si et seulement s'il existe deux composantes pleines disctinctes associées à S.*

Nous commençons par montrer que n'importe quel séparateur minimal d'un graphe 4-cordal G a au plus $2\Delta(G)$ sommets. Au préalable, nous donnons quelques propriétés des séparateurs minimaux d'un graphe quelconque.

Lemme 4.18 ([BBC00])**.** *Soit $X \subseteq V(G)$ un sous-ensemble de sommets qui induit un sous-graphe connexe, et soit C une composante connexe de $G - N[X]$. Alors $N(C) \setminus C$ est un séparateur minimal de G.*

Étant donné un séparateur minimal S d'un graphe G, le lemme suivant donne une technique pour calculer tous les nouveaux séparateurs minimaux « proches » de S.

Lemme 4.19 ([KK98])**.** *Soit S un séparateur minimal d'un graphe quelconque G. Soit un sommet $x \in S$ et soit D une composante connexe de $G \setminus (S \cup N(x))$. Alors l'ensemble $T = N(D)$ est un séparateur minimal de G.*

Pour démontrer la borne sur la largeur arborescente d'un graphe 4-cordal en fonction du degré maximum, nous montrons le lemme suivant :

Lemme 4.20. *Soit S un séparateur minimal d'un graphe 4-cordal G. Soit un sommet $x \in S$ et soit D une composante connexe de $G - (S \cup N(x))$. Si le séparateur minimal $T = N(D)$ n'est pas un sous-ensemble de S, alors T est de taille au plus $2\Delta(G)$.*

Démonstration. On sait par le lemme 4.19 que T est un séparateur minimal du graphe G. Soit C la composante connexe de $G - S$ contenant D.

Partitionnons T en $T_C = T \cap C$ et en $T_S = T \cap S$. Nous affirmons que pour tout $y \in T_S \setminus N(x)$ et pour tout $z \in T_C$, y et z sont adjacents. En effet, par contradiction, soit P_D un chemin sans corde allant de y à z dont les sommets internes sont contenus dans D, et P_F un chemin sans corde de x à y dont les sommets internes sont dans une composante pleine F associée à S, différente de C. On note que $x - P_F - y - P_D - z$ est un cycle sans corde de G. (Les sommets x et z sont bien adjacents puisque $z \in N(D) \cap C \subseteq N(x)$.) On sait aussi que P_F a au moins un sommet interne (car $y \notin N(x)$), et donc ce cycle a pour longueur au moins 5. Contradiction !

De par l'affirmation précédente, tous les sommets de $T_S \setminus N(x)$ sont adjacents à tous les sommets de $T \cap C$; et en particulier à un certain sommet $z \in T \cap C$. Ainsi, $|T| \leq |N(x)| + |N(z)| \leq 2\Delta(G)$. □

Théorème 4.21. *Pour tout séparateur minimal S d'un graphe 4-cordal G, on a $|S| \leq 2\Delta(G)$.*

Démonstration. Berry *et al.* ont montré dans [BBC00] que la procédure suivante produit tous les séparateurs minimaux d'un graphe.

Au cours de la phase d'initialisation, pour chaque sommet x et pour chaque composante connexe C de $G - N[x]$, on calcule le séparateur minimal de type $N(C)$ (voir le lemme 4.18).

On répète ensuite l'itération suivante, jusqu'à ce qu'aucun nouveau séparateur minimal ne soit ajouté : pour chaque séparateur minimal S obtenu à l'itération précédente (ou durant la phase d'initialisation, si l'on est à la première itération), pour chaque sommet $x \in S$ et chaque composante connexe D de $G - (S \cup N(x))$, si $T = N(D)$ n'est pas déjà dans l'ensemble des séparateurs minimaux calculés, on l'ajoute à cet ensemble (voir aussi le lemme 4.19).

Les séparateurs minimaux calculés lors de la phase d'initialisation sont de taille $\Delta(G)$. Supposons que tous les séparateurs minimaux calculés avant la k-ième itération sont de taille au plus $2\Delta(G)$. Les séparateurs T obtenus durant l'itération courante sont aussi de taille au plus $2\Delta(G)$, par le lemme 4.20.

Puisque d'après [BBC00], cet algorithme génère tous les séparateurs minimaux de G, on peut conclure que tous les séparateurs minimaux sont de taille au plus $2\Delta(G)$. □

L'algorithme générique suivant prend en entrée un graphe $G = (V, E)$ ainsi qu'un sous-ensemble de sommets Z et une composante connexe W de $G - Z$. Il calcule une décomposition arborescente du graphe $G[Z \cup W]$ telle que l'un des ensembles de sommets (habituellement appelé sacs) correspondant à un nœud de l'arbre de la décomposition arborescente contient tous les sommets de Z. En particulier, un appel à l'algorithme **MakeDec**$(G, \emptyset, V)$ calcule une décomposition arborescente du graphe G. Par un choix approprié

des sacs, nous allons montrer qu'un graphe 4-cordal a une décomposition arborescente de largeur au plus égale à $3\Delta(G)$.

Algorithme MakeDec(G, Z, W)
Entrée: Un graphe quelconque $G = (V, E)$, un ensemble de sommets Z, une composante connexe W de $G - Z$.
Sortie : Une décomposition arborescente dont l'un des sacs contient Z.

Choisir $B \subseteq Z \cup W$ tel que Z est strictement inclus dans B;
pour chaque *composante connexe* W_i *de* $G - B$ *t.q.* $W_i \subset W$ **faire**
  $Z_i \leftarrow N_G(W_i) \cap B$;
  $T_i \leftarrow$ **MakeDec**(G, Z_i, W_i);
pour chaque *arbre* T_i **faire**
  Ajouter à T_i un noeud contenant les sommets de B et rendre ce noeud adjacent au sac contenant Z_i;
Construire l'arbre de décomposition T obtenu par l'union de tous les arbres T_i, avec le noeud correspondant au sac B comme racine;
retourner T;

Une approche très similaire a été utilisée dans [BGHK95] pour montrer qu'une décomposition arborescente d'un graphe $G = (V, E)$ de largeur au plus $\mathcal{O}(k \log n)$, où k est la largeur arborescente de G et $n = |V|$, peut être trouvée en temps polynomial. Pour être complet, nous rappelons que l'algorithme **MakeDec**(G, Z, W) calcule une décomposition arborescente de $G[Z \cup W]$.

Lemme 4.22 (voir aussi [BGHK95])**.** *L'algorithme* ***MakeDec****(G, Z, W) retourne une décomposition arborescente de* $G[Z \cup W]$.

Démonstration. Nous montrons ce lemme par induction. Le lemme est vrai si le nouveau sac B est exactement $Z \cup W$.

Chaque sommet de $Z \cup W$ apparaît dans au moins un sac. Observons aussi que chaque arête de $G[Z \cup W]$ a ses deux extrémités dans B, ou ses deux extrémités sont dans $Z_i \cup W_i$ pour une certaine composante $W_i \subset W$ de $G - B$. Ainsi, chaque arête est totalement contenue dans au moins un sac de la décomposition arborescente.

Il reste à montrer que pour chaque sommet $v \in Z \cup W$, les sacs de T contenant v induisent un sous-arbre connexe. Si $v \notin B$, tous les sacs contenant v apparaissent dans la même décomposition arborescente T_i, et par induction le lemme est correct. Si $v \in B$, alors v apparaît exactement dans les sous-arbres T_i tels que $v \in Z_i$. Puisque le sac B est adjacent à un sac de T_i contenant Z_i, les sacs de T contenant v induisent un sous-arbre connexe. □

Dans la suite nous utilisons l'algorithme **MakeDec**(G, Z, W) tel que, à chaque appel récursif, Z est un séparateur minimal de G et W est une composante pleine associée à Z.

Théorème 4.23. *Soit G un graphe 4-cordal, alors $tw(G) \leq 3\Delta(G)$; de plus il existe un algorithme qui calcule en temps polynomial une décomposition arborescente de G de largeur au plus $3\Delta(G)$.*

Démonstration. Le théorème est correct pour les graphes sans arête. Soit G un graphe 4-cordal possédant au moins une arête. Nous construisons une décomposition arborescente en utilisant l'algorithme **MakeDec** et telle que tous les sacs soient de taille au plus $3\Delta(G)$. Soit $B_0 = N[x_0]$ le nouveau sac obtenu à l'étape initiale **MakeDec**$(G, \emptyset, V)$ pour un certain sommet x_0. La relation $|B_0| \leq 3\Delta(G)$ est alors vérifiée. Pour chaque composante connexe C de $G - B_0$, son voisinage $S = N(C)$ est un séparateur minimal d'après le lemme 4.18. L'algorithme appelle récursivement **MakeDec**(G, S, C) pour chaque composante connexe C de $G - B_0$. Nous gardons pour invariant que pour chaque appel récursif **MakeDec**(G, Z, W) l'ensemble Z est un séparateur minimal et l'ensemble W est une composante pleine qui lui est associée.

On affirme à présent que pour un tel appel **MakeDec**(G, Z, W), on peut toujours construire un nouveau sac B de taille au plus $3\Delta(G)$. Pour cela, on choisit un sommet arbitraire $x \in Z$ et on prend $B = Z \cup (N(x) \cap W)$. Par le théorème 4.21, la taille de B est au plus égale à $3\Delta(G)$. Pour chaque composante connexe $W' \subset W$ de $G - (Z \cup N(x))$, son voisinage $Z' = N_G(W')$ est un séparateur minimal par le lemme 4.19. Les appels récursifs sont du type **MakeDec**(G, Z', W'), où Z' est un séparateur minimal de G et W' est une composante pleine de $G - Z'$ associée à Z' ; ainsi la construction précédente peut être ré-itérée.

La décomposition arborescente peut être obtenue en temps $\mathcal{O}(nm)$ puisque, pour chaque sac B nouvellement créé, le calcul des composantes de $G - B$ et de leurs voisinages peut être effectué en temps linéaire. □

En combinant le théorème 4.23, le théorème 4.6 et le corollaire 4.7 nous établissons le théorème suivant :

Théorème 4.24. *Il existe un algorithme qui calcule en temps $\mathcal{O}(1.4913^n)$ un ensemble dominant de taille minimum d'un graphe 4-cordal.*

4.6 Graphes faiblement cordaux

Dans cette section, on s'intéresse à la classe des graphes faiblement cordaux. On dit qu'un graphe G est *faiblement cordal* si à la fois G et son complémentaire $\overline{G}$ sont des graphes 4-cordaux. Il est connu que les graphes cordaux sont une sous-classe des graphes faiblement cordaux et, bien sûr, les graphes faiblement cordaux sont une sous-classe des graphes 4-cordaux. D'après [BT01], la largeur arborescente des graphes faiblement cordaux peut être calculée en temps polynomial. Dans la suite nous montrons un résultat plus fort que celui donné à la section 4.5 : si G est un graphe faiblement cordal alors sa largeur arborescente est au plus $2\Delta(G)$.

On sait que, pour tout graphe faiblement cordal, chaque séparateur minimal S est contenu dans le voisinage d'un sommet ou d'une arête (c'est-à-dire dans l'union des voisinages des extrémités d'une arête). Nous allons construire une décomposition arborescente du graphe telle que chaque sac correspond au voisinage d'un sommet ou d'une arête.

Lemme 4.25 ([HM97])**.** *Soit G un graphe faiblement cordal. Pour tout séparateur minimal S de G et pour toute composante pleine C de $G-S$ associée à S, il existe un sommet $v \in C$ ou une arête e de $G[C]$ tel que S soit contenu dans le voisinage du sommet v ou de l'arête e.*

Finalement, nous montrons le théorème suivant :

Théorème 4.26. *Pour tout graphe G faiblement cordal, on a la relation $tw(G) \leq 2\Delta(G)$.*

Démonstration. Le théorème est correct pour tout graphe sans arête. Soit G un graphe faiblement cordal possédant au moins une arête. La construction de la décomposition arborescente est assez similaire à celle du théorème 4.23, basée sur l'algorithme **MakeDec**. On utilise des sacs de taille au plus $2\Delta(G)$. Lors du premier appel **MakeDec(**$G, \emptyset, V$**)**, on commence par un sac $B_0 = N[x]$, pour un certain sommet x quelconque. Pour chaque composante connexe C de $G - N[x]$, son voisinage $S = N(C)$ est un séparateur minimal. On appelle récursivement **MakeDec(**G, S, C**)** pour chaque composante connexe C de $G - B_0$. On garde pour invariant que pour chaque appel récursif **MakeDec(**G, Z, W**)**, Z est un séparateur minimal et W est une composante pleine qui lui est associée. Nous montrons que sous ces hypothèses, le nouveau sac B peut être choisi de sorte que sa taille soit au plus $2\Delta(G)$.

On considère, comme dans le lemme 4.25, un sommet $v \in W$ tel que $Z \subseteq N(v)$, ou une arête $e = \{u, v\}$ de $G[W]$ telle que $Z \subseteq N(u) \cup N(v)$. Dans le premier cas on choisit $N[v]$ comme étant le nouveau sac B et dans le second cas on prend $B = N[u] \cup N[v]$. Dans ces deux cas on a $Z \subset B$ et la taille de B est au plus $2\Delta(G)$. Pour chaque composante connexe $W' \subset W$ de $G - B$ son voisinage $Z' = N_G(W')$ est un séparateur minimal d'après le lemme 4.18. Ainsi nous avons récursivement construit en temps polynomial une décomposition arborescente de G dont chacun des sacs a pour taille au plus $2\Delta(G)$. □

Là encore, en combinant le théorème 4.26, le théorème 4.6 et le corollaire 4.7, notre approche basée sur la largeur arborescente permet d'établir le résultat suivant :

Théorème 4.27. *Un ensemble dominant de taille minimum peut être calculé en temps $\mathcal{O}(1.4842^n)$ pour n'importe quel graphe faiblement cordal.*

4.7 Graphes cercles

On s'intéresse à présent à la classe des graphes cercles pour laquelle nous allons donner un algorithme exponentiel résolvant le problème DOMINATION en utilisant l'approche basée sur la largeur arborescente.

Un *graphe cercle* est un graphe d'intersection de cordes dans un cercle. Plus formellement, G est un graphe cercle s'il existe un cercle avec une collection de cordes telles qu'on

peut associer de façon bijective une corde à chaque sommet de G et tel que deux sommets de G soient adjacents si et seulement si les cordes correspondantes s'intersectent. Le cercle et ses cordes sont appelés un *modèle cercle* du graphe. Nous reproduisons ici la figure 1.4 de la section 1.2.2.5 (page 18).

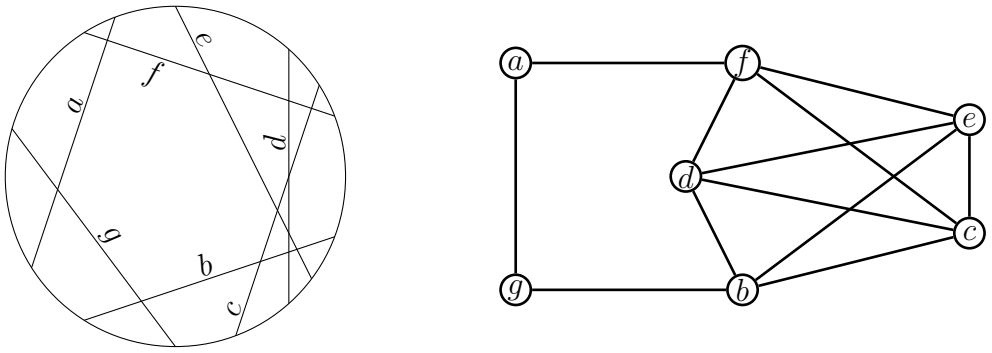


Fig. 4.3 – Exemple d'un graphe cercle et du modèle d'intersection de cordes correspondant.

Notre algorithme se sert d'une borne supérieure sur la largeur arborescente des graphes cercles en fonction de leur degré maximum. Précisément, nous montrons que pour un graphe cercle G, la relation $tw(G) \leq 4\Delta(G)$ est satisfaite.

L'approche utilisée pour montrer cette relation est basée sur l'idée fondamentale de l'algorithme de Kloks pour calculer la largeur arborescente des graphes cercles [Klo96] ; nous rappelons brièvement l'idée de cet algorithme. Considérons un modèle cercle d'un graphe cercle G. On tourne autour du cercle et on place des nouveaux points, appelés *scan-points*, entre deux extrémités de cordes. La largeur arborescente d'un graphe cercle peut être calculée en considérant toutes les triangulations possibles du polygone $\mathcal{P}$ formé par l'enveloppe convexe des scan-points. Le poids d'un triangle dans cette triangulation est le nombre de cordes du modèle cercle qui coupent ce triangle. Le poids d'une triangulation $\mathcal{T}$ est le poids maximum d'un triangle de $\mathcal{T}$. La largeur arborescente du graphe est le poids minimum moins un sur toutes les triangulations possibles $\mathcal{T}$ de $\mathcal{P}$. Nous nous référons à [Klo96] pour les preuves détaillées de la validité de l'approche développée par Kloks. Finalement, pour trouver une décomposition arborescente optimale de G, l'algorithme de Kloks [Klo96] utilise la programmation dynamique pour calculer une triangulation de poids minimum de $\mathcal{P}$.

Théorème 4.28 ([Klo96])**.** *Il existe un algorithme qui calcule en temps $\mathcal{O}(n^3)$ la largeur arborescente d'un graphe cercle et qui calcule aussi une décomposition arborescente optimale.*

Pour les graphes cercles, nous avons montré dans [GKL06] le théorème suivant :

Théorème 4.29. *Pour tout graphe cercle G, la relation $tw(G) \leq 4\Delta(G)$ est vérifiée.*

Idées de la démonstration. La preuve est technique et nous ne donnons ici que les idées principales de la démonstration proposée dans [GKL06]. Comme Kloks, nous introduisons des scan-points entre chaque extrémité de cordes sur le cercle. Ensuite nous cherchons une façon de trianguler le polygone $\mathcal{P}$ formé par l'enveloppe convexe des scan-points. On appelle *scan-lines* les segments de cette triangulation. On note que chacune des cordes du modèle intersecte au plus $\Delta(G)$ autres cordes puisque $\Delta(G)$ est le degré maximum du graphe, et qu'à chaque corde correspond un sommet du graphe. Puis, nous construisons récursivement une triangulation maximale des cordes ne s'intersectant pas dans le graphe. Finalement, il reste des polygones à trianguler et pour lesquels on dispose d'une borne sur le nombre maximum de cordes qui les intersectent. On construit alors une triangulation de ces polygones pour obtenir une triangulation de $\mathcal{P}$ dont chaque triangle est coupé par au plus $4\Delta(G)$ cordes. □

Grâce au théorème 4.29 nous pouvons maintenant appliquer aux graphes cercles l'approche basée sur la largeur arborescente et présentée à la section 4.2.2. Pour tout graphe cercle G nous avons la relation $tw(G) \leq 4\Delta(G)$. De plus la classe des graphes cercles est héréditaire et il existe un algorithme polynomial qui calcule une décomposition arborescente optimale d'un graphe cercle (voir le théorème 4.28). Par conséquent le théorème 4.6 et le corollaire 4.7 peuvent être appliqués pour obtenir le théorème suivant :

Théorème 4.30. *Il existe un algorithme qui calcule en temps $\mathcal{O}(1.4956^n)$ un ensemble dominant de taille minimum d'un graphe cercle.*

4.8 Graphes bipartis

Un graphe biparti est un graphe $G = (V, E)$ dont on peut partitionner les sommets en deux ensembles stables I_1 et I_2. Il possède donc un ensemble stable de taille au moins égale à $n/2$. Le problème DOMINATION est NP-complet, même lorsqu'on se restreint à ces graphes [Ber84]. Dans cette section nous donnons un algorithme permettant de résoudre ce problème sur cette classe de graphes en temps $\mathcal{O}(1.4143^n)$. Plus généralement, nous montrons que si un graphe admet un ensemble stable I alors un ensemble dominant de taille minimum peut être calculé en temps $\mathcal{O}^*(2^{n-|I|})$.

Dans le premier article proposant un algorithme non trivial pour DOMINATION, Fomin, Kratsch et Woeginger présentent un algorithme en temps $\mathcal{O}(1.7321^n)$ et espace exponentiel pour résoudre ce problème sur la classe des graphes bipartis [FKW04]. (L'algorithme de [FKW04] pour le cas général étant en $\mathcal{O}(1.9379^n)$.) Nous présentons cet algorithme au théorème 4.32, puis nous en donnons un nouveau, basé sur le paradigme de la programmation dynamique pour effectuer quelques pré-traitements, dont la borne asymptotique supérieure sera inférieure à celle de l'algorithme de [FKW04].

À la fois l'algorithme de Fomin *et al.* et le nôtre demandent de résoudre un problème de couverture d'ensemble dont la définition est rappelée à la section 3.4 (page 61). Fomin *et al.* donnent le lemme suivant :

Lemme 4.31 (lemme 2 de [FKW04]). *Il existe un algorithme en temps $\mathcal{O}(mk \cdot 2^m)$ qui calcule une couverture d'ensemble minimum d'une instance $(\mathcal{U}, \mathcal{S})$, avec $m = |\mathcal{U}|$ et $k = |\mathcal{S}|$.*

Le théorème suivant ainsi que sa preuve donnent le seul algorithme connu jusqu'alors pour résoudre le problème DOMINATION sur les graphes bipartis.

Théorème 4.32 (théorème 3 de [FKW04]). *Il existe un algorithme en temps $\mathcal{O}(nz \cdot 3^{n-z})$ pour calculer un ensemble dominant minimum d'un graphe ayant un ensemble stable de taille z. En particulier, il existe un algorithme en temps $\mathcal{O}(n^2 \cdot 3^{n/2}) = \mathcal{O}(1.7321^n)$ pour calculer un ensemble dominant minimum d'un graphe biparti.*

Démonstration. Soit $G = (V, E)$ un graphe avec un ensemble stable I de taille z. On rappelle qu'un ensemble stable de taille maximum peut être calculé en temps $\mathcal{O}(1.2108^n)$ en utilisant par exemple l'algorithme de Robson [Rob86]. Pour un graphe biparti, un ensemble stable maximum peut être calculé en temps polynomial via des techniques de couplage ; la taille du stable maximum dans un graphe biparti est égale au nombre de sommets moins la taille d'un couplage maximum.

Soit $R = V \setminus I$ l'ensemble des sommets à l'extérieur de l'ensemble stable. Dans une phase initiale, pour tout ensemble $X \subseteq R$ on associe un ensemble $I_X \subseteq I$ calculé de la façon suivante :

1. on pose $Y = I \setminus N[X]$;
2. on calcule un ensemble $Z \subseteq N[X] \cap I$ de cardinalité minimum tel que $R \setminus (N[X] \cup N[Y]) \subseteq N(Z)$;
3. finalement on a $I_X = Y \cup Z$.

Comme $Y \subseteq I$ et $Z \subseteq I$, il s'ensuit que $I_X \subseteq I$. Puisque $I \subseteq Y \cup N[X]$ cela implique que I est dominé par $X \cup I_X$. La relation $R \setminus (N[X] \cup N[Y]) \subseteq N(Z)$ implique que l'ensemble R est également dominé par $X \cup I_X$. Ainsi, l'ensemble $X \cup I_X$ est un ensemble dominant pour le graphe G.

On affirme que parmi tous les ensembles dominants D de G qui vérifient $D \cap R = X$, l'ensemble dominant $X \cup I_X$ en est un de plus petite taille possible. En effet, $D \cap R = X$ signifie que les sommets de $Y = I \setminus N[X]$ ne peuvent seulement être dominés s'ils appartiennent à l'ensemble D (donc $Y \subseteq D$). De plus, les sommets de $R \setminus (N[X] \cup N[Y])$ doivent tous être dominés par des sommets de $N[X] \cap I$; à la seconde étape, on détermine l'ensemble $Z \subseteq N[X] \cap I$ de cardinalité minimum avec la propriété $R \setminus (N[X] \cup N[Y]) \subseteq N(Z)$. Par conséquent, pour trouver un ensemble dominant minimum de G, il est suffisant de regarder tous les 2^{n-z} sous-ensembles $X \cup I_X$.

Concernant l'analyse du temps d'exécution, étant donné un ensemble X, la seule étape exponentielle et coûteuse pour calculer I_X est celle demandant de déterminer l'ensemble Z. Cette étape revient à résoudre un problème de couverture d'ensembles dont l'univers $\mathcal{U}$ est $R \setminus (N[X] \cup N[Y])$ et la collection $\mathcal{S}$ de sous-ensembles est donnée par $\mathcal{S} = \{N(v) \ : \ v \in N[X] \cap I\}$. On sait aussi que $|\mathcal{U}| \leq |R \setminus X| \leq n - z - |X|$ et $|\mathcal{S}| \leq z$. D'après le lemme 4.31, un tel problème de couverture d'ensembles peut être résolu en temps $\mathcal{O}(nz \cdot 2^{n-z-|X|})$.

Par conséquent, le temps d'exécution total pour résoudre tous les problèmes de couverture d'ensembles pour tous les sous-ensembles $X \subseteq R$ est majoré par

$$\sum_{k=1}^{n-z} \binom{n-z}{k} nz \cdot 2^{n-z-k} = \mathcal{O}(nz \cdot 3^{n-z}).$$

□

Même si l'algorithme rappelé au théorème 4.32 a un temps d'exécution au pire des cas borné par $\mathcal{O}(1.7321^n)$, il n'en demeure pas moins qu'à ce jour le meilleur algorithme pour résoudre DOMINATION dans le cas général est borné par $\mathcal{O}(1.5263^n)$ et espace polynomial (en temps $\mathcal{O}(1.5137^n)$ et espace exponentiel). Il est donc naturel de se demander s'il est possible de déterminer un ensemble dominant minimum d'un graphe biparti dans un temps *meilleur* que celui demandé par l'algorithme général. Nous donnons dans la suite un algorithme, basé sur le paradigme de la programmation dynamique, qui nous permettra de calculer un ensemble dominant minimum d'un graphe biparti en temps $\mathcal{O}(1.4143^n)$ et espace exponentiel.

Théorème 4.33. *Si G est un graphe qui possède un ensemble stable de taille z, alors le problème* DOMINATION *peut être résolu en temps $\mathcal{O}^*(2^{n-z})$.*

Démonstration. Soit $G = (V, E)$ un graphe et I un ensemble stable de G de taille z. Notons $\overline{I} = V \setminus I$ et notons $\{v_1, v_2, \ldots, v_z\}$ les sommets de I.

L'algorithme comporte deux phases : la première est un *pré-traitement* qui, pour tout sous-ensemble $X \subseteq \overline{I}$, calcule un ensemble dominant minimum $D_X \subseteq I$ pour $G[X]$. La seconde phase calcule, pour chaque ensemble $D_{\overline{I}} \subseteq \overline{I}$, un ensemble dominant minimum D du graphe G tel que $D \cap \overline{I} = D_{\overline{I}}$. Pour réaliser la seconde phase *rapidement*, les calculs effectués pendant le pré-traitement sont réutilisés.

La phase de pré-traitement.

Pour tout sous-ensemble $X \subseteq \overline{I}$ et pour tout entier j, $1 \leq j \leq z$, on désigne par $\mathsf{Opt}[X, j]$ un ensemble $D_{X,j}$ de plus petite cardinalité tel que
- $D_{X,j} \subseteq \{v_1, v_2, \ldots, v_j\}$, et
- $D_{X,j}$ est un ensemble dominant pour $G[X]$.

On initialise Opt de sorte que $\mathsf{Opt}[\emptyset, j] = \emptyset$ pour toute valeur de j, $1 \leq j \leq z$, et

$$\mathsf{Opt}[X, 1] = \begin{cases} \{v_1\} & \text{si } X \subseteq N(v_1), \\ \{\infty\} & \text{sinon.} \end{cases}$$

L'ensemble $\{\infty\}$ est un marqueur spécial indiquant qu'un tel sous-ensemble n'existe pas. On pose que la cardinalité de l'ensemble $\{\infty\}$ est égale à ∞.

Le calcul de $\mathsf{Opt}[X, j]$ est effectué en regardant les ensembles X par ordre de cardinalité croissante et par valeur j croissante. Ainsi, $\mathsf{Opt}[X, j]$ est obtenu de la façon suivante :

$$\mathsf{Opt}[X, j] = \begin{cases} \text{l'ensemble de cardinalité minimum entre} \\ \mathsf{Opt}[X, j-1] \text{ et } \{v_j\} \cup \mathsf{Opt}[X \setminus N(v_j), j-1]. \end{cases}$$

Cette procédure de calcul permet de pré-calculer l'ensemble dominant $D_{X,j}$ pour tout ensemble $X \subseteq \overline{I}$ et pour tout entier j, $1 \leq j \leq z$, en temps $\mathcal{O}^*(z2^{|\overline{I}|}) = \mathcal{O}^*(2^{n-z})$.

La phase de calcul.

Étant donné un ensemble $D_{\overline{I}} \subseteq \overline{I}$, nous calculons en temps polynomial un ensemble $D = D_{\overline{I}} \cup D_I$ de taille minimum, tel que D soit un ensemble dominant de G et tel que $D \cap \overline{I} = D_{\overline{I}}$. L'ensemble $D_I \subseteq I$ est donné par $D_I = D_I^1 \cup D_I^2$ avec $D_I^1 = I \setminus N(D_{\overline{I}})$ et $D_I^2 = \mathsf{Opt}[\overline{I} \setminus (N[D_{\overline{I}}] \cup N(D_I^1)), z]$.

Assertion 1. *Parmi tous les ensembles dominants D de G tels que $D \cap \overline{I} = D_{\overline{I}}$, l'ensemble $D = D_{\overline{I}} \cup D_I^1 \cup D_I^2$ est un ensemble dominant de plus petite taille.*

Démonstration. On montre d'abord que $D_{\overline{I}} \cup D_I^1 \cup D_I^2$ est un ensemble dominant. Soit $v \in \overline{I}$. Si $v \in N[D_{\overline{I}}]$ alors v est dominé par $D_{\overline{I}}$. Si $v \notin N[D_{\overline{I}}]$, on distingue deux cas. (1) Si $v \in N(D_I^1)$, alors v est dominé par un sommet de D_I^1. (2) Si $v \notin N(D_I^1)$, alors v appartient à $\overline{I} \setminus (N[D_{\overline{I}}] \cup N(D_I^1))$ et, par définition de Opt, v est dominé par D_I^2.

Si v est un sommet de I, alors soit $v \in N(D_{\overline{I}})$ et v est dominé par $D_{\overline{I}}$, soit $v \in (I \setminus N(D_{\overline{I}}))$ et v est dominé par $D_I^1 = I \setminus N(D_{\overline{I}})$.

Comme $D \cap \overline{I} = D_{\overline{I}}$ et comme I est un ensemble stable, les sommets de $I \setminus N(D_{\overline{I}})$ ne peuvent qu'être dominés par eux-mêmes. Il est donc impératif que $D_I^1 = I \setminus N(D_{\overline{I}})$ soit inclus dans tout ensemble dominant. Ensuite, comme $D \cap \overline{I} = D_{\overline{I}}$, les sommets de $\overline{I} \setminus (N[D_{\overline{I}}] \cup N(D_I^1))$ ne peuvent qu'être dominés par un sous-ensemble de I. Un plus petit sous-ensemble de I qui domine $\overline{I} \setminus (N[D_{\overline{I}}] \cup N(D_I^1))$ est donné par $D_I^2 = \mathsf{Opt}[\overline{I} \setminus (N[D_{\overline{I}}] \cup N(D_I^1)), z]$. Par conséquent, l'ensemble $D = D_{\overline{I}} \cup D_I^1 \cup D_I^2$ est un ensemble dominant de plus petite taille qui respecte $D \cap \overline{I} = D_{\overline{I}}$. □

Finalement, en énumérant les $2^{|\overline{I}|} = 2^{n-z}$ ensembles possibles pour $D_{\overline{I}} = D \cap \overline{I}$, nous déterminons un ensemble dominant D de taille minimum pour le graphe G en temps et espace $\mathcal{O}^*(2^{n-z})$. □

Comme un graphe biparti possède un ensemble stable de taille supérieure à $n/2$, on obtient le corollaire 4.34 :

Corollaire 4.34. *Étant donné un graphe biparti $G = (V, E)$, un ensemble dominant minimum de G peut être calculé en temps $\mathcal{O}(1.4143^n)$.*

Remarque. *L'algorithme établi au théorème 4.33 implique qu'on peut résoudre* DOMINATION *en temps et espace* $\mathcal{O}(1.7088^n)$ *sur un graphe arbitraire. En effet, étant donné un graphe* G*, nous calculons par une simple stratégie gloutonne un ensemble stable maximal* I *du graphe. Soit* $z = 0.2271n$*. Si* $|I| \geq z$ *alors on utilise le théorème 4.33 pour résoudre* DOMINATION *en temps* $\mathcal{O}(1.7088^n)$*. Si* $|I| < z$ *alors* $\gamma(G) < z$ *car* I *est aussi un ensemble dominant de* G*. Dans ce cas, on énumère tous les sous-ensembles de sommets de taille au plus* z *et, en utilisant l'approximation de Stirling donnée à la section 1.5.3 (page 32), on obtient*

$$\binom{n}{z} = \frac{n!}{(z)!(n-z)!} = \mathcal{O}\Big(\Big(\frac{n}{z}\Big)^z \Big(\frac{n}{n-z}\Big)^{n-z}\Big) = \mathcal{O}(1.7088^n).$$

Cette dernière remarque nous permet ainsi d'obtenir le « second meilleur algorithme » pour résoudre DOMINATION en temps et espace exponentiel. L'algorithme de Fomin *et al.* nécessitant un temps $\mathcal{O}(1.5137^n)$ et espace exponentiel.

Théorème 4.35. *Un ensemble dominant de taille minimum peut être calculé en temps* $\mathcal{O}(1.7088^n)$ *et espace exponentiel.*

4.9 Résumé des résultats obtenus

Algorithme obtenu grâce à un grand ensemble de sommets de grand degré.
L'approche basée sur la présence d'un grand nombre de sommets ayant un grand degré et présentée à la section 4.2.1, nous a permis d'obtenir l'algorithme suivant :

Théorème 4.11. *Étant donné un graphe* c*-dense* $G = (V, E)$*,* $0 < c < 1/2$*, on peut calculer un ensemble dominant minimum de* G *en temps* $\mathcal{O}(1.2303^{(1+\sqrt{1-2c})n})$*.*

Algorithmes obtenus en utilisant la largeur arborescente.
Nous avons montré à la section 4.4 qu'on peut calculer en temps $\mathcal{O}^*(3^{tw(G)})$ un ensemble dominant de taille minimum d'un graphe cordal G. En combinant ceci avec l'approche « beaucoup de sommets de grand degré », nous avons établi le théorème :

Théorème 4.16. *Étant donné un graphe cordal* $G = (V, E)$*, on peut calculer un ensemble dominant minimum de* G *en temps* $\mathcal{O}(1.4173^n)$*.*

L'approche basée sur la largeur arborescente présentée à la section 4.2.2, combinée avec les bornes établies aux sections 4.5, 4.6 et 4.7, nous a permis d'obtenir les résultats suivants :

Théorème 4.24. *Étant donné un graphe* 4*-cordal* $G = (V, E)$*, l'algorithme* **Dom-Grand-Deg-PetitTw** *calcule un ensemble dominant minimum de* G *en temps* $\mathcal{O}(1.4913^n)$*.*

Théorème 4.27. *Étant donné un graphe faiblement cordal* $G = (V, E)$*, l'algorithme* **Dom-GrandDeg-PetitTw** *calcule un ensemble dominant minimum de* G *en temps* $\mathcal{O}(1.4842^n)$*.*

Théorème 4.30. *Étant donné un graphe cercle* $G = (V, E)$*, l'algorithme* **Dom-Grand-Deg-PetitTw** *calcule un ensemble dominant minimum de* G *en temps* $\mathcal{O}(1.4956^n)$*.*

Algorithmes obtenus par pré-traitement et Programmation Dynamique.
Grâce à un pré-traitement basé sur le paradigme de la Programmation Dynamique et calculant des ensembles dominants d'une partie d'un graphe, la section 4.8 a permis d'établir les algorithmes suivants :

Théorème 4.33. *Étant donné un graphe $G = (V, E)$ possédant un ensemble stable de taille z, on peut calculer un ensemble dominant minimum de G en temps $\mathcal{O}^*(2^{n-z})$.*

Corollaire 4.34. *Étant donné un graphe biparti $G = (V, E)$, on peut calculer un ensemble dominant minimum de G en temps $\mathcal{O}(1.4143^n)$.*

4.10 Conclusion et perspectives

Nous avons présenté dans ce chapitre plusieurs algorithmes exponentiels pour résoudre le problème Domination sur des classes de graphes pour lesquelles ce problème est NP-complet. Tous les algorithmes que nous avons donnés sont plus rapides que le meilleur algorithme connu résolvant Domination sur des graphes quelconques. En outre, nous avons pointé le fait que n'importe quel algorithme permettant de résoudre le problème Couverture d'ensembles en temps $\mathcal{O}(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ avec $\alpha < 1.2303$, pourrait immédiatement être utilisé pour rendre nos algorithmes encore plus rapides ; excepté pour celui pour les graphes bipartis.

D'autres classes de graphes pourraient faire l'objet d'études pour résoudre le problème Domination. En plus des graphes peu denses (comme par exemple les graphes de degré maximum 3, voir les travaux de Fomin et Høie [FH06]), une autre classe de graphes est particulièrement intéressante : celle des graphes splits. Pour cette dernière classe, en combinant les idées de l'article [FKW04] et le résultat de [FGK05a], il est possible d'obtenir un algorithme en temps $\mathcal{O}(1.2303^n)$. Pour cela, étant donné un graphe split $G = (C, I, E)$, on considère l'instante $(\mathcal{U}, \mathcal{S})$ de Couverture d'ensembles où à chaque sommet de l'ensemble stable I correspond un élément de l'univers $\mathcal{U}$ et à chaque sommet u de la clique C correspond un ensemble $S_u = N(u) \cap I$ de $\mathcal{S}$. Une autre question intéressante, et qui semble également plus difficile, serait d'obtenir un algorithme de type Brancher & Réduire pour résoudre Domination sur les graphes bipartis ayant un temps d'exécution meilleur que celui utilisant la programmation dynamique établi à la section 4.8.

Il semble que les méthodes générales « beaucoup de sommets de grand degré » et « largeur arborescente » que nous avons développées peuvent également s'appliquer à d'autres problèmes NP-complets afin d'obtenir des algorithmes exponentiels pour des classes de graphes avec les propriétés requises. On pense en particulier aux problèmes Ensemble Stable Dominant (voir [GL07]), Ensemble Dominant Total ou encore Ensemble Stable.

Les bornes linéaires sur la largeur arborescente en fonction du degré maximum d'un graphe sont intéressantes par elles-mêmes et il est fort probable que de telles bornes soient utiles pour obtenir des algorithmes exponentiels pour d'autres classes de graphes et d'autres

problèmes, d'une façon similaire à notre approche pour DOMINATION. Il semble assez facile d'établir des bornes de ce genre pour les graphes d'intervalles (pour lesquels $tw(G) \leq \Delta(G)$) et les graphes d'intervalles circulaires (*circular arc graphs* en anglais, et pour lesquels $tw(G) \leq 2\Delta(G) + 1$). Notons que le problème DOMINATION peut être résolu en temps linéaire sur ces deux classes de graphes [HT91] et c'est pourquoi il n'y a pas d'intérêt à utiliser ces deux bornes supérieures sur la largeur arborescente pour résoudre ce problème.

Finalement, tout comme déterminer des classes de graphes pour lesquelles une telle borne linéaire supérieure n'existe pas, trouver des graphes permettant d'établir une borne inférieure sur la largeur arborescente en fonction du degré maximum serait également une question pertinente. De façon générale, ces bornes supérieures sont des résultats intéressants sur la structure des classes de graphes.

Chapitre 5

Problème de la clique dominante

5.1 Présentation du problème

De nombreuses variantes du problème classique Domination ont fait l'objet d'études. Les deux monographies de Haynes *et al.* ([HHS98a] et [HHS98b]) en décrivent une centaine avec des motivations à la fois pratique et théorique. Parmi ces variantes, quelques-unes sont bien connues et ont fait l'objet de publications [FGK06b, GL06, RS04] présentant des algorithmes exacts exponentiels :

- Ensemble Dominant Connexe : ce problème demande de trouver un ensemble dominant de plus petite taille qui soit également connexe ;
- Ensemble Stable Dominant : ce problème demande de trouver un ensemble dominant de plus petite taille qui soit également un ensemble stable du graphe ; ce problème est aussi connu sous le nom Ensemble Stable Maximal Minimum puisqu'il est équivalent à trouver le plus petit ensemble stable d'un graphe qui soit maximal par inclusion ;
- Ensemble Dominant Total : ce problème demande de trouver un ensemble dominant du graphe tel que tous les sommets (même ceux appartenant à l'ensemble) aient au moins un voisin dans l'ensemble. Autrement dit, le sous-graphe induit par cet ensemble dominant ne doit pas contenir de sommet isolé.

Dans la suite, nous nous intéressons au problème Clique Dominante défini ci-après. Notons que ce problème peut être facilement résolu en temps $\mathcal{O}^*(2^n)$ en énumérant tous les sous-ensembles de sommets et en vérifiant si l'un d'eux satisfait les propriétés requises.

Clique Dominante

entrée : Un graphe $G = (V, E)$.

question : Est-ce que G admet un ensemble dominant C tel C soit une clique ? C'est-à-dire, existe-t-il un sous-ensemble $C \subseteq V$ tel que
(i) pour tout sommet $u \in V \setminus C$, u a au moins un voisin dans C ; et
(ii) pour tout $u \in C$, pour tout $v \in C$, l'arête $\{u, v\}$ appartient à E.

L'étude des cliques dominantes a été initiée dans [CK90] par une motivation provenant des sciences sociales. Étant donné une population avec des relations entre les individus, il est demandé de trouver une clique de personnes (au sens sociologique du terme) se connaissant toutes entre elles, telle que chaque individu n'appartenant pas à la clique en connaît au moins un membre. En fait, le problème d'existence « étant donné un graphe G, décider si G admet une clique dominante », que nous noterons ExCD, est NP-complet même lorsqu'on se restreint à des classes de graphes particulières comme les graphes faiblement cordaux ou les graphes de co-comparabilité [BK87, KS93]. Cela implique immédiatement que les deux versions d'optimisation qui découlent naturellement du problème sont NP-difficiles. Le problème Clique Dominante Minimum, dénoté par MinCD, requiert de trouver une clique dominante de cardinalité minimum d'un graphe G ou d'indiquer que G n'a pas de clique dominante. Le problème Clique Dominante Maximum, dénoté par MaxCD, demande quant à lui de trouver une clique dominante de taille maximum ou de répondre qu'elle n'existe pas. La complexité de MinCD sur différentes classes de graphes a été étudiée dans les années quatre-vingt et quatre-vingt-dix (voir [Kra98] et la table 5.1).

Tab. 5.1 – Compexité des problèmes de cliques dominantes.

classe de graphe	ExCD	MinCD	MaxCD
cordaux	P[KDL94]	NP-c [BK87]	P[KDL94]
split	P	NP-c [BK87]	P
biparti	P [BK87]	P [BK87]	P
comparabilité	P [BK87]	P [BK87]	
co-comparabilité	NP-c [KS93]	NP-c [KS93]	NP-c [KS93]
intervalles	P [BK87]	P [BK87]	P
permutation	P [BK87]	P [BK87]	
cercle	P [Kei93]	P [Kei93]	
sans AT	NP-c [KS93]	NP-c [KS93]	NP-c [KS93]
largeur arborescente bornée	P [ALS91]	P [ALS91]	P

Comme indiqué au chapitre 3, plusieurs algorithmes exponentiels ont été donnés pour résoudre le problème classique Domination ([FKW04, RS04, Gra06, FGK05a]). L'algorithme le plus rapide connu à ce jour étant celui de Fomin *et al.* basé sur un algorithme résolvant le problème Couverture d'ensembles et utilisant le paradigme *Brancher & Réduire* (voir la section 2.3.1). Ils ont ainsi obtenu un temps d'exécution borné par $\mathcal{O}(1.5263^n)$ et utilisant un espace polynomial, et $\mathcal{O}(1.5137^n)$ en nécessitant un espace exponentiel. Pour établir ces temps d'exécution, leur analyse est basée sur la technique *Mesurer pour Conquérir* présentée à la section 2.3.1.2. L'algorithme nécessitant un espace exponentiel est obtenu en utilisant la technique mémorisation expliquée à la section 2.3.1.3.

Ces résultats et la puissance du paradigme Brancher & Réduire combiné à une analyse utilisant Mesurer pour Conquérir nous ont encouragés à chercher des algorithmes exponentiels pour des variantes du problème de la domination. Par exemple, Fomin *et al.*

[FGK06b] ont établi un algorithme en $\mathcal{O}(1.9407^n)$ utilisant l'approche Brancher & Réduire pour résoudre le problème Ensemble Dominant Connexe. C'est essentiellement grâce à l'analyse de type Mesurer pour Conquérir que ces chercheurs ont réussi à démontrer un temps d'exécution meilleur que $\mathcal{O}^*(2^n)$ pour leur algorithme.

Les seuls résultats connus et publiés pour des algorithmes exponentiels résolvant le problème de clique dominante ont été établis par Culberson *et al.* [CGA05]. Leur algorithme résoud le problème d'existence ExCD. L'objectif principal de leur article est l'étude de la transition de phase du problème ExCD dans les graphes randomisés du modèle classique $G_{n,p}$. C'est-à-dire, ils ont cherché à établir à la fois de façon expérimentale puis théorique à partir de quelle probabilité d'arête un graphe a une grande probabilité d'avoir une clique dominante lorsque son nombre de sommets est grand. Leur algorithme, utilisant lui aussi le paradigme Brancher & Réduire, a été utilisé pour mener ces études expérimentales et aucune analyse sur le temps d'exécution au pire des cas n'est proposée. Il est donc naturel de s'intéresser à la complexité de l'algorithme de Culberson *et al.* Nous montrons dans la prochaine section que le temps d'exécution au pire des cas de cet algorithme est $\Omega(1.4142^n)$.

Il existe également un algorithme simple qui s'exécute en temps $3^{n/3}\, n^{\mathcal{O}(1)} = \mathcal{O}(1.4423^n)$ pour résoudre les problèmes ExCD et MaxCD. Cet algorithme énumère toutes les cliques maximales et vérifie pour chacune d'elles la propriété d'être un ensemble dominant. Cet algorithme utilise un algorithme à délai polynomial pour générer tous les ensembles stables maximaux par inclusion [JYP88] et son temps d'exécution provient du résultat de Moon et Moser [MM65] indiquant que le nombre d'ensembles stables maximaux d'un graphe à n sommets est au plus $3^{n/3}$.

À la section 5.3 nous présentons un algorithme qui s'exécute en temps $\mathcal{O}(1.3387^n)$ et utilisant un espace polynomial pour calculer une clique dominante de taille minimum d'un graphe G ou pour indiquer que ce graphe ne possède pas de clique dominante. Notre algorithme résoud aussi dans le même temps le problème d'existence ExCD et est donc plus rapide que celui de Culberson *et al.* au pire des cas. L'algorithme est conçu en utilisant le paradigme Brancher & Réduire visant à employer des régles de réduction et de branchement pour résoudre récursivement le problème. Finalement, pour analyser le temps d'exécution au pire des cas de l'algorithme, nous utilisons une mesure non standard pour la taille d'une entrée d'un (sous-)problème associée à la technique Mesurer pour Conquérir. L'analyse sera décrite en détail dans les sections suivantes. La solution numérique d'un système d'environ 210 récurrences linéaires, chacune d'elles dépendant d'au plus 7 poids est très difficile à calculer sans l'aide d'un ordinateur et c'est pourquoi nous utilisons un programme basé sur une recherche locale randomisée et utilisant la méthode de Newton pour établir $\mathcal{O}(1.3387^n)$ comme borne supérieure du temps d'exécution au pire des cas de notre algorithme. Néanmoins, bien qu'une puissance de calcul importante soit nécessaire pour déterminer les valeurs optimales des poids utilisés dans la mesure μ, étant donné les récurrences et les valeurs de ces poids, il est facile de vérifier que la solution du système donnée est correcte.

En outre, puisque les méthodes dont nous disposons pour analyser les algorithmes de type Brancher & Réduire, même en utilisant la technique Mesurer pour Conquérir, semblent

surestimer le temps d'exécution des algorithmes, une borne asymptotique inférieure sur le temps d'exécution au pire des cas présente un grand intérêt. Nous montrons à la section 5.5 que le temps d'exécution au pire des cas de notre algorithme est $\Omega(1.2599^n)$.

Enfin, nous établissons à la section 5.6 un algorithme en $\mathcal{O}(1.3234^n)$ utilisant la technique Mémorisation pour résoudre MinCD en nécessitant un espace exponentiel.

5.2 L'algorithme de Culberson *et al.*

Dans [CGA05] Culberson *et al.* proposent l'algorithme Brancher & Réduire `DomClq` qui décide si un graphe G possède une clique dominante ou n'en possède pas. L'algorithme commence par brancher sur tous les voisins d'un sommet w de degré minimum de G, c'est-à-dire que, pour chaque voisin v de w, l'algorithme exécute `DomClq`$(G, \{v\}, N(v), V \setminus N[v])$ (voir l'algorithme `DomClq`). Ceci est justifié par le fait que l'on cherche un ensemble dominant D de G, donc soit le sommet w appartient à D, soit au moins un voisin de w doit être dans D. On note que si w appartient à D et $N(w) \cap D = \emptyset$ alors, puisque D induit une clique de G et que w est le sommet de degré minimum, cela signifie que G est un graphe complet et donc n'importe quel sommet de $N(w)$ pourrait remplacer w dans D. Ainsi, il est suffisant de brancher sur tous les voisins du sommet de degré minimum.

Nous commençons par redonner l'algorithme de Culberson *et al.* en utilisant des notations cohérentes avec les notations qui seront utilisées dans les sections suivantes (voir l'algorithme `DomClq`).

Comme annoncé dans la section précédente, le principal intérêt de leur travail était de déterminer la phase de transition du problème d'existence ExCD sur des graphes randomisés dans le modèle classique $G_{n,p}$. Ils ont établi un seuil égal à $\frac{3-\sqrt{5}}{2}$ et ont utilisé l'algorithme `DomClq` pour réaliser des tests expérimentaux. Culberson *et al.* indiquent dans leur article avoir réalisé des tests pour des graphes randomisés jusqu'à 1000 sommets (et la valeur de p proche du seuil $\frac{3-\sqrt{5}}{2}$). Une analyse théorique du temps d'exécution au pire des cas n'est pas proposée par les auteurs.

Il est naturel de comparer l'algorithme `DomClq`, qui n'a d'ailleurs pas été conçu pour avoir un temps d'exécution au pire des cas compétitif, avec l'algorithme que nous proposons à la section 5.3. Cependant, comme Fomin *et al.* l'ont souligné dans l'article [FGK05a], les bornes supérieures des algorithmes de type Brancher & Réduire surestiment souvent la vraie (et non connue) borne supérieure. Par conséquent, comparer les bornes supérieures au pire des cas du temps d'exécution des deux algorithmes pourrait nous amener à des conclusions incorrectes. C'est pourquoi, particulièrement dans cette situation, la recherche d'une borne inférieure sur le temps d'exécution au pire des cas est d'un grand intérêt afin de comparer l'algorithme `DomClq` au nôtre. Nous établissons le théorème suivant :

Théorème 5.1. *Le temps d'exécution au pire des cas de l'algorithme* `DomClq` *est* $\Omega(1.4142^n)$.

Démonstration. Pour démontrer le théorème, nous allons utiliser une famille de graphes pour laquelle nous étudierons le comportement de l'algorithme, c'est-à-dire que nous observerons une exécution de l'algorithme sur n'importe quel graphe de la famille. Les graphes

Algorithme DomClq(G, S, A, F)
Entrée: Un graphe $G = (V, E)$**,** $S \subseteq V$ **un ensemble de sommets sélectionnés,** $A \subseteq V$ **un ensemble de sommets disponibles,** $F \subseteq V$ **un ensemble de sommets libres.**
Sortie : Un booléen indiquant si G **a une clique dominante.**

```
si F = ∅ alors
    DOMCLQ ← S
    retourner « oui »
trouver w ∈ F tel que |N_A(w)| = min_{v∈F} |N_A(v)|
P ← N_A(w)
si P = ∅ alors
    retourner « non »
A'' ← A
tant que P ≠ ∅ faire
    choisir v ∈ P
    S ← S ∪ {v}
    A' ← N_{A''}(v)
    F' ← F \ N_F[v]
    si DomClq(G, S, A', F') alors
        retourner « oui »
    S ← S \ {v}
    A'' ← A'' \ {v}
    P ← P \ {v}
retourner « non »
```

utilisés pour montrer la borne inférieure sont notés par $G_{k,\ell} = (V_{k,\ell}, E_{k,\ell})$, $k, \ell \geq 1$. Leur ensemble de sommets est défini par

$$\begin{aligned} V_{k,\ell} &= \{w, v, a, b\} \cup \{x_i : 0 \leq i \leq \ell\} \cup \{y_i : 1 \leq i \leq \ell\} \\ &\quad \cup \{u_i : 1 \leq i \leq k\} \cup \{s_i : 1 \leq i \leq k + \ell - 1\}. \end{aligned}$$

Nous constatons que $G_{k,\ell}$ possède précisément $n = 2k + 3\ell + 4$ sommets. L'ensemble des arêtes de $G_{k,\ell}$ est donné par

$$\begin{aligned} E_{k,\ell} &= \{\{w, v\}\} \cup \{\{v, x_i\} : 0 \leq i \leq \ell\} \cup \{\{v, y_i\} : 1 \leq i \leq \ell\} \\ &\quad \cup \{\{v, s_i\} : 1 \leq i \leq k + \ell - 1\} \cup \bigcup_{i=1}^{k} \{\{u_i, s_j\} : i \leq j \leq i + \ell - 1\} \\ &\quad \cup \{\{a, x_i\} : 0 \leq i \leq \ell\} \cup \{\{b, y_i\} : 1 \leq i \leq \ell\} \cup E'_{k,\ell}. \end{aligned}$$

Finalement, tous les sommets de l'ensemble $S = \{s_i : 1 \leq i \leq k + \ell - 1\} \cup \{x_i : 0 \leq i \leq \ell\} \cup \{y_i : 1 \leq i \leq \ell\}$ sont adjacents entre eux dans le graphe $G_{k,\ell}$, sauf pour les arêtes

suivantes qui sont inexistantes $E^-_{k,\ell} = \{\{s_i, x_0\} : i \mod \ell \neq 0\} \cup \{\{x_i, y_j\} : 1 \leq i, j \leq \ell\}$. On défini donc $E'_{k,\ell}$ par $E'_{k,\ell} = \{\{u, v\} : u, v \in S, u \neq v\} \setminus E^-_{k,\ell}$. Le graphe de la figure 5.1 donne une représentation du graphe $G_{k,5}$ et permet d'aider à la compréhension de la preuve.

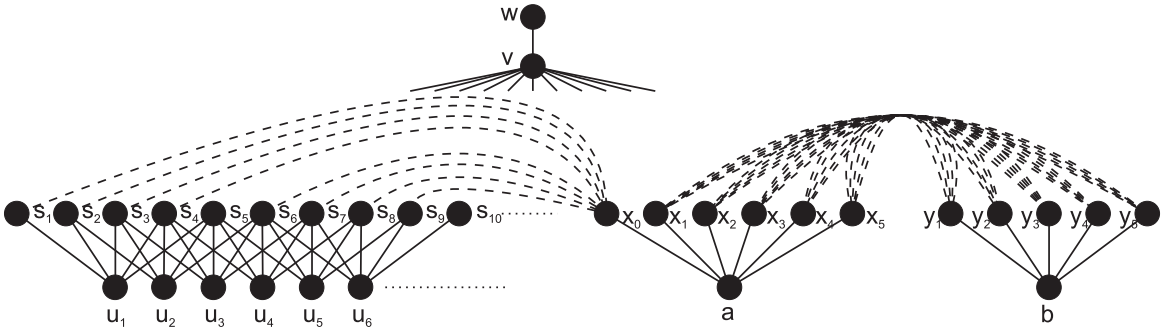


Fig. 5.1 – Le graphe $G_{k,5}$ (les lignes en pointillé représentent des non-arêtes de $G_{k,5}$).

Soit C une clique dominante de $G_{k,\ell}$. Comme C est un ensemble dominant, il contient le sommet b ou l'un de ses voisins. Cependant $b \in C$ est impossible puisque dans ce cas le sommet a ne peut avoir de voisin dans C. Par construction du graphe $G_{k,\ell}$, tous les sommets y_i ont le même voisinage et, sans perte de généralité, nous pouvons supposer que $y_1 \in C$. Cela implique que le sommet x_0 appartient à C car x_0 est le seul voisin commun aux sommets a et y_1. Par conséquent, et cela est important pour l'analyse, le sommet x_0 doit appartenir à toutes les cliques dominantes et donc aucun sommet s_i avec pour indice i satisfaisant $i \mod \ell \neq 0$ n'appartient à une clique dominante de $G_{k,\ell}$.

Considérons l'algorithme `DomClq`. Lorsqu'il s'exécute sur un graphe $G_{k,\ell}$, d'abord un sommet de degré minimum est choisi ; ce sommet est w. Ensuite l'algorithme branche sur v qui est l'unique voisin de w dans $G_{k,\ell}$. Ainsi un appel à `DomClq`$(G_{k,\ell}, \{v\}, S, \{u_i : 1 \leq i \leq k\} \cup \{a, b\})$ est réalisé. L'algorithme `DomClq` choisi un sommet dans $F = \{u_i : 1 \leq i \leq k\} \cup \{a, b\}$ avec le plus petit nombre de voisins dans S. Supposons qu'à chaque fois que l'algorithme doit choisir un tel sommet, il choisit le sommet libre u_i ayant le plus petit indice i. Ensuite, l'algorithme branche sur le voisinage de u_i. Remarquons que, comme nous l'avions noté précedemment, aucun sommet s_i avec $i \mod \ell \neq 0$ n'appartient à une clique dominante. Ainsi, `DomClq` branche vraiment sur chacun des cinq voisins de u_i, en supposant, sans perte de généralité, que l'algorithme considère les sommets $s_j \in N(u_i)$ par ordre d'indice croissant.

Considérons à présent l'arbre de recherche obtenu après le branchement sur les voisins de $u_1, u_2, \ldots, u_{t-1}$, $2 \leq t \leq k$. Lorsqu'on branche sur les voisins de u_t, l'ensemble des sommets déjà rejetés du graphe original $G_{k,\ell}$ est $\{w, v\} \cup \{u_j : 1 \leq j < t\} \cup \{s_j : 1 \leq j < t\}$. On se demande alors combien de sous-problèmes sont obtenus en branchant sur les voisins de u_t ?

- lorsqu'on branche sur s_t (le premier voisin de u_t), l'algorithme rejette 2 sommets (car $N[s_t] = \{s_t, u_t\}$), et ainsi on obtient un sous-problème de taille $q - 2$,
- lorsqu'on branche sur s_{t+1} (le second voisin de u_t), l'algorithme rejette 4 sommets : $s_t, s_{t+1}, u_t, u_{t+1}$, et on obtient un sous-problème de taille $q - 4$,

– lorsqu'on branche sur s_{t+2} (le troisième voisin de u_t), l'algorithme rejette 6 sommets : $s_t, s_{t+1}, s_{t+2}, u_t, u_{t+1}, u_{t+2}$, et on obtient un sous-problème de taille $q-6$,

– ...

– lorsqu'on branche sur $s_{t+\ell-1}$ (le dernier voisin de u_t), l'algorithme supprime 2ℓ sommets : $s_t, s_{t+1}, s_{t+2}, \ldots, s_{t+\ell-1}, u_t, u_{t+1}, u_{t+2}, \ldots, u_{t+\ell-1}$, et on obtient un sous-problème de taille $q-2\ell$,

On désigne par $L[q]$ le nombre maximum de feuilles dans l'arbre de recherche lorsque l'algorithme `DomClq` est appliqué à un sous-graphe induit de $G_{k,\ell}$ avec $q \leq n$ sommets. D'après notre analyse précédente, nous obtenons la récurrence

$$L[q] \geq L[q-2] + L[q-4] + L[q-6] + \cdots + L[q-2\ell].$$

En résolvant cette récurrence en substituant $L[q]$ par α^q, on obtient $L[q] \geq 1.4142^q$ pour tout entier $l \geq 15$.

En conclusion, 1.4142^n est une borne inférieure sur le nombre maximum de feuilles dans l'arbre de recherche correspondant à une exécution de l'algorithme `DomClq` sur un graphe ayant n sommets. □

Nous montrerons à la section 5.4 que la borne inférieure que nous venons d'établir sur le temps d'exécution au pire des cas de l'algorithme de Culberson *et al.* est asymptotiquement supérieure à la borne supérieure de l'algorithme `mdc` que nous présentons à la prochaine section.

5.3 Un algorithme Brancher & Réduire pour le problème MinCD

Dans cette section nous présentons notre nouvel algorithme pour résoudre le problème NP-difficile MinCD. Etant donné un graphe $G = (V, E)$ en entrée de l'algorithme, nous supposons que G est connexe. Dans le cas contraire, G ne peut pas avoir de clique dominante.

En guise de remarque préliminaire, on peut noter que n'importe quel graphe possédant une clique dominante a nécessairement pour diamètre au plus 3. En effet, supposons que $G = (V, E)$ est un graphe ayant pour clique dominante C et un sommet u d'excentricité supérieure à 3. On constate que tout sommet de C a pour excentricité au plus 2. Ainsi u ne peut appartenir à C. Si u n'appartient pas à C, alors il a un voisin $v \in C$, v a pour excentricité au plus 2, et donc u a pour excentrictité au plus 3 ; ce qui en contradiction avec l'hypothèse. Cette condition nécessaire sur la structure que le graphe doit avoir pour contenir une clique dominante peut être utile pour une implémentation de notre algorithme ; néanmoins, il semble difficile de l'utiliser pour améliorer l'analyse théorique au pire des cas du temps d'exécution de l'algorithme proposé.

Notre algorithme utilise quatre types de sommets et maintient une partition de V en quatre sous-ensembles disjoints :

- S est l'ensemble des sommets *sélectionnés* ou *choisis*, c'est-à-dire que ces sommets ont déjà été choisis pour une solution potentielle (et donc $G[S]$ induit un graphe complet) ;
- D est l'ensemble des sommets *rejetés*, c'est-à-dire les sommets qui ont déjà été supprimés du graphe d'entrée et qui ne peuvent appartenir à une solution ;
- $A = \bigcap_{s \in S} N(s) \setminus (S \cup D)$ est l'ensemble des sommets *disponibles*, c'est-à-dire les sommets qui pourraient être encore ajoutés à la solution potentielle S ;
- $F = V \setminus (N[S] \cup D)$ est l'ensemble des sommets *libres*, c'est-à-dire les sommets qui ont encore besoin d'être dominés (et donc, pour chaque sommet libre, au moins l'un de ses voisins disponibles doit être ajouté à S).

L'algorithme `mdc` détermine la taille d'une solution optimale SOL, c'est-à-dire d'une clique dominante de plus petite cardinalité possible telle que les propriétés suivantes sont satisfaites :

(i) $S \subseteq SOL$,

(ii) $D \cap SOL = \emptyset$, et

(iii) $F \cap SOL = \emptyset$.

On dira d'une clique dominante vérifiant ces propriétés qu'elle est une clique dominante *respectant* la partition (S, D, A, F). Si une telle clique dominante est inexistante alors `mdc` retourne « ∞ ». Pour une discussion sur les règles d'arrêt, de réduction et de branchement de notre algorithme, nous nous reportons à la prochaine section.

L'intuition est la suivante : les sommets de S appartiennent à la solution alors que les sommets de D n'y appartiennent pas. Les sommets libres F sont les sommets qui n'ont pas de voisin en S et les sommets disponibles A sont les sommets qui peuvent être choisis pour dominer des sommets libres.

Pour déterminer une clique dominante de taille minimum d'un graphe $G = (V, E)$, un appel à $\texttt{mdc}(G, \{v\}, \emptyset, N(v), V \setminus N[v])$ est effectué pour chaque sommet $v \in N[w]$, où w est un sommet de G de degré minimum ; nous empruntons cette idée à Culberson [CGA05].

La validité de cette approche, déjà montrée à la section 5.2 et que nous redonnons ici par souci d'être complet, se montre de la façon suivante. Supposons que C est une clique dominante minimum de G. Puisque C est un ensemble dominant de G, C doit contenir un sommet de $N[w]$, noté v_0. Ainsi, choisir $S = \{v_0\}$ et $D = \emptyset$ implique que seuls les voisins de v_0 sont encore disponibles et que tous les autres sommets (les sommets non adjacents à v_0) deviennent des sommets libres qui ont besoin d'être dominés (en ajoutant quelques sommets disponibles à S). Ainsi, $\texttt{mdc}(G, \{v_0\}, \emptyset, N(v_0), V \setminus N[v_0])$ retourne la taille d'une clique dominante minimum de G. Si le graphe G n'a pas de clique dominante, alors pour tout sommet $v \in N[w]$ $\texttt{mdc}(\{v\}, \emptyset, N(v), V \setminus N[v])$ retourne « ∞ ».

On note également qu'avec une légère modification, notre algorithme peut aussi retourner une clique dominante minimum SOL, s'il en existe une, au lieu de retourner uniquement sa cardinalité $|SOL|$.

Algorithme mdc(G,S,D,A,F)
Entrée: un graphe $G = (V, E)$ et une partition (S, D, A, F) de ses sommets.
Sortie : la cardinalité minimum d'une clique dominante de G respectant la partition, si elle existe.

si $\exists u \in F$ *t.q.* $d_A(u) = 0$ **alors**
  retourner ∞ **(H1)**
sinon si $F = \emptyset$ **alors**
  retourner $|S|$ **(H2)**
sinon si $\exists v \in A$ *t.q.* $d_F(v) = 0$ **alors**
  retourner mdc$(G, S, D \cup \{v\}, A \setminus \{v\}, F)$ **(R1)**
sinon si $\exists u \in F$ *t.q.* $d_A(u) = 1$ **alors**
  soit $v \in A$ **l'unique voisin de** u **dans** A
  retourner mdc$(G, S \cup \{v\}, D \cup \overline{N_A[v]} \cup N_F(v), N_A(v), F \setminus N_F(v))$ **(R2)**
sinon si $\exists v_1, v_2 \in A$ *t.q.* $N(v_1) \subseteq N(v_2)$ **alors**
  retourner mdc$(G, S, D \cup \{v_1\}, A \setminus \{v_1\}, F)$ **(R3)**
sinon si $\exists u_1, u_2 \in F$ *t.q.* $N_A(u_1) \subseteq N_A(u_2)$ **alors**
  retourner mdc$(G, S, D \cup \{u_2\}, A, F \setminus \{u_2\})$ **(R4)**
sinon si $\exists u \in F$ *t.q.* $\big(\forall v' \in N_A(u), d_F(v') = 1$ *et*
  $\exists v \in N_A(u)$ *t.q.* $A \setminus N_A(u) \subseteq N_A(v)\big)$ **alors**
  soit $v \in N_A(u)$ **un tel sommet satisfaisant la condition** **(R5)**
  retourner mdc$(G, S \cup \{v\}, D \cup (N_A(u) \setminus \{v\}) \cup \{u\}, A \setminus N_A(u), F \setminus \{u\})$
sinon
  choisir $v \in A$ **de** F**-degré maximum**
  si $d_F(v) = 1$ **alors**
    soit u **un sommet de** F
    retourner $\min_{v \in N_A(u)}\{$**mdc**$(G, S \cup \{v\}, D \cup (N_A(u) \setminus \{v\}) \cup \{u\} \cup \overline{N_A[v]}, A \setminus (\overline{N_A(v)} \cup N_A(u)), F \setminus \{u\})\}$ **(B1)**
  sinon
    retourner
    $\min\big($**mdc**$(G, S \cup \{v\}, D \cup \overline{N_A[v]} \cup N_F(v), N_A(v), F \setminus N_F(v)),$
    mdc$(G, S, D \cup \{v\}, A \setminus \{v\}, F)\big)$ **(B2)**

5.4 Analyse de l'algorithme

Dans cette section nous proposons une preuve de correction de notre algorithme et nous donnons une analyse sur son temps d'exécution au pire des cas.

5.4.1 Preuve de correction

De façon habituelle, la preuve de correction d'un algorithme de type Brancher & Réduire est facile à établir. L'algorithme s'arrête, c'est-à-dire que le sous-problème correspond à une feuille dans l'arbre de recherche, si l'une des deux conditions suivantes est respectée. Soit il existe un sommet libre sans voisin disponible (H1) et donc il n'existe pas de clique dominante pour l'instance courante ; soit il n'existe plus de sommet libre (H2) et donc S est une clique dominante de plus petite taille pour cette instance.

Autrement, l'algorithme applique éventuellement des règles de réduction sur l'instance courante, puis il branche en utilisant les règles (B1) ou (B2) sur au moins deux sous-problèmes qui sont résolus récursivement. Dans chaque sous-problème, l'algorithme choisit un sommet disponible et l'ajoute à S et/ou rejette (supprime) des sommets de G (en les ajoutant à D). Ces changements de S et de D impliquent des mises à jour sur les ensembles A et F.

Dans un premier temps, expliquons la validité des règles de réduction. Soit (S, D, A, F) la partition courante.

(R1) Si un sommet disponible v n'a pas de voisin libre alors v peut être supprimé.

(R2) Si $v \in A$ est l'unique voisin disponible d'un sommet $u \in F$ alors v doit être choisi et ajouté à S pour dominer u.

(R3) S'il existe deux sommets $v_1, v_2 \in A$ tels que $N(v_1) \subseteq N(v_2)$ alors pour toute clique dominante C contenant v_1, il existe une clique dominante $C' = (C - \{v_1\}) \cup \{v_2\}$ avec $|C'| \leq |C|$. Ainsi, on peut supprimer de façon sûre le sommet v_1.

(R4) S'il existe deux sommets $u_1, u_2 \in F$ tels que $N_A(u_1) \subseteq N_A(u_2)$ alors on peut supprimer $u_2 \in F$. Pour le vérifier, remarquons que toute clique dominante respectant la partition $(S, D \cup \{u_2\}, A, F \setminus \{u_2\})$ contient un voisin $v \in A$ de u_1 et donc aussi un voisin de u_2.

(R5) Soit $u \in F$ un sommet libre dont tous les voisins $v \in A$ ont u comme unique voisin libre. Si l'un des voisins disponibles, que nous appellerons v_0, est adjacent à tous les sommets de $A \setminus N_A(u)$, alors il existe une clique dominante minimum qui contient v_0 respectant la partition. En effet, notons qu'une clique dominante doit contenir un sommet de $N_A(u)$, et que seulement l'un de ces sommets sera choisi (tous les autres seront supprimés immédiatement par application de la règle (R1)), et donc v_0 est le meilleur choix puisqu'il n'implique pas la suppression des sommets disponibles restants (c'est-à-dire ceux de $A \setminus N_A(u)$).

On considère maintenant la validité des règles de branchement. À nouveau, on considère (S, D, A, F) une partition des sommets de G lors de l'application d'une règle de branche-

ment. On remarque que le F-degré minimum d'un sommet disponible est au moins 1 car tout sommet disponible v avec $d_F(v) = 0$ aurait été supprimé par la règle de réduction (R1).

(B1) Si tous les sommets disponibles ont exactement un voisin libre alors la règle (B1) demande de choisir un sommet libre quelconque $u \in F$. Il est alors évident qu'une clique dominante quelconque respectant (S, D, A, F) doit contenir précisément un voisin $v \in A$ de u. C'est pourquoi, pour chaque voisin $v \in A$ de u, la règle (B1) branche en un sous-problème en sélectionnant v (et en mettant à jour la partition). La cardinalité minimum de toutes les cliques dominantes obtenues est la cardinalité de la clique dominante respectant (S, D, A, F).

(B2) Pour un sommet disponible v de F-degré au moins 2, soit v est sélectionné, soit v est rejeté ; ce qui est un branchement trivial et nécessairement correct.

5.4.2 Analyse du temps d'exécution

On s'intéresse maintenant à l'analyse du temps d'exécution de l'algorithme `mdc`.

L'analyse du temps d'exécution au pire des cas d'un algorithme de type Brancher & Réduire est habituellement une tâche ardue et même les meilleures méthodes connues ne donnent que des bornes supérieures. Nous allons mener dans cette section une analyse, la plus précise possible, en utilisant l'approche Mesurer pour Conquérir combinée avec une fonction de mesure non standard.

Afin de borner le progrès réalisé par notre algorithme à chaque étape de branchement, on utilise la technique Mesurer pour Conquérir qui a été introduite dans [FGK05a] (voir aussi [FGK05b]). À chaque sommet v de G, on affecte un poids qui dépend du nombre de ses voisins libres si v est un sommet disponible, ou dépendant du nombre de ses voisins disponibles si v est un sommet libre. Pour $i \geq 0$, on définit par $a_i \in [0, 1]$ le poids d'un sommet disponible v avec $d_F(v) = i$. Pour $i \geq 0$, on définit par $f_i \in [0, 1]$ le poids d'un sommet libre v avec $d_A(v) = i$.

On utilise la mesure non standard suivante sur la taille d'une entrée d'un sous-problème :

$$\begin{aligned} \mu &= \mu(G, S, D, A, F) \\ &= \sum_{i=0}^{n} \sum_{\substack{v \in A, \\ d_F(v)=i}} a_i + \sum_{i=0}^{n} \sum_{\substack{v \in F, \\ d_A(v)=i}} f_i \end{aligned}$$

Puisque (S, D, A, F) forme une partition des sommets de $G = (V, E)$, il est facile de voir que $\mu(G, S, D, A, F) \leq |A \cup F| \leq |V| = n$.

À cause des règles (H1), (R1) et (R2) nous posons $a_0 = 0$ et $f_0 = f_1 = 0$. Afin de simplifier l'analyse, nous imposons $a_i = a_{\geq 3}$ pour tout $i \geq 3$, et $f_i = f_{\geq 6}$ pour tout $i \geq 6$. Par $a_{\geq 3} \in [0, 1]$ nous dénotons le poids d'un sommet disponible v avec $d_F(v) \geq 3$ et par

$f_{\geq 6} \in [0,1]$ nous dénotons le poids d'un sommet libre v avec $d_A(v) \geq 6$. Nous pouvons alors réécrire la mesure que nous considérerons dans la suite sous la forme suivante :

$$\begin{aligned}\mu &= \mu(G,S,D,A,F) \\ &= \sum_{\substack{v\in A,\\ d_F(v)=1}} a_1 + \sum_{\substack{v\in A,\\ d_F(v)=2}} a_2 + \sum_{\substack{v\in A,\\ d_F(v)\geq 3}} a_{\geq 3} \\ &\quad + \sum_{\substack{v\in F,\\ d_A(v)=2}} f_2 + \sum_{\substack{v\in F,\\ d_A(v)=3}} f_3 + \sum_{\substack{v\in F,\\ d_A(v)=4}} f_4 + \sum_{\substack{v\in F,\\ d_A(v)=5}} f_5 + \sum_{\substack{v\in F,\\ d_A(v)\geq 6}} f_{\geq 6}\end{aligned}$$

Par définition des poids, on a $0 \leq a_1, a_2, a_{\geq 3}, f_2, f_3, f_4, f_5, f_{\geq 6} \leq 1$. Pour simplifier davantage l'analyse nous imposons que $a_1 \leq a_2 \leq a_{\geq 3}$ et $f_2 \leq f_3 \leq f_4 \leq f_5 \leq f_{\geq 6}$. [1]

La notation suivante sera également utile dans la suite de l'analyse :

$$\Delta f_i = \begin{cases} 0 & \text{si } i \geq 7 \\ f_i - f_{i-1} & \text{si } 3 \leq i \leq 6 \\ f_2 & \text{si } i = 2 \end{cases}$$

Notons $P[\mu]$ le nombre maximum de sous-problèmes résolus récursivement par l'algorithme `mdc` afin de calculer une solution d'une instance de taille μ.

Tout d'abord, commençons par considérer les règles de réduction. Pour chacune des cinq règles de réduction, lorsqu'on les applique, soit un sommet est choisi, soit un ensemble non vide de sommets est supprimé. C'est pourquoi le nombre consécutif d'applications des règles de réduction à un sous-problème (sans branchement intermédiaire) est au plus n. Puisque chaque règle de réduction peut être facilement implémentée de sorte que son application soit réalisée en temps polynomial, le temps d'exécution de `mdc` sur un sous-problème (qui correspond à un nœud de l'arbre de recherche), excepté le temps passé dans les appels récursifs, est polynomial. De plus, nous voulons souligner le fait qu'à cause du choix de la mesure, aucune application d'une règle de réduction à un problème n'augmentera sa mesure.

On va maintenant s'attaquer à la partie la plus intéressante : l'analyse des règles de branchement (B1) et (B2). Dans une analyse dite classique du temps d'exécution des algorithmes Brancher & Réduire avec une mesure dite *standard* (dans le cas de graphes, la mesure standard d'un sous-problème est usuellement le nombre des sommets), les deux règles de branchement seraient facile à analyser et l'on n'obtiendrait que quelques récurrences. En utilisant l'approche Mesure pour Conquérir, cette analyse est plus intéressante, mais aussi un peu plus fastidieuse. D'un autre côte, Mesurer pour Conquérir permet l'obtention d'algorithmes relativement simples avec un temps d'exécution compétitif puisque

[1] L'expérience basée sur le calcul d'autres choix pour les poids indique que, sans ces restrictions, il est difficile d'obtenir des meilleurs bornes supérieures pour le temps d'exécution au pire des cas. D'autre part, le nombre de poids différents utilisés dans la mesure est crucial pour la partie calculatoire de l'analyse et doit être la plus petite possible.

l'analyse par cas qui était directement apparente dans l'algorithme de type Brancher & Réduire (voir comme exemple [TT77]) est maintenant « transférée » dans l'analyse du temps d'exécution de l'algorithme.

(B1) L'algorithme `mdc` choisit un sommet libre u et, pour chaque voisin disponible v, il s'appelle récursivement en choisissant v, c'est-à-dire en ajoutant le sommet v à l'ensemble S. On rappelle que la règle (B1) est uniquement appliquée quand tous les sommets disponibles ont pour F-degré 1. Ainsi quand `mdc` choisit un voisin disponible v de u, alors tous les autres voisins disponibles de u voient leur F-degré passer à 0 et seront donc rejetés et supprimés de A par application de la règle de réduction (R1). Finalement, on observe qu'à cause de la règle de réduction (R5) chaque voisin disponible de u doit avoir au moins un non-voisin dans $A \setminus N_A(u)$. Par conséquent, pour tout $d_A(u) \geq 2$, on obtient la récurrence :

– pour $d_A(u) = i$ avec $2 \leq i \leq 5$ on obtient

$$P[\mu] \leq 1 + i \cdot P[\mu - i \cdot a_1 - f_i - a_1] \tag{5.1}$$

– pour $d_A(u) \geq 6$ on obtient

$$P[\mu] \leq 1 + d_A(u) \cdot P[\mu - d_A(u) \cdot a_1 - f_{\geq 6} - a_1] \tag{5.2}$$

(B2) L'algorithme `mdc` choisit un sommet disponible $v \in A$ de F-degré maximum. Puisque ni la règle (R1) ni la règle (B1) n'ont été appliquées, on peut en déduire que $d_F(v) \geq 2$. En utilisant la règle (B2) on branche en deux sous-problèmes : soit en choisissant le sommet v (branche IN), soit en rejetant v (branche OUT).

Pour obtenir le sous-problème (branche IN), le sommet v est supprimé de A, tous les voisins libres de v sont supprimés de F et pour tout sommet $w \in N_A(N_F(v)) \setminus \{v\}$ son F-degré va diminuer. En plus, si un sommet $x \in N_A(N_F(v))$ a seulement un voisin libre alors ce sommet x sera supprimé lorsque la règle de réduction (R1) sera appliquée au sous-problème (branche IN). On obtient ainsi :

$$\Delta_{IN} \geq a_{d_F(v)} + \sum_{u \in N_F(v)} f_{d_A(u)} + \sum_{w \in N_A(N_F(v)) \setminus \{v\}} \left(a_{d_F(w)} - a_{d_F(w) - |N_F(v) \cap N_F(w)|}\right) \tag{5.3}$$

Pour le sous-problème (branche OUT), le sommet v est rejeté et donc supprimé de A. Ainsi le A-degré de tous les voisins libres u de v diminue de 1. Par ailleurs, lorsque la règle (R2) est appliquée au sous-problème (branche OUT), les voisins libres ayant seulement un voisin disponible seront supprimés de F. Pour dominer un tel sommet, l'algorithme a besoin de mettre leurs seuls voisins restants dans S. Pour un sommet $y \in A$, on note $N_{F2}(y) = \{u \in N_F(y) : d_A(u) = 2\}$ l'ensemble des voisins libres de v ayant précisément deux voisins disponibles et dont l'un d'eux est y. En conséquent, on obtient :

$$\Delta_{OUT} \geq a_{d_F(v)} + \sum_{u \in N_F(v) \setminus N_{F2}(v)} \Delta f_{d_A(u)} + \sum_{u \in N_{F2}(v)} f_{d_A(u)} + \sum_{w \in N_A(N_{F2}(v)) \setminus \{v\}} a_{d_F(w)} \tag{5.4}$$

Finalement, en combinant les deux équations (5.3) et (5.4) on obtient la récurrence :

$$P[\mu] \leq 1 + P[\mu - \Delta_{OUT}] + P[\mu - \Delta_{IN}]. \tag{5.5}$$

Optimisation des valeurs des poids. Afin d'établir la meilleure borne supérieure possible sur le temps d'exécution au pire des cas de l'algorithme `mdc` en utilisant les récurrences sus-citées, nous devons choisir des valeurs pour les poids a_1, a_2, $a_{\geq 3}$, f_2, f_3, f_4, f_5 et $f_{\geq 6}$ de sorte à minimiser la solution $\mathcal{O}^*(\alpha^n)$ du système de récurrences (5.1),(5.2) et (5.5). Pour optimiser les poids, il suffit de résoudre le système de récurrences obtenu en attribuant dans la récurrence (5.5) toutes les valeurs possibles pour $d_F(v) \geq 2$, pour $d_A(u) \geq 2$ pour tout $u \in N_F(v)$ (on rappelle que le nombre de voisins libres de v est précisément $d_F(v)$) et pour $d_F(w)$ tel que $1 \leq d_F(w) \leq d_F(v)$. Le nombre de ces récurrences est infini ; heureusement on peut se restreindre aux récurrences avec $2 \leq d_F(v) \leq 4$, $2 \leq d_A(u) \leq 7$ pour tout $u \in N_F(v)$, et $1 \leq d_F(w) \leq d_F(v)$. En effet, comme $\Delta f_{d_A(u)} = 0$ pour chaque sommet libre u avec $d_A(u) \geq 7$, toute solution d'une récurrence avec $d_A(u) > 7$ pour un sommet $u \in N_F(v)$ sera majorée par solution d'une récurrence avec $d_A(u) = 7$. On rappelle aussi que $a_i = a_{\geq 3}$ pour tout $i \geq 3$ et que $f_i = f_{\geq 6}$ pour tout $i \geq 6$. La section 5.4.3 explique comment simplifier davantage le système de récurrences afin de diminuer sensiblement le nombre de récurrences et de simplifier le calcul des poids. De façon similaire, si on considère la récurrence (5.2) sous l'hypothèse $a_1 \geq 0.22$ et $f_{\geq 6} = 1$, alors toutes les solutions des récurrences avec $d_A(u) \geq 7$ seront majorées par celles avec $d_A(u) = 6$. Ainsi le système de récurrences à résoudre est fini. Nous utilisons un programme pour générer toutes ces récurrences linéaires et essayons de supprimer celles qui sont superflues. Les détails concernant le choix des récurrences sont donnés à la section suivante.

Pour tout choix fixé des 8 poids et respectant les contraintes pré-citées, ce système d'environ 210 récurrences est facile à résoudre. En utilisant un programme basé sur une recherche locale, les valeurs de ces 8 poids sont cherchées de sorte à minimiser la borne supérieure sur le temps d'exécution. Nous avons obtenu numériquement les valeurs de poids suivantes : $a_1 = 0.7655$, $a_2 = 0.9595$, $a_{\geq 3} = 1$, $f_2 = 0.3813$, $f_3 = 0.7485$, $f_4 = 0.9259$, $f_5 = 0.9880$, $f_{\geq 6} = 1$. En utilisant ces valeurs, nous établissons $\mathcal{O}(1.3387^n)$ comme borne supérieure sur le temps d'exécution.

Il y a trois récurrences justes, c'est-à-dire pour lesquelles la solution correspond à la borne supérieure établie. La première d'entres elles est obtenue pour la situation suivante. Il existe un sommet disponible v de F-degré maximum avec $N_F(v) = \{u_i : \ 1 \leq i \leq 3\}$. Pour tout i, $1 \leq i \leq 3$, $|N_A(u_i)| = 6$ et pour n'importe quel i_1, i_2 tels que $1 \leq i_1, i_2 \leq 3$, $N_A(u_{i_1}) \cap N_A(u_{i_2}) = \{v\}$. De plus, pour tout voisin disponible v' d'un sommet u_i on a $d_F(v') = 3$. Si dans cette situation l'algorithme branche sur v en utilisant la règle (B2), la récurrence correspondante est alors $P[\mu] \leq 1 + P[\mu - (a_3 + 3 \cdot f_{\geq 6} + 15 \cdot (a_{\geq 3} - a_2))] + P[\mu - (a_3 + 3 \cdot (f_{\geq 6} - f_5))]$, qui est une récurrence dont la solution correspond précisément au temps d'exécution annoncé.

La seconde récurrence juste survient lorsqu'il existe un sommet disponible v de F-degré maximum égal à deux. Ses deux voisins libres, u_1 et u_2, ont chacun aucun voisin disponible en commun autre que le sommet v. Leurs degrés satisfont $d_A(u_1) = d_A(u_2) = 3$. De plus, pour tout sommet $v' \in N_A(u_1) \cup N_A(u_2)$ on a $d_F(v') = 2$. Dans ce cas, l'algorithme branche sur v en utilisant la règle (B2) et la récurrence correspondante est $P[\mu] \leq 1 + P[\mu - (a_2 + 2 \cdot f_3 + 4 \cdot (a_2 - a_1))] + P[\mu - (a_2 + 2 \cdot (f_3 - f_2))]$.

La troisième récurrence juste est obtenue lorsque l'algorithme `mdc` utilise la règle de branchement (B1) pour brancher sur les quatre voisins disponibles d'un sommet libre u, c'est-à-dire que l'on a $d_A(u) = 4$. Dans ce cas, la récurrence que l'on obtient est $P[\mu] \leq 1 + 4 \cdot P[\mu - 4 \cdot a_1 - f_4 - a_1]$.

Théorème 5.2. *L'algorithme* `mdc` *résoud le problème* MinCD *en temps* $\mathcal{O}(1.3387^n)$ *et espace polynomial.*

Remarque. *La résolution de toutes les récurrences obtenues d'une analyse standard dans laquelle aucune disctinction entre les sommets n'aurait été faite, donnerait* $\mathcal{O}(1.4656^n)$ *comme borne supérieure sur le temps d'exécution au pire des cas de* `mdc`*.*

Il n'est pas improbable que le temps d'exécution au pire des cas de l'algorithme Brancher & Réduire `mdc` est $\mathcal{O}(\alpha^n)$ pour une certaine constante $\alpha < 1.3387$. Une amélioration significative de la borne supérieure demanderait certainement un choix très différent pour la fonction de mesure ou mieux encore, des techniques d'analyses nouvelles. Néanmoins, la section 5.5 donne une borne inférieure sur le temps d'exécution au pire des cas de l'algorithme et permet donc de se forger une idée sur la meilleure borne supérieure qu'il serait possible d'atteindre.

5.4.3 Simplification du système de récurrences

Pour caculer les valeurs optimales des poids utilisés dans la définition de la mesure non standard employée à la section 5.4.2, nous avons besoin de générer un système de récurrences puis de le résoudre. Nous avons vu que le nombre de ces récurrences est infini et que, heureusement, en restreignant l'intervalle des valeurs prises par quelques degrés et en posant des restrictions sur les valeurs des poids, il est possible de réduire le nombre de récurrences à considérer de sorte que seulement un nombre fini, bien qu'éventuellement grand, subsite. Dans cette section, nous expliquons comment réduire davantage le nombre de récurrences afin de faciliter le calcul des poids optimaux.

On rappelle la récurrence correspondant à la règle de branchement (B2) :

$$P[\mu] \leq 1 + P[\mu - \Delta_{OUT}] + P[\mu - \Delta_{IN}]. \tag{5.5}$$

avec

$$\Delta_{IN} \geq a_{d_F(v)} + \sum_{u \in N_F(v)} f_{d_A(u)} + \sum_{w \in N_A(N_F(v)) \setminus \{v\}} \left(a_{d_F(w)} - a_{d_F(w) - |N_F(v) \cap N_F(w)|}\right) \tag{5.3}$$

$$\Delta_{OUT} \geq a_{d_F(v)} + \sum_{u \in N_F(v) \setminus N_{F2}(v)} \Delta f_{d_A(u)} + \sum_{u \in N_{F2}(v)} f_{d_A(u)} + \sum_{w \in N_A(N_{F2}(v)) \setminus \{v\}} a_{d_F(w)} \tag{5.4}$$

La borne inférieure que nous venons de donner sur Δ_{IN} semble demander que tous les sous-graphes possibles de G induits par $\{v\} \cup N_F(v) \cup N_A(N_F(v))$ ont besoin d'être considérés lors de la génération des récurrences. Même en prenant en compte seulement les arêtes de G avec une extrémité dans $N_F(v)$ et une dans $N_A(N_F(v))$, cela voudrait générer un grand nombre de récurrences. C'est pourquoi nous allons chercher à établir une borne inférieure sur Δ_{IN} qui nous évitera de considérer les sommets de $N_A(N_F(v))$. L'utilisation d'une telle borne nous permettra de réduire de façon draconienne le nombre de récurrences.

Soit $G = (V, E)$ un graphe, (S, D, A, F) une partition de ses sommets et $\mu(G) = \mu(G, S, D, A, F)$ la mesure de G définie à la section 5.4.2. Soit d le F-degré maximum d'un sommet disponible de G. Dans la suite, nous considérons les changements lorsque la règle de branchement (B2) est appliquée à G et nous montrons comment obtenir une borne inférieure pour Δ_{IN} et Δ_{OUT}.

5.4.3.1 Une borne inférieure pour Δ_{IN}

Lors de l'application de la règle (B2) et de la sélection d'un sommet v de F-degré maximum, le gain pour la mesure est obtenu par la suppression de v et de $N_F(v)$, et par la diminution du F-degré des sommets de $N_A(N_F(v)) \setminus \{v\}$. Pour établir une borne inférieure sur Δ_{IN}, on va borner inférieurement la somme des $a_{d_F(w)} - a_{d_F(w)-|N_F(v)\cap N_F(w)|}$ prise sur tous les sommets $w \in N_A(N_F(v)) \setminus \{v\}$.

Soit $w \in A$ un sommet dont h voisins libres sont supprimés avec $h \leq d_F(w) \leq d$. On désigne par $W_{h,d}$ une borne inférieure sur le gain dans Δ_{IN} obtenu par la diminution du F-degré de w ou par la suppression de w par une règle de réduction appliquée au sous-problème (branche IN). En utilisant les bornes inférieures $W_{h,d}$, la borne inférieure sur Δ_{IN} suivante est établie :

$$\Delta_{IN} \geq a_{d_F(v)} + \sum_{u \in N_F(v)} f_{d_A(u)} + \sum_{w \in N_A(N_F(v)) \setminus \{v\}} W_{|N_F(w)\cap N_F(v)|, d_F(v)}$$

La table suivante liste les valeurs de $W_{h,d}$ en fonction de h et de d.

(h, d)	$W_{h,d}$
$(1, 1)$	a_1
$(1, 2)$	$\min(a_1, a_2 - a_1)$
$(1, 3)$	$\min(a_1, a_2 - a_1, a_{\geq 3} - a_2)$
$(2, 2)$	a_2
$(2, 3)$	$\min(a_2, a_{\geq 3} - a_1)$
$(2, 4)$	$\min(a_2, a_{\geq 3} - a_1, a_{\geq 3} - a_2)$
$(3, 3)$	$a_{\geq 3}$
$(3, 4)$	$\min(a_{\geq 3}, a_{\geq 3} - a_1)$
$(3, 5)$	$\min(a_{\geq 3}, a_{\geq 3} - a_1, a_{\geq 3} - a_2)$
sinon	0

Nous montrons maintenant la validité des valeurs annoncées. Clairement, zéro est une borne inférieure triviale pour $W_{h,d}$ pour tout (h,d). Si $(h,d) = (1,1)$ alors w a seulement un voisin dans F et on supprime ce voisin ; par conséquent w sera supprimé de G par application de la règle de réduction (R1) et on gagne au moins a_1. Si $(h,d) = (1,2)$ alors w a un voisin libre (voir le cas $(h,d) = (1,1)$) ou w a deux voisins libres dont l'un est supprimé et le F-degré de w diminue de a_2 à a_1. Si $(h,d) = (2,3)$ alors, soit $d_F(w) = 2$, on supprime ses deux voisins et par application de (R1) w est supprimé ; soit $d_F(w) = 3$, et la suppression de deux de ses voisins aboutit à $a_{\geq 3} - a_1$. Les autres cas peuvent se montrer de façon très similaire.

Pour améliorer davantage la borne inférieure, on impose quelques restrictions sur les poids qui doivent être calculés :

$$\forall k \;\; 2 \leq k \leq d \quad \Rightarrow \quad 2W_{1,k} \leq W_{2,k}$$

$$\forall k \;\; 3 \leq k \leq d \quad \Rightarrow \quad 3W_{1,k} \leq W_{3,k}$$

Ces conditions permettent de générer un plus petit nombre de récurrences que celles obtenues d'une énumération directe de toutes les récurrences. En fait, ces restrictions garantissent que le gain $W_{h,d}$ est plus petit quand pour deux (respectivement trois) sommets disponibles distincts, exactement un voisin libre est supprimé, que lorsque exactement deux (respectivement trois) voisins libres d'un sommet disponible sont supprimés.

Pour $i \geq 0$, soit λ_i le nombre de sommets disponibles $w \in N_A(N_F(v)) \setminus \{v\}$ pour lesquels précisément i voisins libres sont supprimés du graphe quand la règle (B2) est appliquée à G et qu'un sommet $v \in A$ est sélectionné. À cause des conditions précédentes, pour tout entier positif d et pour tout $\lambda_i \geq 0$, on a :

$$\begin{aligned}
\sum_{i\geq 0} \lambda_i W_{i,d} &\geq \sum_{1\leq i\leq 3} \lambda_i W_{i,d} \\
&= \lambda_1 W_{1,d} + \lambda_2 W_{2,d} + \lambda_3 W_{3,d} \\
&= \lambda_1 W_{1,d} + 2\lambda_2/2W_{2,d} + 3\lambda_3/3W_{3,d} \\
&\geq \lambda_1 W_{1,d} + 2\lambda_2 W_{1,d} + 3\lambda_3 W_{1,d} \\
&= W_{1,d} \sum_{i=1}^{3} i\lambda_i
\end{aligned}$$

Par conséquent, chaque $w \in N_A(N_F(v)) \setminus \{v\}$ contribue pour une fois $W_{1,d}$ pour chaque sommet de $N_F(v)$ dans la somme $\sum_{i=1}^{3} i\lambda_i$, et donc on obtient la même valeur quand chaque sommet $u \in N_F(v)$ contribue pour une fois $W_{1,d}$ pour chaque w de $N_A(N_F(v)) \setminus \{v\}$. Notons que si $d_F(v) \leq 3$ alors $\sum_{i=1}^{3} i\lambda_i = \sum_{u\in N_F(v)} (d_A(u) - 1)$; et si $d_F(v) > 3$ alors $W_{1,d_F(v)} = 0$.

$$\Delta_{IN} \geq a_{d_F(v)} + \sum_{u\in N_F(v)} f_{d_A(u)} + W_{1,d_F(v)} \sum_{u\in N_F(v)} (d_A(u) - 1) \tag{5.6}$$

Finalement, on obtient une borne inférieure pour Δ_{IN} qui nécessite de considérer seulement v, ses voisins libres, et leur A-degrés. Comme les calculs le montrent, cette borne inférieure est assez fine pour établir une bonne borne supérieure sur le temps d'exécution de l'algorithme, lorsqu'on la combine avec la borne inférieure pour Δ_{OUT} que nous allons établir à la prochaine sous-section.

5.4.3.2 Une borne inférieure pour Δ_{OUT}

L'idée de base est la même que celle de la sous-section précédente. On veut établir une borne inférieure pour Δ_{OUT} qui utilise seulement le sommet v, ses voisins libres et leurs A-degrés.

Le sommet $v \in A$ est rejeté lors de l'application de la règle (B2) et l'on obtient le sous-problème (branche OUT). On rappelle que l'on avait désigné par $N_{F2}(v)$ l'ensemble $\{u \in N_F(v) : d_A(u) = 2\}$. Soit $d_{F2}(v)$ représentant la cardinalité de l'ensemble $N_{F2}(v)$.

Lors de la suppression du sommet $v \in A$, chaque sommet $u \in N_F(v)$ ayant pour A-degré égal à 2 reste avec seulement un unique voisin disponible ; et donc la règle de réduction (R2) va choisir cet unique voisin et supprimer le sommet u. Soit $Q = N_A(N_{F2}(v)) \setminus \{v\}$ l'ensemble des sommets supprimés par application de (R2) sur un sous-problème (branche OUT). Par $X_{d_{F2}(v),d}$ on dénote une borne inférieure sur le gain obtenu par la suppression de tous les sommets de Q.

Nous distinguons trois cas et nous utilisons la notation $\xi_i = |\{v \in Q \ : \ d_F(v) = i\}|$, $i \geq 0$.

Pour $d = 2$. Le gain obtenu par la suppression des sommets de Q est $\xi_1 a_1 + \xi_2 a_2$.

$$\begin{aligned} \xi_1 a_1 + \xi_2 a_2 &= \xi_1 a_1 + 2\xi_2 a_2/2 \\ &\geq \xi_1 \min(a_1, a_2/2) + 2\xi_2 \min(a_1, a_2/2) \\ &\geq d_{F2}(v) \min(a_1, a_2/2) \end{aligned}$$

Donc $X_{d_{F2}(v),2} = d_{F2}(v) \min(a_1, a_2/2)$.

Pour $d = 3$. Le gain obtenu par la suppression des sommets de Q est $\xi_1 a_1 + \xi_2 a_2 + \xi_3 a_{\geq 3}$, et on obtient

$$\begin{aligned} \xi_1 a_1 + \xi_2 a_2 + \xi_3 a_{\geq 3} &= \xi_1 a_1 + 2\xi_2 a_2/2 + 3\xi_3 a_{\geq 3}/3 \\ &\geq \xi_1 \min(a_1, a_2/2, a_{\geq 3}/3) + 2\xi_2 \min(a_1, a_2/2, \\ &\quad a_{\geq 3}/3) + 3\xi_3 \min(a_1, a_2/2, a_{\geq 3}/3) \\ &\geq d_{F2}(v) \min(a_1, a_2/2, a_{\geq 3}/3). \end{aligned}$$

Donc $X_{d_{F2}(v),3} = d_{F2}(v) \min(a_1, a_2/2, a_{\geq 3}/3)$.

Pour $d > 3$. Le gain obtenu par la suppression des sommets de Q est au moins $X_{d_{F2}(v),d} = \min(d_{F2}(v) \min(a_1, a_2/2, a3/3), a_{\geq 3})$ pour tout $d > 3$. En effet, remarquons que si $|Q| = 1$ alors tous les sommets de $N_{F2}(v)$ sont adjacents à l'unique sommet de Q ayant un grand F-degré.

Remarque. *Comme* $a_1 \leq a_2 \leq a_{\geq 3}$*, si* $d_{F2}(v) = 1$ *alors le gain est au moins* a_1*, et si* $d_{F2}(v) = 2$ *alors le gain est au moins* $\min(2a_1, a_2)$*, c'est-à-dire que pour tout d,* $X_{1,d} = a_1$ *et pour tout d,* $X_{2,d} = \min(2a_1, a_2)$*.*

Par conséquent, on obtient la borne inférieure sur Δ_{OUT} suivante :

$$\Delta_{OUT} \geq a_{d_F(v)} + \sum_{u \in N_F(v) \setminus N_{F2}(v)} \Delta f_{d_A(u)} + \sum_{u \in N_{F2}(v)} f_{d_A(u)} + X_{d_{F2}(v), d_F(v)} \tag{5.7}$$

Les bornes inférieures présentées dans cette section ont été utilisées pour générer un système de récurrences, qui à son tour a été résolu pour obtenir les poids et le temps d'exécution mentionnés à la section 5.4.2.

5.5 Une borne inférieure exponentielle sur le temps d'exécution

En mesurant la taille des sous-problèmes récursivement résolus, nous avons obtenu une borne supérieure sur le temps d'exécution au pire des cas de l'algorithme `mdc`. Néanmoins la borne démontrée pourrait n'être qu'une estimation pessimiste du vrai temps d'exécution au pire des cas. Un meilleur choix pour la fonction de mesure, tout comme l'utilisation de nouvelles techniques pour analyser les algorithmes Brancher & Réduire pourraient aboutir à des bornes supérieures plus correctes, plus proches de la réalité.

Comme notre borne supérieure sur le temps d'exécution de `mdc` peut être significativement plus grande que la borne réelle, il est naturel de chercher une borne inférieure qui donnerait une idée précise sur l'éloignement de la borne supérieure $\mathcal{O}(1.3387^n)$ par rapport à la vraie borne.

Théorème 5.3. *Le temps d'exécution au pire des cas de l'algorithme* `mdc` *est* $\Omega(1.2599^n)$*.*

Démonstration. Pour montrer cette borne, nous considérons la famille de graphes G_k défini pour tout entier $k \geq 1$ (voir la figure 5.2).

L'ensemble des sommets du graphe G_k est défini par $V_k = \{w, v\} \cup \bigcup_{\substack{1 \leq i \leq k \\ 1 \leq j \leq 5}} \{v_{i,j}\} \cup \bigcup_{1 \leq i \leq k} \{u_i\}$. L'ensemble des arêtes E_k du graphe G_k consiste en l'arête $\{w, v\}$ et en l'union de $\bigcup_{\substack{1 \leq i \leq k \\ 1 \leq j \leq 5}} \big\{\{v, v_{i,j}\}, \{u_i, v_{i,j}\}\big\}$ et $\bigcup_{\substack{1 \leq i < i' \leq k \\ 1 \leq j, j' \leq 5}} \big\{\{v_{i,j}, v_{i',j'}\} : (i', j') \neq (i+1, j)\big\}$. On note que le graphe G_k a précisément $6k + 2$ sommets.

Pour établir une borne inférieure exponentielle sur le temps d'exécution au pire des cas, nous bornons inférieurement le nombre de feuilles dans un arbre de recherche obtenu d'une exécution de l'algorithme `mdc` sur le graphe G_k (voir la figure 5.3). Lorsque plusieurs sommets seront candidats pour l'application d'une règle de branchement, sans perte de généralité, le choix fait par l'algorithme sera celui qui impliquera un nombre maximum de feuilles dans l'arbre. Un sommet de degré minimum dans G_k est le sommet w et donc

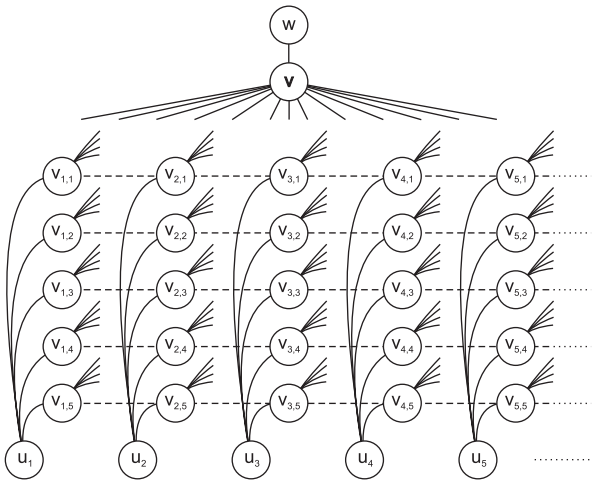


Fig. 5.2 – Le graphe G_k (les lignes en pointillé représentent des non arêtes).

pour notre analyse nous considérons l'exécution de $\texttt{mdc}(G_k, S_1, D_1, A_1, F_1)$ avec $S_1 = \{v\}$, $D_1 = \emptyset$, $A_1 = N_{G_k}(v)$ et $F_1 = V_k \setminus N_{G_k}[v] = \{u_1, u_2, \ldots, u_k\}$.

Notons qu'aucune règle de réduction ne peut être appliquée. Pour chaque i, tous les sommets $v_{i,j}$, $1 \leq j \leq 5$, ont pour unique non voisin $v_{i+1,j}$ dans $\{v_{i+1,j} : 1 \leq j \leq 5\}$. De plus, puisque tous les sommets disponibles n'ont qu'un seul voisin libre, c'est la règle de branchement (B1) qui sera appliquée. Nous supposons qu'à chaque fois que `mdc` branche dans un sous-problème en utilisant (B1) alors il branche sur le voisinage du sommet libre u_i avec le plus petit indice i possible.

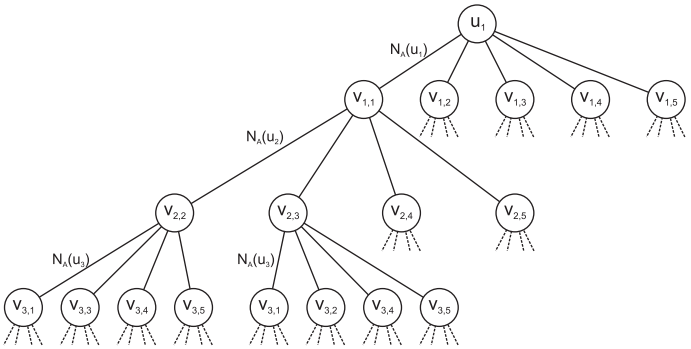


Fig. 5.3 – Une partie de l'arbre de recherche.

Considérons le premier branchement. D'après nos règles, l'algorithme branche sur le voisinage du sommet u_1, c'est-à-dire sur $v_{1,j}$, $1 \leq j \leq 5$, en 5 sous-problèmes. Considérons alors un sous-problème, disons que v_{1,j_1} a été sélectionné et donc la partition du sous-problème est (S_2, D_2, A_2, F_2) avec $S_2 = S_1 \cup \{v_{1,j_1}\}$, $D_2 = D_1 \cup \{v_{2,j_1}\} \cup \bigcup_{1 \leq j \leq 5} \{v_{1,j} : j \neq j_1\}$, $A_2 = A_1 \setminus (\{v_{2,j_1}\} \cup \bigcup_{1 \leq j \leq 5} \{v_{1,j}\})$ et $F_2 = F_1 \setminus \{u_1\}$.

Les cinq sous-problèmes ont les mêmes ensembles F_2 et quasiment les mêmes ensembles A_2. Ainsi le graphe restant après la suppression des sommets rejetés est essentiellement le

même. Par notre construction, les arguments précédents s'appliquent à n'importe quelle partition (S_2, D_2, A_2, F_2) et ainsi `mdc` branche en utilisant la règle (B1) sur chacun des *quatres* voisins disponibles de u_2. Notons que v_{2,j_1} n'est pas adjacent à v_{1,j_1} qui n'est donc pas disponible.

Supposons que pour un certain entier l, $2 \leq l \leq k-1$, la partition obtenue par des branchements successifs sur les voisins disponibles de $u_1, u_2, \ldots, u_{l-1}$ lors de l'application de la règle (B1) soit notée. De plus, on suppose que $A_l = \bigcup_{\substack{l \leq i \leq k \\ 1 \leq j \leq 5}} \{v_{i,j}\} \setminus \{v_{l,m}\}$ pour un certain entier $m \in \{1,2,3,4,5\}$ et $F_l = \bigcup_{l \leq i \leq k} \{u_i\}$. Alors l'algorithme `mdc` n'applique ni une règle d'arrêt, ni une règle de réduction. En effet, il est évident que (H1), (H2), (R1) et (R2) ne peuvent pas être appliquées puisque chaque sommet disponible a précisément un voisin libre et que chaque sommet libre a au moins quatre voisins disponibles. La règle de réduction (R3) ne peut également être appliquée car pour tous les couple de sommets disponibles u et v, il existe deux sommets u' et v' dans $G[A_l]$ tels que $u' \notin N(u)$, $u \in N(v)$ et $v' \notin N(v)$, $v \in N(u)$ (par exemple pour $v_{i,j}$ et pour $v_{i',j'}$ avec $l \leq i, i' \leq k-1$, $1 \leq j, j' \leq 5$, $v_{i,j} \neq v_{i',j'}$, on peut considérer les sommets $v_{i+1,j}$ et $v_{i'+1,j'}$). La règle de réduction (R4) n'est pas applicable car chaque sommet libre possède ses propres voisins disponibles. Concernant la règle (R5), pour chaque sommet libre u_i, $l \leq i \leq k-1$, chaque voisin disponible $v_{i,j}$, $1 \leq j \leq 5$, n'est pas adjacent à $v_{i+1,j}$ et $v_{i+1,j} \notin N_A(u_i)$. Par conséquent, (R5) ne peut s'appliquer. Ainsi, l'algorithme `mdc` applique la règle de branchement (B1) sur le voisinage du sommet u_i pour un certain i ; sans perte de généralité on suppose que l'algorithme choisit le sommet $u_i \in F_l$ avec l'indice i le plus petit possible, c'est-à-dire que l'algorithme choisit le sommet u_l.

Ensuite, lorsque l'algorithme branche sur les voisins disponibles de u_l pour obtenir un sous-problème tel que décrit précédemment, `mdc` crée une partition $(S_{l+1}, D_{l+1}, A_{l+1}, F_{l+1})$ avec $A_{l+1} = \bigcup_{\substack{l+1 \leq i \leq k \\ 1 \leq j \leq 5}} \{v_{i,j}\} \setminus \{v_{l+1,m}\}$ pour un certain entier $m \in \{1,2,3,4,5\}$ et $F_{l+1} = \bigcup_{l+1 \leq i \leq k} \{u_i\}$. Cela montre que l'algorithme `mdc` branche successivement sur le voisinage des sommets $u_1, u_2, u_3, \ldots u_{k-1}$ en utilisant la règle de branchement (B1).

Par conséquent, chaque application de la règle (B1) branche en 4 sous-problèmes (sauf pour le premier sommet u_1 où l'on obtient 5 sous-problèmes).

Ainsi le nombre de feuilles dans l'arbre de recherche est $\Omega(4^{n/6}) = \Omega(1.2599^n)$, où $n = 6k+2$ est le nombre de sommets de G_k. En effet, pour chaque sommet libre, l'algorithme branche sur ses (au moins) quatre voisins disponibles restants et il supprime de l'ensemble des sommets disponibles et libres au plus 6 sommets (excepté éventuellement quand le graphe restant a moins de 12 sommets). □

5.6 Un algorithme nécessitant un espace exponentiel

En 1986 [Rob86], Robson a montré que le temps d'exécution au pire des cas d'un algorithme Brancher & Réduire utilisant un espace polynomial pouvait être diminué. Cette amélioration nécessite alors l'utilisation d'un espace exponentiel. En particulier, cette technique appelée *mémorisation* a permis à Robson d'obtenir un algorithme qui résout le pro-

blème ENSEMBLE STABLE en temps $\mathcal{O}(1.2109^n)$ et espace exponentiel, contre $\mathcal{O}(1.2278^n)$ et espace polynomial. Une description de la technique est donnée à la section 2.3.1.3 (page 48) et nous la rappelons ci-dessous en quelques lignes.

L'idée principale est de mémoriser dans une base de données de taille exponentielle les solutions de tous les sous-problèmes résolus récursivement. Lorsqu'un nouveau sous-problème est résolu, si l'algorithme doit calculer une solution d'un sous-problème rencontré précedemment, la solution déjà calculée est obtenue de la base de données et temps polynomial (see [Rob86]). On va donc appliquer la technique Mémorisation à notre algorithme `mdc` (utilisant un espace polynomial) et appeler le nouvel algorithme nécessitant un espace exponentiel `mdc-exp`.

Théorème 5.4. *L'algorithme `mdc-exp` résoud le problème* MINCD *en temps* $\mathcal{O}(1.3234^n)$ *en utilisant un espace exponentiel.*

Démonstration. Chaque fois que l'algorithme `mdc-exp` résoud un sous-problème (G, S, D, A, F), on enregistre la solution correspondante dans une base de données. Précisément, on enregistre la taille d'une clique minimum qui est à la fois un sous-ensemble de A et qui domine tous les sommets de F. (On pourrait aussi enregistrer directement une clique minimum). On note que pour un ensemble fixé $S \cup D$, les sous-ensembles $A \subseteq V$ et $F \subseteq V$ sont déterminés de façon unique. En fait, étant donné un graphe $G = (V, E)$ et un sommet $v \in S$ (comme par exemple le premier sommet ajouté à l'ensemble S quand `mdc` est appelé), lorsque l'on considère l'ensemble $U = A \cup F$, on voit que $A = N(v) \cap U$ et que $F = U \setminus N(v)$. Par conséquent, pour tout graphe $G = (V, E)$ et pour tout sommet $v \in V$, il y a au plus 2^n sous-ensembles de sommets $U \subseteq V$ et donc la base de données ne contient pas plus que 2^n éléments.

De façon similaire à l'analyse de `mdc` à la section 5.4.2, on désigne par $P[n]$ le nombre maximum de sous-problèmes qui sont récursivement résolus par `mdc-exp` pour calculer une solution d'une entrée de taille n. Soit $P_i[n]$, $i \leq n$, le nombre maximum de sous-problèmes (G, S, D, A, F) qui sont résolus quand `mdc-exp` est appelé sur un graphe ayant n sommets et $i = |W|$ avec

$$W = \{u \in A \ : \ d_F(u) \geq 1\} \cup \{u \in F \ : \ d_A(u) \geq 2\}.$$

Notons que tous les sommets de $(A \cup F) \setminus W$ sont traités par une règle de réduction de `mdc`, et donc toute instance ayant un tel sommet peut être réduite, éventuellement par une séquence de réductions, à une instance plus petite avec $(A \cup F) \setminus W = \emptyset$.

À cause de la mesure μ utilisée dans l'analyse du temps d'exécution de `mdc`, a_1 et f_2 sont les plus petits poids dans μ, et donc la suppression de k sommets de W diminue la mesure par au moins $k \cdot \min\{a_1, f_2\}$. Ainsi, on a $P_i[n] \leq \alpha^{n-i\min\{a_1,f_2\}}$, où $\mathcal{O}^*(\alpha^n)$ est une borne supérieure sur le temps d'exécution de l'algorithme `mdc` établie en considérant la mesure μ.

De plus, il y a $\binom{n}{i}$ sous-ensembles de sommets de cardinalité i, et donc $P_i[n] \leq \binom{n}{i}$ pour tout $i \leq n$ puisque chaque sous-problème est résolu au plus une fois. Par conséquent

$P_i[n] \leq \min\{\alpha^{n-i\min\{a_1,f_2\}}, \binom{n}{i}\}$. Pour trouver la plus grande valeur de $P_i[n]$ pour un certain $i \in [0,n]$, nous devons connaître la (les) valeur(s) de β qui satisfont $\alpha^{n-\beta n\min\{a_1,f_2\}} = \binom{n}{\beta n}$ avec $i = \beta n$, $0 \leq \beta \leq 1$ (voir la figure 5.4).

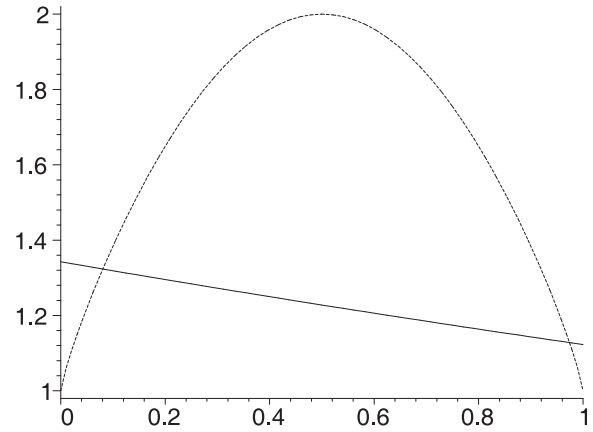


Fig. 5.4 – Les courbes pleines et en pointillé représentent respectivement les fonctions $f(\beta) = 1.342521^{1-0.6068\beta}$ et $g(\beta) = 1/(\beta^\beta(1-\beta)^{1-\beta})$ pour $0 \leq \beta \leq 1$. Comme f est une fonction strictement décroissante dans l'intervalle $[0,1]$, on s'intéresse à la plus petite valeur de $\beta \in [0,1]$ telle que $f(\beta) = g(\beta)$.

Ainsi on a $(\alpha^{1-\beta\min\{a_1,f_2\}})^n = (1/(\beta^\beta(1-\beta)^{1-\beta}))^n$ qui implique $\alpha^{1-\beta\min\{a_1,f_2\}} = 1/(\beta^\beta(1-\beta)^{1-\beta})$. Par conséquent, en faisant la somme sur toutes les valeurs possibles de i, on obtient $P[n] = \sum_{i=0}^{n} P_i[n] = \mathcal{O}(n \cdot P_{\beta n}[n])$.

En utilisant les valeurs $a_1 = 0.7457$, $a_2 = 0.9580$, $a_{\geq 3} = 1$, $f_2 = 0.6068$, $f_3 = 0.8675$, $f_4 = 0.9780$, $f_5 = 0.9973$, $f_{\geq 6} = 1$ pour les poids dans la mesure μ, nous obtenons comme borne supérieure $\mathcal{O}(\alpha^n)$ pour `mdc`, avec $\alpha = 1.342521$. On notera que les valeurs des poids ont été calculées de sorte à balancer au mieux les expressions $\alpha^{n-\beta n\min\{a_1,f_2\}}$ et $\binom{n}{\beta n}$.

Finalement, on obtient que $0.08057 > \beta > 0.08056$ où β satisfait $1.342521^{1-0.6068\beta} = 1/(\beta^\beta(1-\beta)^{1-\beta})$. Cela implique la borne annoncée pour le temps d'exécution de `mdc-exp`. □

5.7 Conclusion et perspectives

Plusieurs questions intéressantes reliées aux algorithmes exacts et exponentiels pour les problèmes de clique dominante se posent.

L'algorithme `mdc` résoud en temps $\mathcal{O}(1.3387^n)$ et en espace polynomial le problème MinCD ; l'algorithme `mdc-exp` le résoud également en demandant seulement un temps de $\mathcal{O}(1.3234^n)$ mais en nécessitant un espace exponentiel. Ces deux algorithmes permettent de même de résoudre le problème d'existence ExCD également NP-complet et pour lequel le seul algorithme exponentiel connu demandait un temps d'exécution au pire des cas

d'au moins $\Omega(1.4142^n)$. Il serait particulièrement intéressant de développer des algorithmes asymptotiquement plus rapides que nos algorithmes pour résoudre ExCD. Ce dernier problème étant plus facile que celui de minimisation.

Pour le moment, comme nous l'avons précisé en début de chapitre, le meilleur algorithme qui résoud la version de maximisation MaxCD prend un temps $\mathcal{O}^*(3^{n/3})$ et demande d'énumérer les cliques maximales du graphe donné en entrée. Il devrait être possible d'obtenir des algorithmes plus rapides. On note que l'algorithme `mdc` ne peut pas être modifié facilement pour résoudre MaxCD ; en particulier certaines règles de réduction ne peuvent plus être utilisées, comme les règles (R1), (R3) et (R5).

Les problèmes MaxCD et ExCD peuvent être résolus en temps polynomial sur la classe des graphes cordaux en énumérant toutes les cliques maximales (il y en a au plus n [FG65] et peuvent être trouvées en temps linéaire – un algorithme en temps $\mathcal{O}(n+m)$ est donné dans [Gav72] –) et en vérifiant pour chacune d'elles la propriété d'être un ensemble dominant. D'un autre côté, le problème MinCD est NP-difficile sur la classe des graphes splits. En utilisant l'algorithme pour Couverture d'ensembles donné dans [FGK05a] (voir aussi à la section 3.4), nous pouvons obtenir un algorithme en $\mathcal{O}(1.2303^n)$ pour résoudre la version de minimisation sur cette classe de graphes. En effet, si $G = (C, I, E)$ est un graphe split avec C une clique de G et I un ensemble stable de G, alors on détermine une clique dominante minimum de G en résolvant l'instance de Couverture d'ensembles définie par $\mathcal{U} = I$ et $\mathcal{S} = \{N(v) \ : \ v \in C\}$.

Cette approche peut être étendue aux graphes cordaux. Comme un graphe cordal possède au plus n cliques maximales qui peuvent être déterminées en temps linaire [FG65, Gav72], il suffit alors d'appliquer l'approche précédemment décrite pour les graphes splits : pour chacune des (au plus) n cliques maximales C_i on crée l'instance $\mathcal{S}_i = \{N(v) \ : \ v \in C_i\}$, $\mathcal{U}_i = V \setminus C_i$ sur laquelle on résout le problème Couverture d'ensembles. On obtient ainsi un algorithme en $\mathcal{O}(1.2303^n)$.

Une question intéressante serait d'étudier les problèmes MinCD et ExCD sur les graphes faiblement cordaux puisque le problème d'existence est NP-complet pour cette classe de graphes [BK87].

Chapitre 6

Une généralisation de la domination

Les généralisations de problèmes particuliers permettent de considérer des problèmes plus larges. Il nous a semblé très naturel de nous intéresser à de telles généralisations pour le problème classique de la domination sous le point de vue des algorithmes exponentiels. Résoudre de tels problèmes est une tâche particulièrement attrayante et non triviale. Une généralisation est appelée *Domination partielle* et nous en proposons une étude à la section 7.1 (page 144). À la section 7.2 (page 162), nous considérons une autre généralisation appelée *Domination avec des puissances variables*. Dans ce chapitre, nous nous intéressons à une généralisation encore plus forte, la (σ, ϱ)-domination, qui permet de modéliser de nombreux problèmes de graphes comme l'atteste la table 6.1.

6.1 Introduction

Le problème (σ, ϱ)-domination considère deux ensembles $\sigma \subseteq \mathbb{N}$ et $\varrho \subseteq \mathbb{N}$ d'entiers positifs ou nuls. Étant donné un graphe $G = (V, E)$ et un sous-ensemble de sommets $S \subseteq V$, l'ensemble S est appelé (σ, ϱ)-*dominant* de G si :

(i) $|N(v) \cap S| \in \sigma$ pour tout sommet $v \in S$, et
(ii) $|N(v) \cap S| \in \varrho$ pour tout sommet $v \in V \setminus S$.

Un exemple d'ensemble (σ, ϱ)-dominant d'un graphe est proposé à la figure 6.1.

Ce problème est connu depuis les années 90 et a été introduit par J.A. Telle [Tel94] afin de généraliser des problèmes de type domination dans les graphes. D'ailleurs beaucoup de choix pour les ensembles σ et ϱ donnent des problèmes NP-complets bien connus demandant de décider si un graphe possède un ensemble (σ, ϱ)-dominant ou de chercher un ensemble (σ, ϱ)-dominant de taille minimum ou maximum. La table 6.1 liste quelques uns de ces problèmes et la figure 6.2 (page 123) en donne une définition.

Dans ce chapitre on s'intéresse aux problèmes de la (σ, ϱ)-domination pour décider l'existence, rechercher et compter ces ensembles. Précisément, nous étudions les problèmes suivants :

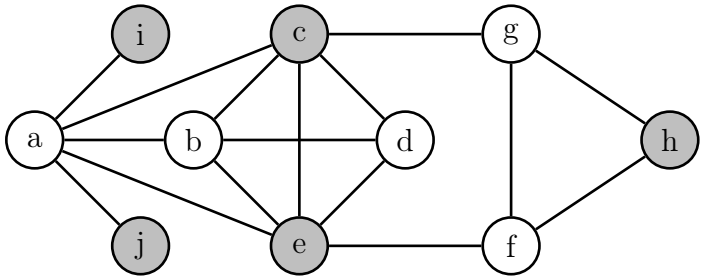


Fig. 6.1 – Exemple pour lequel les sommets grisés du graphe sont un ensemble (σ, ϱ)-dominant avec $\sigma = \{0, 1\}$ et $\varrho = \{2, 4, 8\}$.

σ	ϱ	ensembles (σ, ϱ)-dominant
$\mathbb{N}$	$\mathbb{N}^*$	ensemble dominant
$\mathbb{N}$	$\{r, r+1, \ldots\}$	r-ensemble dominant
$\{0\}$	$\mathbb{N}$	ensemble stable
$\mathbb{N}$	$\{1\}$	ensemble dominant parfait
$\{0\}$	$\{1\}$	code parfait
$\{0, 1\}$	$\{1\}$	ensemble dominant parfait faible
$\{0\}$	$\{0, 1\}$	ensemble stable fort
$\{0\}$	$\mathbb{N}^*$	ensemble stable dominant
$\mathbb{N}^*$	$\mathbb{N}^*$	ensemble dominant total
$\{1\}$	$\{1\}$	ensemble dominant total parfait
$\{1\}$	$\mathbb{N}$	couplage induit
$\{0, \ldots, r\}$	$\mathbb{N}$	sous-graphe induit de degré borné
$\{r\}$	$\mathbb{N}$	sous-graphe r-régulier induit

Tab. 6.1 – Quelques exemples d'ensembles (σ, ϱ)-dominants, $\mathbb{N}^*$ est l'ensemble des entiers strictement positifs, $\mathbb{N}$ est l'ensemble des entiers positifs ou nuls – voir aussi la figure 6.2 (page de droite) pour une définition de quelques problèmes.

r-Ensemble Dominant (Ensemble Dominant Parfait)

entrée : Un graphe $G = (V, E)$.

question : Le graphe G admet-il un ensemble $D \subseteq V$ tel que pour tout $v \in V \setminus D$, v a au moins r voisins (exactement un voisin) dans D ?

Code Parfait

entrée : Un graphe $G = (V, E)$.

question : Le graphe G admet-il un ensemble stable $D \subseteq V$ tel que pour tout $v \in V \setminus D$, v a précisément un seul voisin dans D ?

Ensemble Dominant Parfait Faible

entrée : Un graphe $G = (V, E)$.

question : Le graphe G admet-il un ensemble $D \subseteq V$ tel que pour tout $v \in V \setminus D$, v a précisément un seul voisin dans D et tel que $\Delta(G[D]) \leq 1$?

Ensemble Stable Fort (Ensemble Stable Dominant)

entrée : Un graphe $G = (V, E)$.

question : Le graphe G admet-il un ensemble stable $D \subseteq V$ tel que pour tout $v \in V \setminus D$, v a au plus (au moins) un voisin dans D ?

Ensemble Dominant Total (Ensemble Dominant Total Parfait)

entrée : Un graphe $G = (V, E)$.

question : Le graphe G admet-il un ensemble $D \subseteq V$ tel que pour tout $v \in V$, v a au moins (exactement) un voisin dans D ?

Couplage induit

entrée : Un graphe $G = (V, E)$.

question : Le graphe G admet-il un ensemble $D \subseteq V$ tel que $G[D]$ soit une collection disjointe de K_2 ?

Sous-graphe induit de Degré Borné (Sous-graphe r-Régulier induit)

entrée : Un graphe $G = (V, E)$.

question : Le graphe G admet-il un ensemble $D \subseteq V$ tel que $\Delta(G[D]) \leq r$ (tel que $\Delta(G[D]) = r$) ?

Fig. 6.2 – Définition de quelques problèmes pouvant être vus comme un cas particulier de (σ, ϱ)-domination – voir aussi la Table 6.1. Selon la nature du problème, il peut être difficile de décider si un tel ensemble existe ou de déterminer un ensemble de taille minimum ou maximum.

$\exists(\sigma, \varrho)$-Dominant

entrée : Un graphe $G = (V, E)$.

question : Le graphe G contient-il un ensemble (σ, ϱ)-dominant ?

Enum-(σ, ϱ)-Dominant

entrée : Un graphe $G = (V, E)$.

question : Lister (énumérer) tous les ensembles (σ, ϱ)-dominants de G.

#-(σ, ϱ)-Dominant

entrée : Un graphe $G = (V, E)$.

question : Déterminer le nombre d'ensembles (σ, ϱ)-dominants de G.

Max-(σ, ϱ)-Dominant

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Le graphe G contient-il un ensemble (σ, ϱ)-dominant de taille supérieure ou égale à k ?

Min-(σ, ϱ)-Dominant

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Le graphe G contient-il un ensemble (σ, ϱ)-dominant de taille inférieure ou égale à k ?

Déjà pour de nombreux choix particuliers de σ et de ϱ le problème d'existence $\exists(\sigma, \varrho)$-Dominant est NP-complet. En particulier, étant donné un graphe, les problèmes Code Parfait, Ensemble Dominant Total Parfait et Ensemble Dominant Parfait Faible sont NP-complets. De façon plus générale, Telle [Tel94] montre que les problèmes d'existences suivants sont NP-complets :

Théorème 6.1 ([Tel94]). *Les problèmes de décision* $\exists(\sigma = \{0\}, \varrho = \{q, q+1, q+2, \ldots\})$-DOMINANT *sont NP-complets pour tout entier* $q \in \{2, 3, \ldots\}$.
Le problème de décision $\exists(\sigma = \{1\}, \varrho = \mathbb{N}^*)$-DOMINANT*, appelé* couplage induit dominant*, est NP-complet.*

En revanche, le problème d'existence ENSEMBLE STABLE DOMINANT pour lequel $\sigma = \{0\}$ et $\varrho = \mathbb{N}^*$ admet un algorithme polynomial pour le résoudre puisque n'importe quel ensemble stable maximal trouvé par une stratégie gloutonne sera une solution au problème.

D'autre part, si les ensembles σ et ϱ sont finis, Telle montre le théorème suivant :

Théorème 6.2 ([Tel94]). *Le problème de décision* $\exists(\sigma, \varrho)$-DOMINANT *est NP-complet si* $0 \notin \varrho$ *et les ensembles* σ *et* ϱ *sont finis.*

Une conséquence immédiate des résultats établis par Telle est que les problèmes d'optimisation correspondants MIN-(σ, ϱ)-DOMINANT et MAX-(σ, ϱ)-DOMINANT sont également NP-difficiles lorsque les ensembles σ et ϱ respectent les conditions des théorèmes 6.1 et 6.2. Il est aussi montré dans [Tel94] qu'il y a un ensemble de problèmes pour lesquels les versions d'optimisations sont NP-difficiles alors que le problème d'existence est facile.

Théorème 6.3 ([Tel94]). *Les problèmes de décision* MIN-(σ, ϱ)-DOMINANT *et* MAX-(σ, ϱ)-DOMINANT*, avec* $\sigma = \{0, 1\}$ *et* $\varrho = \mathbb{N}^*$*, sont tous deux NP-complets alors que le problème* $\exists(\sigma, \varrho)$-DOMINANT *est facile.*

Bien sûr, le problème d'énumération est le plus difficile de tous ces problèmes puisque, dès que l'on dispose de la liste de tous les ensembles (σ, ϱ)-dominants, on peut rapidement et facilement (i) vérifier que la liste est non vide (et ainsi résoudre $\exists(\sigma, \varrho)$-DOMINANT), (ii) comparer les ensembles de la liste pour résoudre les versions de maximisation et de minimisation du problème, et (iii) compter le nombre d'ensembles dans la liste (et ainsi résoudre $\#$-(σ, ϱ)-DOMINANT).

Dans la suite nous donnons un algorithme général énumérant tous les ensembles (σ, ϱ)-dominants d'un graphe G en temps $\mathcal{O}^*(c^n)$ pour un certain $c < 2$, nécessitant seulement un espace polynomial, si l'ensemble σ est sans successeur (c'est-à-dire que σ ne contient pas deux entiers consécutifs) et si les deux ensembles σ et ϱ sont finis, ou si l'un d'eux est infini et $\sigma \cap \varrho = \emptyset$. Par conséquent, on peut également trouver un ensemble (σ, ϱ)-dominant maximum ou minimum en temps $\mathcal{O}(c^n)$ avec $c < 2$. Nous soulignons dès à présent que la technique utilisée dans cet algorithme ainsi que dans l'analyse de son temps d'exécution est une combinaison de la technique standard de branchement avec un mécanisme de *rechargement*. Ce système de rechargement est motivé par les techniques de rechargement utilisées dans des preuves, particulièrement dans des théorèmes de coloration de graphes. Nous bâptisons cette nouvelle technique générale *Brancher & Recharger*. Il est très probable que celle-ci puisse être utile pour résoudre d'autres problèmes.

Notre algorithme implique également une borne supérieure non triviale c^n, avec $c < 2$, sur le nombre d'ensembles (σ, ϱ)-dominants d'un graphe à n sommets sous les conditions

sus-citées concernant σ et ϱ. L'analyse de notre algorithme permet donc d'obtenir des conséquences combinatoires particulièrement intéressantes en soi. Pour plusieurs algorithmes exacts, leur temps d'exécution est basé sur la connaissance de théorèmes combinatoires bornant le nombre de certains objets. Par exemple l'algorithme de Fomin *et al.* [FKW04] résolvant Domination (voir sa description à la section 3.2) se sert d'un résultat de Reed [Ree96] bornant la taille d'un ensemble dominant minimum et par conséquent ceci permet de borner le nombre d'ensembles dominants. L'algorithme pour le nombre dominant de [BH06] est également basé sur une telle borne. En outre, un certain nombre d'algorithmes de coloration sont basés sur le nombre de certains ensembles stables maximaux et de sous-graphes bipartis d'un graphe [Bys04, BMS05, Epp03, Law76].

D'un autre côté, l'analyse du temps d'exécution au pire des cas d'algorithmes Brancher & Réduire permet souvent d'établir des bornes supérieures combinatoires. Le plus fameux résultat combinatoire de ce type est le théorème bien connu de Moon et Moser établissant que le nombre maximum de cliques maximales ou encore d'ensembles stables maximaux d'un graphe a n sommets est borné par $3^{n/3}$ [MM65]. Alors que la preuve originale est purement combinatoire, elle pourrait être aisément transformée en un algorithme Brancher & Réduire énumérant tous les ensembles stables maximaux. Les techniques utilisées dans l'analyse d'algorithmes exacts ont été utilisées plus tard pour obtenir par exemple des bornes sur le nombre d'ensembles dominants minimaux, d'ensembles de sommets en retour minimaux (en anglais *feedback vertex sets*) et de sous-graphes r-réguliers maximaux [FGP06, FGPS05, GRS06]. De façon générale, les algorithmes exacts et leurs analyses semblent être des outils particulièrement utiles pour obtenir de tels résultats combinatoires (à un facteur polynomial près).

Finalement, nous présentons un algorithme utilisant le paradigme assez peu connu Trier & Chercher et dont une description est donnée à la section 2.3.3, pour trouver les ensembles $(\{p\},\{q\})$-dominants de taille maximum et de taille minimum, ainsi que pour compter le nombre d'ensembles $(\{p\},\{q\})$-dominants en temps $\mathcal{O}^*(2^{n/2})$. Dans l'article [Woe03], Woeginger explique comment créer des algorithmes utilisant ce paradigme pour résoudre le problème Somme d'un sous-ensemble (Subset Sum en anglais) et le problème Sac à dos Binaire (Binary Knapsack en anglais). L'idée fondamentale est une utilisation intelligente de tris et de recherches. En utilisant une astuce nous montrerons également qu'il est possible de résoudre avec cette technique le problème (σ,ϱ)-domination non seulement limité à des singletons.

6.2 Observations préliminaires et théorème principal

Nous appelons un ensemble d'entiers *sans successeur* s'il ne contient pas deux entiers consécutifs, autrement dit S est sans successeur si pour tout entier $i \in S$ alors $(i+1) \notin S$. Dans la suite, nous utilisons les notations $\sigma_{\max}$ et $\varrho_{\max}$ pour désigner respectivement $\max \sigma$ et $\max \varrho$. On pose $\sigma_{\max} = \infty$ si l'ensemble σ est infini et $\varrho_{\max} = \infty$ si ϱ est infini. Nous désignons $\mathbb{N}^*$ l'ensemble des entiers strictement positifs et par $\mathbb{N}$ l'ensemble des entiers positifs, c'est-à-dire $\mathbb{N} = \mathbb{N}^* \cup \{0\}$.

Un graphe à n sommets peut contenir jusqu'à 2^n ensembles (σ, ϱ)-dominants ; par exemple si $0 \in \sigma \cap \varrho$ alors le graphe sans arête en contient 2^n. Un exemple moins trivial est obtenu pour $\sigma = \varrho = \{0, 1, \ldots, d\}$ puisque n'importe quel sous-ensemble de sommets dans un graphe de degré maximum d est un ensemble (σ, ϱ)-dominant.

Ainsi, si nous voulons un algorithme d'énumération significativement plus rapide que $\Theta(2^n)$, nous sommes forcés d'imposer quelques restrictions sur les ensembles σ et ϱ.

La condition essentielle requise par notre algorithme est que l'ensemble σ soit sans successeur. Néanmoins, quelques exemples simples montrent que cette condition seule n'est pas suffisante. Par exemple si σ et ϱ sont tous deux infinis et sans successeur, il est impossible d'énumérer tous les ensembles (σ, ϱ)-dominants en un temps $\mathcal{O}^*(c^n)$ pour un certain $c < 2$. En effet, soit σ l'ensemble des entiers pairs, soit ϱ l'ensemble des entiers impairs et soit $G = K_n$ un graphe complet. Alors le graphe contient 2^{n-1} ensembles (σ, ϱ)-dominants : tous les sous-ensembles de cardinalité impaire sont une solution au problème. De même si σ est sans successeur et est fini, mais les ensembles σ et ϱ ne sont pas disjoints : si on considère $\sigma = \{0\}$ et $\varrho = \mathbb{N}$ alors le graphe en étoile $G = K_{1,n-1}$ contient $2^{n-1}+1$ ensembles (σ, ϱ)-dominants (voir la figure 6.3).

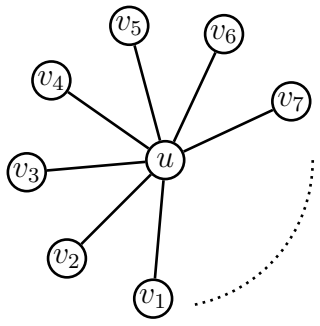


Fig. 6.3 – Un graphe $K_{1,n-1}$ pour lequel il existe $2^{n-1} + 1$ ensembles $(\{0\}, \mathbb{N})$-dominants : $\{D$ tels que $D \subseteq \{v_1, \ldots, v_{n-1}\}\} \cup \{u\}$.

Une autre observation concerne les graphes non connexes. Le nombre d'ensembles (σ, ϱ)-dominants dans un tel graphe est égal au produit des nombres d'ensembles (σ, ϱ)-dominants pour ses composantes connexes. C'est pourquoi dans la suite nous ne considérerons que des graphes connexes. Néanmoins, l'analyse de nos algorithmes est aussi valable pour des graphes sans sommet isolé, ce qui est plus intéressant pour le résultat combinatoire principal de ce chapitre :

Théorème 6.4. *Si l'ensemble σ est sans successeur et si l'une des conditions suivantes est vérifiée :*

- *les ensembles σ et ϱ sont tous deux finis, ou*
- *σ ou ϱ est fini et $\sigma \cap \varrho = \emptyset$,*

alors tout graphe sans sommet isolé contient au plus c^n ensembles (σ, ϱ)-dominants, où $c = c_{\sigma,\varrho} < 2$ est une constante qui dépend des ensembles σ et ϱ. De plus, tous ces ensembles (σ, ϱ)-dominants peuvent être énumérés en temps $\mathcal{O}^*(c^n)$, où c est la même constante.

Nous allons prouver ce théorème de manière algorithmique dans la prochaine section. Comme nous l'avons déjà annoncé, l'algorithme ne résout pas seulement le problème d'énumération Enum-(σ, ϱ)-Dominant en temps $\mathcal{O}^*(c^n)$, mais aussi les problèmes $\exists(\sigma, \varrho)$-Dominant, $\#$-(σ, ϱ)-Dominant, Max-(σ, ϱ)-Dominant et Min-(σ, ϱ)-Dominant ; seul le facteur polynomial dans la notation $\mathcal{O}^*$ diffère. Nous précisons dès maintenant que la constante c dépend uniquement des valeurs de $\sigma_{\max} = \max \sigma$ et $\varrho_{\max} = \max \varrho$.

De récents travaux de Gupta *et al.* [GRS06] donnent des algorithmes d'énumération et des bornes supérieures et inférieures sur le nombre de sous-graphes r-réguliers maximaux d'un graphe donné en entrée. Les sous-graphes r-réguliers induits sont précisément les ensembles $(\{r\}, \mathbb{N})$-dominants. Leurs résultats sont donc en relation avec notre travail. Comme on peut s'y attendre pour des (σ, ϱ) particuliers, leurs bornes sont meilleures que nos bornes générales.

6.3 L'algorithme Brancher & Recharger

6.3.1 Description de l'algorithme

À partir de maintenant, nous supposons que les ensembles σ et ϱ satisfont les hypothèses énoncées au théorème 6.4. L'algorithme que nous proposons est basé sur un nouveau paradigme bâptisé *Brancher & Réduire* et utilise seulement une règle de branchement simple. Pour garantir un temps d'exécution plus rapide que 2^n, le branchement est combiné à un mécanisme de rechargement. L'idée principale de l'algorithme est de garantir qu'à n'importe quelle étape de branchement sur un sommet v choisi, la mesure du graphe donné en entrée décroît au moins par 1 lorsque le sommet v est rejeté de la solution et par $1 + \epsilon$ lorsque le sommet v est selectionné dans les ensembles (σ, ϱ)-dominants candidats générés jusqu'à ce moment de l'exécution. La constante $\epsilon > 0$ dépend des ensembles σ et ϱ et plus précisément des valeurs de $\sigma_{\max}$ et de $\varrho_{\max}$. L'utilisation d'un vecteur de branchement $(1, 1 + \epsilon)$ avec $\epsilon > 0$ implique immédiatement un temps d'exécution plus rapide que 2^n puisque la plus grande racine positive de l'équation $x^{1+\epsilon} = 1 + x^{\epsilon}$ est inférieure à 2 pour tout $\epsilon > 0$.

Initialement tous les sommets $v \in V$ d'un graphe $G = (V, E)$ sans sommet isolé sont affectés d'un poids $w(v) = 1$. On désigne le poids total du graphe G par $w(G) = \sum_{v \in V} w(v)$. Ainsi au début de l'exécution on a l'égalité $w(G) = n$. L'algorithme construit récursivement les ensembles candidats S pour des ensembles (σ, ϱ)-dominants de G. Il appelle récursivement la procédure `SigmaRho` qui consiste en trois sous-routines : `Forcer` (qui se charge d'identifier les sommets qui doivent ou non être placés dans S), `Recharger` (qui prépare le terrain pour la prochaine sous-routine en envoyant des charges de valeur ϵ à travers certaines arêtes), et `Brancher` (le cœur de l'algorithme, puisque responsable du temps d'exécution exponentiel de l'algorithme et de la base de la fonction exponentielle bornant ce temps d'exécution). Ces trois sous-routines et la procédure travaillent avec le même graphe G et le laissent inchangé ainsi qu'avec une variable globale L qui est la liste des ensembles candidats S. Leurs paramètres sont S, $\overline{S}$ (contenant les sommets rejetés de

l'ensemble candidat), une fonction de pondération w et un graphe orienté auxiliaire H qui est une orientation d'un sous-graphe couvrant de G. Le graphe H trace les mouvements de charges dues au rechargement. De plus, les sous-routines `Forcer`, `Recharger` et `Brancher` sont appelées sur un certain sommet libre v. À chaque étape de l'algorithme, nous dirons d'un sommet v qu'il est *libre* s'il n'appartient pas à $S \cup \overline{S}$. Les sommets libres ont un poids positif, les sommets alloués ont un poids nul. Une fois qu'un sommet est alloué à S (on dit alors que le sommet est *selectionné*) ou à $\overline{S}$ (le sommet est alors dit *rejeté*) il ne change plus de statut au cours des appels récursifs suivants. Nous allons maintenant décrire les détails des pseudocodes des algorithmes.

La variable globale L et le graphe d'entrée G ne sont pas mentionnés dans les paramètres d'appel des algorithmes.

Le choix pour ϵ est donné par :

$$\epsilon = \begin{cases} \frac{1}{1+\max(p,q)} & \text{si les deux ensembles } \sigma \text{ et } \varrho \text{ sont finis,} \\ \frac{1}{1+\min(p,q)} & \text{si au moins un des ensembles } \sigma \text{ ou } \varrho \text{ est fini et } \sigma \cap \varrho = \emptyset. \end{cases}$$

Procédure `SigmaRho`$(S, \overline{S}, w, H)$
  si *il n'existe plus de sommet libre* **alors**
    $L \leftarrow L \cup \{S\}$
  sinon
    Soit v **le dernier sommet libre dans l'ordre BFS de** V
    si $v = v_1$ **alors**
      `/*` v_1 `est le premier sommet dans l'ordre BFS pré-calculé dans l'algorithme Main-EnumSigmaRho (voir plus loin) */`
      $L \leftarrow L \cup \{S, S \cup \{v\}\}$
    sinon
      `Forcer`$(v, S, \overline{S}, w, H)$
      si `Forcer` *s'est arrêté* **alors**
        Stop
      si v *est encore libre* **alors**
        `Recharger`$(v, S, \overline{S}, w, H)$
        `Brancher`$(v, S, \overline{S}, w, H)$
      sinon
        `SigmaRho`$(S, \overline{S}, w, H)$

Sous-routine `Recharger`$(v, S, \overline{S}, w, H)$
  si $w(v) < 1$ **alors**
    soit $\{w_1, \ldots, w_t\} = \{x$ **tels que** $(vx) \in E(H)\}$
    pour $i = 1$ **à** t **faire**
      soit u_i **un autre voisin libre de** w_i **dans** G
    pour $i = 1$ **à** t **faire**
      $w(u_i) \leftarrow w(u_i) - \epsilon$
      $E(H) \leftarrow (E(H) \cup \{u_i w_i\}) \setminus \{v w_i\}$
    $w(v) \leftarrow 1$

Remarque. *Notons que $w_1, \ldots, w_t$ sont des sommets distincts, alors que les sommets $u_1, \ldots, u_t$ ne le sont pas nécessairement. Si un certain sommet u est le sommet libre choisi par plusieurs, disons k sommets de $w_1, \ldots, w_t$, alors son poids décroît de $k\epsilon$ et k arêtes partant de u sont ajoutées à H. Le lemme 6.8 montre qu'un sommet w_i possède toujours un autre voisin libre dans G.*

Sous-routine `Brancher`$(v, S, \overline{S}, w, H)$
  1. $S' \leftarrow S$; $\overline{S'} \leftarrow \overline{S} \cup \{v\}$; $w' \leftarrow w$; $w'(v) \leftarrow 0$; $H' \leftarrow H$
    `SigmaRho`$(S', \overline{S'}, w', H')$
  2. **soit** u **un voisin libre de** v
    $S \leftarrow S \cup \{v\}$; $w(v) \leftarrow 0$; $w(u) \leftarrow w(u) - \epsilon$; $E(H) \leftarrow E(H) \cup \{uv\}$
    `SigmaRho`$(S, \overline{S}, w, H)$

Remarque. *Comme habituellement dans les algorithmes de type Brancher & Réduire, les valeurs des paramètres S, $\overline{S}$, w et H aux étapes 1 et 2 sont ceux passés en paramètre à la sous-routine.*

La dernière sous-routine dépend des ensembles σ et ϱ et du choix pour la valeur de ϵ. Si les deux ensembles sont finis alors $\epsilon \cdot (1 + \max(p, q)) = 1$ et on définit `Forcer` comme étant la sous-routine `Forcer-a` (voir ci-après). Si $\sigma \cap \varrho = \emptyset$ alors $\epsilon \cdot (1 + \min(p, q)) = 1$, et on définit la sous-routine `Forcer` comme étant `Forcer-b`. Notons que dans ce cas au moins l'un des ensembles σ ou ϱ doit être fini.

Sous-routine `Forcer-a`$(v, S, \overline{S}, w, H)$
  si $\exists x \in S$ *tel que* v *est son unique voisin libre* **alors**
    cas où
      $|N(x) \cap S| \in \sigma$ **alors** $\overline{S} \leftarrow \overline{S} \cup \{v\}, w(v) \leftarrow 0$
      $|N(x) \cap S| + 1 \in \sigma$ **alors** $S \leftarrow S \cup \{v\}, w(v) \leftarrow 0$
      $\{|N(x) \cap S|, |N(x) \cap S| + 1\} \cap \sigma = \emptyset$ **alors Stop**
  si $\exists x$ *tel que* $|N(x) \cap S| > \max\{\sigma_{\max}, \varrho_{\max}\}$ **alors Stop**

Sous-routine `Forcer-b`$(v, S, \overline{S}, w, H)$
tant que ($\exists x$ *tel que* x *est libre et* $|N(x) \cap S| > \min\{\sigma_{\max}, \varrho_{\max}\}$) **ou**
($\exists y \in S$ *avec un unique voisin libre* z) **ou** ($\exists$ *un sommet libre* u *sans voisin libre*)
faire
  soit x **ou** y, z **ou** u **de tels sommets**
  cas où
    $|N(x) \cap S| > \max\{\sigma_{\max}, \varrho_{\max}\}$ **alors Stop**
    $|N(x) \cap S| > \sigma_{\max}$ **alors** $\overline{S} \leftarrow \overline{S} \cup \{x\}; w(x) \leftarrow 0$
    $|N(x) \cap S| > \varrho_{\max}$ **alors** $S \leftarrow S \cup \{x\}; w(x) \leftarrow 0$
    $|N(y) \cap S| \in \sigma$ **alors** $\overline{S} \leftarrow \overline{S} \cup \{z\}; w(z) \leftarrow 0$
    $|N(y) \cap S| + 1 \in \sigma$ **alors** $S \leftarrow S \cup \{z\}; w(z) \leftarrow 0$
    $\{|N(y) \cap S|, |N(y) \cap S| + 1\} \cap \sigma = \emptyset$ **alors Stop**
    $|N(u) \cap S| \in \sigma$ **alors** $S \leftarrow S \cup \{u\}; w(u) \leftarrow 0$
    $|N(u) \cap S| \in \varrho$ **alors** $\overline{S} \leftarrow \overline{S} \cup \{u\}; w(u) \leftarrow 0$
    $|N(u) \cap S| \notin \sigma \cup \varrho$ **alors Stop**

Nous avons décrit la procédure récursive et ses sous-routines ; l'algorithme complet, appelé `EnumSigmaRho-Principal` décrit ci-après, peut être formalisé comme un appel à la procédure `SigmaRho`. De plus, l'algorithme effectue les pré-traitements nécessaires et la vérification finale de chacun des ensembles ajoutés à la liste L.

Algorithm `EnumSigmaRho-Principal`(G)
Pré-traitement : Choisir arbitrairement un sommet v_1 et ordonner l'ensemble des sommets de G selon un ordre BFS commençant du sommet v_1.
Initialisation : $L \leftarrow \emptyset\,; S \leftarrow \emptyset\,; \overline{S} \leftarrow \emptyset\,; H \leftarrow (V(G), \emptyset)$
  pour $v \in V(G)$ **faire** $w(v) \leftarrow 1$
`SigmaRho`$(S, \overline{S}, w, H)$
pour tous les $S \in L$ **faire**
  si S *n'est pas un ensemble* (σ, ϱ)*-dominant de* G **alors**
    $L \leftarrow L \setminus \{S\}$
retourner L

L'ordre BFS (en anglais *breadth first search*, en français *largeur d'abord*) est l'ordre de visite des sommets en appliquant l'algorithme de parcours en largeur sur un graphe. Cet algorithme de parcours s'exécute en $\mathcal{O}(n + m)$ et visite d'abord tous les sommets situés à distance k d'un sommet appelé racine, avant de visiter les sommets situés à distance $k + 1$. (voir par exemple [CLRS02]).

6.3.2 Preuve de validité de l'algorithme

On montre la validité de l'algorithme à travers les lemmes suivants et du fait qu'il branche sur chaque sommet dont l'appartenance à S ou à $\overline{S}$ n'est pas forcée.

Lemme 6.5. (invariant de boucle) *Au moment de chaque appel à la procédure* `SigmaRho`*, les invariants suivants sont respectés :*

1. *si x est un sommet libre alors $w(x) = 1 - d\epsilon$, où d est le degré sortant du sommet x dans H ;*
2. *le graphe H est l'union disjointe d'étoiles orientées vers l'extérieur, et si l'arête $\{xy\} \in E(H)$ alors cela implique que $y \in S$;*
3. *on a $w(x) = 0$ pour tout $x \in S \cup \overline{S}$;*
4. *on a $w(x) \geq 0$ pour tout sommet libre x, et de plus on a $w(x) > 0$ après l'exécution de la sous-routine* `Forcer`*.*

Démonstration.

1. Le poids d'un sommet libre. Au début, le poids de tous les sommets est égal à 1 et le graphe H est sans arête. Ainsi le degré sortant de tout sommet de H est égal à 0. L'invariant est vérifié par induction sur le nombre d'appels récursifs. Les poids des sommets libres sont modifiés au cours des exécutions des sous-routines `Recharger` et `Brancher`, et dans chaque cas le nombre de ϵ qui sont soustraits (ou ajoutés) au poids d'un sommet est le même que le nombre d'arêtes orientées qui partent de ce sommet et qui sont ajoutés à H (respectivement supprimés de H).

2. La structure de H. Au début le graphe H est sans arête. Il est modifié par les sous-routines `Recharger` et `Brancher`. Lors du rechargement, les arêtes vw_i sont remplacées par u_iw_i et lors du branchement l'arête uv est ajoutée. Dans chacun de ces cas l'extrémité est un sommet alloué à S et tout sommet de S est le sommet d'arrivée d'au plus un arc orienté de H. Dans le cas d'un branchement, cela est dû au fait que $w(v) = 1$ avant que `Brancher` ne soit appelé sur v, et donc v n'a ni arc entrant, ni arc sortant.

3. Poids d'un sommet alloué. On voit facilement que le poids d'un sommet alloué à S ou à $\overline{S}$ devient 0 au moment où il est alloué.

4. Les poids des sommets libres sont positifs. Cela est garanti par les sous-routines `Forcer`. Distinguons les deux cas dépendant de σ et de ϱ :

4a. Si les ensembles σ et ϱ sont tous deux finis, on utilise `Forcer-a` : un sommet libre noté x devrait avoir un poids 0 ou inférieur à 0 seulement si son degré sortant $t > \max(\sigma_{\max}, \varrho_{\max})$ dans H. Mais alors x doit avoir t voisins dans S. De façon certaine, au début du premier appel à la procédure `SigmaRho`, il n'existe pas un tel sommet. Le nombre de voisins en S peut augmenter durant la première partie de l'exécution de la sous-routine `Forcer-a`, mais cela serait immédiatement remarqué dans la seconde partie de cette sous-routine, et l'exécution serait arrêtée. La seule autre possibilité est durant la seconde partie de la sous-routine `Brancher`, lorsque v est selectionné dans S. Dans ce cas le poids du

sommet u, le voisin libre de v, était positif avant que `Brancher` soit appelée et deviendrait 0 pour un court moment – la sous-routine `Forcer-a` lors de l'appel suivant à `SigmaRho` découvrirait que u a trop de voisins en S et arrêterait l'exécution, à moins que cet appel ne soit lui-même le dernier, et nous sommes dans ce cas dans une feuille de l'arbre de recherche correspondant à l'exécution de l'algorithme.

4b. Si l'un des ensembles σ ou ϱ est fini et $\sigma \cap \varrho = \emptyset$, on utilise `Forcer-b` : un sommet libre noté x devrait avoir un poids 0 ou inférieur à 0 seulement si son degré sortant $t > \min(\sigma_{\max}, \varrho_{\max})$ dans H, et donc au moins autant de voisins dans S. Comme précédemment, un tel sommet est découvert par la sous-routine `Forcer-b` et alloué à S ou à $\overline{S}$ (ou alors l'exécution s'arrête). Après cela, le nombre de voisins en S d'un sommet libre peut être augmenté seulement dans la seconde partie la sous-routine `Brancher`. Dans ce cas, le poids d'un tel sommet libre, appelé u dans la sous-routine, décroît seulement de un ϵ, et puisqu'il était positif, il devient égal à 0 pour un court instant. La sous-routine `Forcer-b` lors de l'appel suivant à `SigmaRho` découvre que le sommet u a trop de voisins dans S et alloue u ; ou s'arrête ou contribue à la liste L des ensembles candidats si on est dans une feuille de l'arbre de recherche correspondant à l'exécution de l'algorithme. □

Lemme 6.6. (Arrêt) *Si la sous-routine* `Forcer` *s'arrête avec les valeurs courantes S et $\overline{S}$, alors G ne contient pas d'ensemble (σ, ϱ)-dominant M tel que $S \subseteq M \subseteq V \setminus \overline{S}$.*

Démonstration.

a) Les ensembles σ et ϱ sont tous deux finis, on utilise `Forcer-a`. Si `Forcer-a` s'arrête à cause d'un sommet x qui a plus que $\max(\sigma_{\max}, \varrho_{\max})$ voisins dans S, alors un tel ensemble S ne peut pas être un sous ensemble de n'importe quel ensemble (σ, ϱ)-dominant M. En effet, si $x \in M$ alors on a que $|N(x) \cap M| \geq |N(x) \cap S| > \sigma_{\max} = \max \sigma$ et $|N(x) \cap M|$ ne peut être une valeur de σ ; de même $|N(x) \cap M| \geq |N(x) \cap S| > \varrho_{\max} = \max \varrho$ et $|N(x) \cap M|$ ne peut être une valeur de ϱ si $x \notin M$. Si `Forcer-a` s'arrête à cause qu'un certain sommet $x \in S$ a un unique voisin libre v, mais que ni $|N(x) \cap S|$ ni $|N(x) \cap S| + 1$ ne sont dans σ, alors aucun ensemble M contenant S est un ensemble (σ, ϱ)-dominant puisque $|N(x) \cap M|$ est égal à $|N(x) \cap S|$ ou à $|N(x) \cap S| + 1$, dependant du fait que $v \in M$ ou pas.

b) L'un des ensembles σ ou ϱ est fini et $\sigma \cap \varrho = \emptyset$, on utilise `Forcer-b`. Les deux premières raisons pour que `Forcer-b` s'arrête sont les mêmes que précédemment. Si la sous-routine s'arrête à cause d'un certain sommet libre x pour lequel $|N(x) \cap S|$ n'appartient ni à σ ni à ϱ, alors aucun ensemble M avec $S \subseteq M$ ne peut être un ensemble (σ, ϱ)-dominant puisque x ne peut ni appartenir à M, ni être à l'extérieur de M. □

Lemme 6.7. (Nécessité) *Si à un certain moment, avec pour valeurs courantes S et $\overline{S}$, la sous-routine* `Forcer` *veut placer un sommet x dans S (respectivement dans $\overline{S}$), alors pour tout ensemble (σ, ϱ)-dominant de G tel que $S \subseteq M \subseteq V \setminus \overline{S}$, le sommet x appartient aussi à M (respectivement x n'appartient pas à M).*

Démonstration.

On distingue deux cas selon que `Forcer-a` ou `Forcer-b` est utilisée. Dans les deux cas on suppose que M est un ensemble (σ, ϱ)-dominant tel que $S \subseteq M \subseteq V \setminus \overline{S}$.

a) Les ensembles σ et ϱ sont tous deux finis, on utilise la sous-routine `Forcer-a`. Supposons que v est l'unique voisin libre de $x \in S$ et que $|N(x) \cap S| \in \sigma$. Alors $|N(x) \cap S| + 1 \notin \sigma$ car l'ensemble σ est sans successeur. Ainsi v ne peut appartenir à M puisque dans ce cas $|N(x) \cap M| = |N(x) \cap S| + 1 \notin \sigma$. De la même façon, $|N(x) \cap S| + 1 \in \sigma$ implique $|N(x) \cap S| \notin \sigma$ et donc le sommet v doit appartenir à M puisque c'est la seule possibilité d'ajouter un voisin de x à M.

b) L'un des ensembles σ ou ϱ est fini et $\sigma \cap \varrho = \emptyset$, on utilise la sous-routine `Forcer-b`. Si z est l'unique voisin libre d'un sommet $y \in S$, on utilise le même argument que précédemment. Si x est un sommet libre tel que $|N(x) \cap S| > \sigma_{\max}$ (et $\leq q$ puisque la sous-routine ne s'est pas arrêtée à l'étape précédente), x ne peut appartenir à M parce que $|N(x) \cap M| \geq |N(x) \cap S| > \sigma_{\max} = \max \sigma$ et $|N(x) \cap M|$ ne peut être dans σ, alors que si $|N(x) \cap S| > \varrho_{\max}$, alors x ne peut être en dehors de M car $|N(x) \cap M| \geq |N(x) \cap S| > \varrho_{\max} = \max \varrho$ et $|N(x) \cap M|$ ne peut être dans ϱ. Finalement, si le sommet u est un sommet libre sans voisin libre, alors $|N(u) \cap M| = |N(u) \cap S|$ et l'appartenance de u à M est déterminée de façon unique puisque $\sigma \cap \varrho = \emptyset$. □

Lemme 6.8. (Correction) *Les sous-routines* `Recharger` *et* `Brancher` *peuvent toujours être exécutées.*

Démonstration. La sous-routine `Forcer` garantit qu'aucun voisin de v dans S n'a le sommet v comme son unique voisin libre. Dans la sous-routine `Recharger`, vw_i est un arc de H et ainsi $w_i \in S$ pour tout $i = 1, \ldots, t$. Cela implique que chaque sommet w_i possède un autre voisin libre et donc la sous-routine `Recharger` peut bien s'exécuter.
Pour la sous-routine `Brancher`, on distingue deux cas :

`Forcer-a` : On note que les sommets de G sont alloués à S ou à $\overline{S}$ seulement quand on essaye de brancher dessus au cours de la sous-routine `Forcer-a` précédente, ou dans la sous-routine `Brancher` elle même. Ainsi lorsqu'on considère v comme étant le dernier sommet libre dans l'ordre BFS des sommets de G, soit $v = v_1$ est la racine (et alors on n'a pas à s'inquiéter de vérifier quoi que ce soit, on ajoute simplement S et $S \cup \{v\}$ à la liste L des candidats) ou alors v a un prédécesseur u dans l'arbre BFS de G. Ce sommet u apparaît plus tôt dans l'ordre BFS de G et il n'y a donc pas eu d'essai pour brancher dessus jusqu'à présent, et donc il est libre au moment où v est traité. Si u est déjà dans S ou dans $\overline{S}$ alors c'est que u a été forcé dans S ou $\overline{S}$. Supposons que u a été sélectionné ou rejeté par `Forcer-a`. Alors cela se produit seulement lorsqu'on a branché sur ce sommet u, ce qui est impossible puisqu'il est avant v dans l'ordre BFS.

`Forcer-b` : Si le sommet v n'a pas de voisin libre alors v, renommé en u, devient alloué dans l'exécution précédente de la sous-routine `Forcer-b` ou la sous-routine est arrêtée, mais dans aucun de ces cas le sommet v ne peut rester libre pour un futur branchement. Par conséquent le sommet v a bien un voisin libre. □

6.3.3 Analyse du temps d'exécution

Le poids d'une instance $(G, w, S, \overline{S}, H)$ est défini par $w(G) = \sum_{v \in V} w(v)$. Lors de chaque branchement sur un sommet v, cette mesure de la taille de l'entrée décroît de 1

lorsque v est rejeté et elle décroît de $1+\epsilon$ quand v est séléctionné. Dans la terminologie utilisée à la section 2.3.1.2 (page 44), on dit alors que le vecteur de branchement est $(1, 1+\epsilon)$. On constate que le temps d'exécution de chaque appel à `SigmaRho`, hormis les appels récursifs, est polynomial. Donc le temps d'exécution total est $\mathcal{O}^*(T)$ où T est le nombre de feuilles dans l'arbre de recherche correspondant à une exécution de l'algorithme. Notons également que chaque ensemble (σ, ϱ)-dominant correspond à au moins une feuille de l'arbre de recherche.

On désigne par $T[k]$ le nombre maximum de feuilles dans l'arbre de recherche qui seraient générées par n'importe quelle exécution de notre algorithme sur une instance de poids k. À cause du vecteur de branchement, on obtient que

$$T[k] \leq T[k-1] + T[k-1-\epsilon].$$

Ainsi le nombre d'ensembles (σ, ϱ)-dominants, qui est lui aussi borné par $T[n]$, dans un graphe à n sommets ne contenant pas de sommet isolé est au plus c^n, et le temps d'exécution de notre algorithme qui énumère tous ces ensembles est $\mathcal{O}^*(c^n)$, où c est la plus grande racine du polynôme caractéristique

$$x^{1+\epsilon} - x^{\epsilon} - 1 = 0.$$

La table 6.2 donne pour quelques valeurs particulières de $\varphi = \frac{1}{\epsilon} - 1$ la base de la fonction exponentielle qui borne le temps d'exécution au pire des cas de notre algorithme. Si les ensembles σ et ϱ sont finis alors $\varphi = \max(\sigma_{\max}, \varrho_{\max})$ et si au moins l'un des ensembles σ et ϱ est fini et que $\sigma \cap \varrho = \emptyset$ alors $\varphi = \min(\sigma_{\max}, \varrho_{\max})$.

φ	0	1	2	3	4	5	6	7	8	100
c	1.6181	1.7549	1.8192	1.8567	1.8813	1.8987	1.9116	1.9216	1.9296	1.9932

Tab. 6.2 – Temps d'exécution $\mathcal{O}^*(c^n)$ au pire des cas de notre algorithme pour quelques valeurs de $\varphi = \frac{1}{\epsilon} - 1$.

On note que la valeur de c converge vers la valeur 2 quand φ tend vers l'infini (c'est-à-dire quand ϵ tend vers 0).

6.4 Des bornes inférieures exponentielles

La conséquence combinatoire de notre algorithme indique que, sous certaines conditions sur les ensembles σ et ϱ, tout graphe à n sommets sans sommet isolé contient au plus $2^{n(1-\delta)}$ ensembles (σ, ϱ)-dominants, avec $\delta > 0$.

On remarque qu'en choisissant $\sigma = \{0\}$ et $\varrho = \mathbb{N}^*$, on obtient le problème de l'ensemble stable dominant. Dans ce cas, les ensembles (σ, ϱ)-dominants sont précisément les

ensembles stables maximaux par inclusion. Notre théorème implique que le nombre d'ensembles stables maximaux d'un graphe est borné supérieurement par 1.6181^n alors que le résultat de Moon et Moser [MM65] nous dit que la valeur correcte est $3^{n/3} \approx 1.4423^n$.

Alors que la borne supérieure de Moon et Moser est atteinte, on peut vérifier cela en considérant une collection disjointe de K_3, d'autres bornes comme celle de Fomin *et al.* pour le nombre maximum d'ensembles dominants minimaux d'un graphe [FGPS05] sont très probablement sur-estimées. De la même façon, nos bornes supérieures établies par l'approche générale ne sont probablement pas égales aux bornes inférieures pour des valeurs particulières de (σ, ϱ). Il est donc naturel de chercher des bornes inférieures.

Soit σ l'ensemble des entiers pairs de l'intervalle $[0, r-1]$ et ϱ l'ensemble des entiers impairs de ce même intervalle où $r \geq 2$ est un entier positif. Considérons le graphe $G = sK_r$ constitué de l'union disjointe de s copies d'un graphe complet K_r.

Ce graphe G possède $2^{(r-1)s} = 2^{\frac{r-1}{r}n} = 2^{\frac{\max\{p,q\}}{\max\{p,q\}+1}n} = 2^{\frac{\min\{p,q\}+1}{\min\{p,q\}+2}n}$ ensembles (σ, ρ)-dominants (voir la figure 6.4).

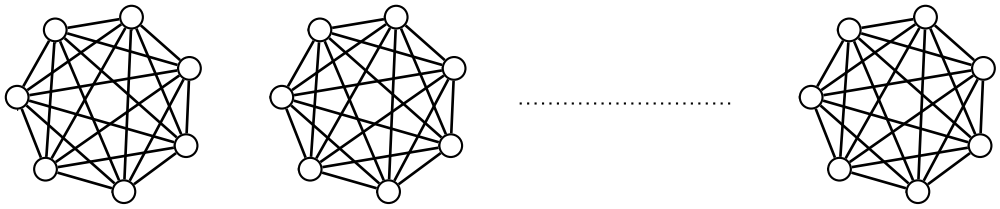


Fig. 6.4 – Un graphe composé de s K_7. Chaque sous-ensemble de sommets de taille impaire d'un K_7 est un ensemble (σ, ϱ)-dominant avec $\sigma = \{0, 2, 4, 6\}$ et $\varrho = \{1, 3, 5\}$.

Comme σ et ϱ sont tous deux finis et que $\sigma \cap \varrho = \emptyset$, nous pouvons utiliser les deux variantes de notre algorithme (c'est-à-dire utiliser notre algorithme avec `Forcer-a` ou avec `Forcer-b`). La table 6.3 compare les bases des bornes supérieures exponentielles obtenues de l'analyse de notre algorithme (avec les deux variantes pour $\epsilon = \frac{1}{1+\max(p,q)}$ et pour $\epsilon = \frac{1}{1+\min(p,q)}$, notées respectivement c_{max} et c_{min}) avec la base $a_r = 2^{\frac{r-1}{r}}$ de la fonction exponentielle donnant la borne inférieure de l'exemple que nous venons de donner.

6.5 L'approche Trier & Chercher

Nous utilisons dans cette section une technique pour la conception d'algorithmes exacts basée sur le tri et la recherche. Woeginger [Woe03], un article de Horowitz et Sahni paru en 1974 [HS74] ainsi qu'un article datant de 1981 de Schroeppel et Shamir [SS81] montrent comment utiliser ce paradigme afin d'obtenir des algorithmes exponentiels pour résoudre les problèmes NP-difficiles Somme d'un sous-ensemble et Sac à dos Binaire en temps $\mathcal{O}^*(2^{n/2})$. L'emploi de cette technique pour résoudre ces deux problèmes est rappelé à la section 2.3.3 (page 53).

r	2	3	4	5	6	7	8	9	101
c_{max}	1.7549	1.8192	1.8567	1.8813	1.8987	1.9116	1.9216	1.9296	1.9932
c_{min}	1.6181	1.7549	1.8192	1.8567	1.8813	1.8987	1.9116	1.9216	1.9932
a_r	1.4142	1.5874	1.6817	1.7411	1.7817	1.8114	1.8340	1.8517	1.9863

Tab. 6.3 – Bases c_{max} et c_{min} des bornes supérieures de l'algorithme dans ses deux variantes, et base a_r de la borne inférieure pour le graphe G avec les ensembles σ et ρ précédemment décrits.

Nous étudions dans la suite l'utilité de *Trier & Chercher* pour construire des algorithmes pour le problème $\exists(\sigma, \varrho)$-Dominant défini à la section 6.1. On rappelle que l'objet du problème est de décider si un graphe possède un ensemble (σ, ϱ)-dominant. Ces problèmes sont polynomiaux ou NP-complets selon le choix pour (σ, ϱ) (voir [Tel94]). Par exemple, décider si un graphe admet un ensemble stable dominant, c.-à-d. $\sigma = \{0\}$ et $\varrho = \mathbb{N}^*$, admet un algorithme polynomial puisque n'importe quel ensemble stable maximal par inclusion est solution et qu'un tel ensemble peut facilement être trouvé par une stratégie gloutonne.

Le problème NP-complet $\exists(\{0\}, \{1\})$-Dominant est appelé Code Parfait et peut être résolu en temps $\mathcal{O}(1.1730^n)$ [DJB04] (voir aussi [Dah06]). L'algorithme est basé sur la résolution du problème de satisfaction exacte, appelé XSAT, et l'approche a été généralisée au problème X$_i$SAT. Ce problème défini pour $i \geq 0$ attend en entrée une formule en forme normale conjonctive et doit décider s'il existe une affectation des variables telle que chaque clause contient précisément i littéraux évalués à vrai. Ces algorithmes utilisent *Trier & Chercher* et leurs temps d'exécution est $\mathcal{O}^*(2^{n/2})$.

Notons que pour tout entier $i \geq 0$ le problème d'existence $\exists(\{i\}, \{i+1\})$-Dominant peut être réduit au problème X$_{i+1}$SAT en associant à chaque sommet v du graphe G une clause contenant un littéral positif x_u pour chaque sommet $u \in N[v]$. La figure 6.5 propose un exemple. De façon similaire, pour un entier $i \geq 1$, le problème $\exists(\{i\}, \{i\})$-Dominant peut être résolu grâce à l'algorithme qui résoud X$_i$SAT en affectant à chaque sommet v du graphe G une clause contenant un littéral positif x_u pour chaque sommet $u \in N(v)$. Comme le nombre de variables dans la formule CNF associée au graphe G est égale au nombre de sommets de G, le temps d'exécution de ces algorithmes est $\mathcal{O}(2^{n/2} \cdot n^{\mathcal{O}(1)})$.

On s'intéresse maintenant à l'application de la technique *Trier & Chercher* pour résoudre le problème pour une collection plus large d'ensembles (σ, ϱ), et aussi pour résoudre non seulement le problème d'existence mais aussi les problèmes d'optimisation (maximum et minimum) correspondants et le problème demandant de compter le nombre d'ensembles étant une solution.

Théorème 6.9. *Le problème* $\exists(\{p\}, \{q\})$-Dominant *peut être résolu en temps* $\mathcal{O}^*(2^{n/2}) = \mathcal{O}^*(1.4143^n)$.

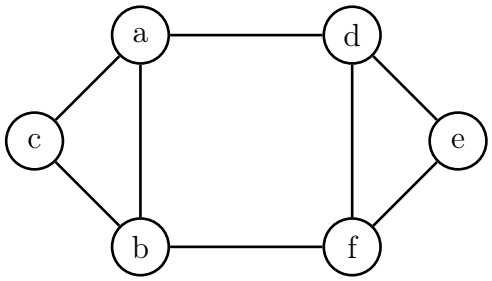


Fig. 6.5 – Exemple d'un graphe G à 6 sommets. Pour résoudre $\exists(\{i\},\{i+1\})$-DOMINANT sur G on résout X_{i+1}SAT sur la formule $\mathcal{F}_{(\{i\},\{i+1\})} = (x_a \vee x_b \vee x_c \vee x_d) \wedge (x_b \vee x_a \vee x_c \vee x_f) \wedge (x_c \vee x_a \vee x_b) \wedge (x_d \vee x_a \vee x_e \vee x_f) \wedge (x_f \vee x_b \vee x_d \vee x_e) \wedge (x_e \vee x_d \vee x_f)$.
Pour résoudre $\exists(\{i\},\{i\})$-DOMINANT sur G on résout X_iSAT sur la formule $\mathcal{F}_{(\{i\},\{i\})} = (x_b \vee x_c \vee x_d) \wedge (x_a \vee x_c \vee x_f) \wedge (x_a \vee x_b) \wedge (x_a \vee x_e \vee x_f) \wedge (x_b \vee x_d \vee x_e) \wedge (x_d \vee x_f)$.

Démonstration. Soit deux entiers $p, q \in \mathbb{N}$ et soit $G = (V, E)$ le graphe en entrée. On pose $k = \lfloor n/2 \rfloor$. Comme nous l'avons expliqué dans la description générale de la technique, l'algorithme patitionne l'ensemble des sommets en $V_1 = \{v_1, v_2, \dots v_k\}$ et en $V_2 = \{v_{k+1}, \dots, v_n\}$. Ensuite, pour chaque sous-ensemble $S_1 \subseteq V_1$, il calcule le vecteur $\vec{s_1} = (x_1, \dots, x_k, x_{k+1}, \dots, x_n)$ où

$$x_i = \begin{cases} p - |N(v_i) \cap S_1| & \text{si } 1 \leq i \leq k \text{ et } v_i \in S_1 \\ q - |N(v_i) \cap S_1| & \text{si } 1 \leq i \leq k \text{ et } v_i \notin S_1 \\ |N(v_i) \cap S_1| & \text{si } k+1 \leq i \leq n \end{cases}$$

et pour chaque sous-ensemble $S_2 \subseteq V_2$, il calcule le vecteur $\vec{s_2} = (x_1, \dots, x_k, x_{k+1}, \dots, x_n)$ où

$$x_i = \begin{cases} |N(v_i) \cap S_2| & \text{si } 1 \leq i \leq k \\ p - |N(v_i) \cap S_2| & \text{si } k+1 \leq i \leq n \text{ et } v_i \in S_2 \\ q - |N(v_i) \cap S_2| & \text{si } k+1 \leq i \leq n \text{ et } v_i \notin S_2. \end{cases}$$

Après le calcul de ces 2^{k+1} vecteurs, l'algorithme ordonne ceux correspondant à V_2 dans l'ordre lexicographique. Ensuite, pour chaque vecteur $\vec{s_1}$ (correspondant à un sous-ensemble $S_1 \subseteq V_1$), en utilisant une recherche dichotomique il teste s'il existe un vecteur $\vec{s_2}$ (correspondant à un sous-ensemble $S_2 \subseteq V_2$), tel que $\vec{s_2} = \vec{s_1}$. On remarque que le choix des vecteurs garanti que $\vec{s_2} = \vec{s_1}$ si et seulement si $S_1 \cup S_2$ est un ensemble $(\{p\},\{q\})$-dominant du graphe G. Bien sûr, un vecteur fixé $\vec{s_1}$ peut être trouvé en temps $n \cdot \log(2^{n/2}) = n^2/2$, s'il existe, dans l'ensemble des vecteurs de V_2 triés par ordre lexicographique. Par conséquent le temps d'exécution total est borné par $\mathcal{O}^*(2^{n/2})$. □

En modifiant légèrement l'algorithme décrit dans la preuve précédente, nous pouvons éviter d'avoir des copies multiples de vecteurs identiques dans la table correspondant à V_2. On obtient ainsi le corollaire suivant :

Corollaire 6.10. *Il est possible de résoudre les problèmes* Max-$(\{p\},\{q\})$-Dominant, Min-$(\{p\},\{q\})$-Dominant *et* #-$(\{p\},\{q\})$-Dominant *en temps* $\mathcal{O}^*(2^{n/2})$.

Démonstration. L'algorithme que nous avons proposé dans le théorème précédent demande d'être un peu modifié. Au lieu de trier tous les vecteurs correspondant à V_2 dans l'ordre lexicographique, les copies multiples de vecteurs identiques sont supprimées et chaque vecteur est stocké avec un entier indicant son nombre original d'occurences.

De plus, pour le problème de maximisation (respectivement de minimisation), avec chaque vecteur on stocke un sous-ensemble $S_i \subseteq V_i$ de cardinalité maximum (respectivement minimum) qui a généré ce vecteur. □

Remarque. *Le problème* Enum-$(\{p\},\{q\})$-Dominant *ne peut être résolu par cette approche. En effet, si on considère un graphe G sans arête et $p = q = 0$ alors G admet 2^n ensembles $(\{0\},\{0\})$-dominants. La difficulté essentielle qui fait que l'on ne peut utiliser l'appoche est que pour un vecteur donné de la première table, il n'y a pas nécessairement qu'un seul vecteur dans la seconde qui pourrait convenir, et si pour chaque vecteur de la première table tous les vecteurs de la seconde conviennent alors on obtient $2^{n/2} \cdot 2^{n/2} = 2^n$ solutions.*

Finalement, l'approche que nous avons établie pour des ensembles σ et ϱ réduits à des singletons peut être étendue à certains ensembles infinis pour σ et pour ϱ. Soit $m \geq 2$ un entier fixé et $k \in \{0, 1, \ldots, m-1\}$. On désigne par $k + m\mathbb{N}$ l'ensemble $\{m \cdot \ell + k : \ \ell \in \mathbb{N}\}$.

Théorème 6.11. *Soit un entier $m \geq 2$ et deux entiers $p, q \in \{0, 1, \ldots, m-1\}$. Les problèmes* $\exists(p + m\mathbb{N}, q + m\mathbb{N})$-Dominant, Max-$(p + m\mathbb{N}, q + m\mathbb{N})$-Dominant, Min-$(p + m\mathbb{N}, q + m\mathbb{N})$-Dominant *et* #-$(p + m\mathbb{N}, q + m\mathbb{N})$-Dominant *peuvent être résolus en temps* $\mathcal{O}^*(2^{n/2})$.

Démonstration. Pour établir ce théorème, on reprend les algorithmes du théorème 6.9 et du corollaire 6.10 où les composantes des vecteurs sont maintenant calculées modulo m ; il en est de même pour les additions et les soustractions des vecteurs qui sont également effectuées modulo m.

Soit deux entiers $p, q \in \{0, 1, \ldots, m-1\}$ et soit $G = (V, E)$ le graphe en entrée. On pose $k = \lfloor n/2 \rfloor$. À nouveau, nous séparons les sommets du graphe en deux sous-ensembles $V_1 = \{v_1, v_2, \ldots v_k\}$ et $V_2 = \{v_{k+1}, \ldots, v_n\}$. Maintenant, pour chacun des 2^k sous-ensembles $S_1 \subseteq V_1$, on calcule le vecteur $\vec{s_1} = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$ défini par

$$x_i = \begin{cases} (p - |N(v_i) \cap S_1|) \mod m & \text{si } 1 \leq i \leq k \text{ et } v_i \in S_1 \\ (q - |N(v_i) \cap S_1|) \mod m & \text{si } 1 \leq i \leq k \text{ et } v_i \notin S_1 \\ |N(v_i) \cap S_1| \mod m & \text{si } k+1 \leq i \leq n. \end{cases}$$

Pour chaque sous-ensemble $S_2 \subseteq V_2$, on détermine le vecteur $\vec{s_2} = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$ où

$$x_i = \begin{cases} |N(v_i) \cap S_2| \mod m & \text{si } 1 \leq i \leq k \\ (p - |N(v_i) \cap S_2|) \mod m & \text{si } k+1 \leq i \leq n \text{ et } v_i \in S_2 \\ (q - |N(v_i) \cap S_2|) \mod m & \text{si } k+1 \leq i \leq n \text{ et } v_i \notin S_2. \end{cases}$$

– Pour le problème $\exists(p + m\mathbb{N}, q + m\mathbb{N})$-DOMINANT, une fois le calcul des vecteurs réalisé, on ordonne les vecteurs correspondant à un sous-ensemble de V_2 dans l'ordre lexicographique puis, pour chaque vecteur $\vec{s_1}$, on cherche s'il existe un $\vec{s_2}$ tel que $\vec{s_2} = \vec{s_1}$. La recherche d'un tel vecteur $\vec{s_2}$ est faite en utilisant une recherche dichotomique dans l'ensemble des vecteurs de V_2 préalablement triés par ordre lexicographique. Le problème admet une solution si et seulement s'il existe un vecteur $\vec{s}$ qui correspond à la fois à un sous-ensemble S_1 de V_1 et à un sous-ensemble S_2 de V_2.

– Pour les problèmes MAX-$(p + m\mathbb{N}, q + m\mathbb{N})$-DOMINANT et MIN-$(p + m\mathbb{N}, q + m\mathbb{N})$-DOMINANT, on trie les vecteurs $\vec{s_2}$ par ordre lexicographique. Ensuite, en parcourant l'ensemble trié des 2^k vecteurs, pour chaque ensemble de vecteurs identiques (ils sont « contigus » dans l'ensemble trié), on ne garde qu'un seul vecteur avec l'ensemble $S_2 \subseteq V_2$ de cadinalité maximum (respectivement minimum) qui a produit un tel vecteur. On regarde ensuite pour chaque vecteur $\vec{s}$, correspondant à un ensemble $S_1 \subseteq V_1$, s'il existe ce même vecteur dans l'ensemble trié des vecteurs $\vec{s_2}$ qui correspond à des sous-ensembles $S_2 \subseteq V_2$. Parmi toutes les correspondances trouvées, on retourne celle pour laquelle la somme $|S_1| + |S_2|$ est maximum (respectivement minimum).

– Pour le problème $\#$-$(p + m\mathbb{N}, q + m\mathbb{N})$-DOMINANT, on trie les vecteurs $\vec{s_2}$ par ordre lexicographique. Puis, on parcourt l'ensemble trié des vecteurs $\vec{s_2}$. Pour chaque ensemble de vecteurs identiques (là encore, ils sont « contigus » dans l'ensemble trié), on ne conserve qu'un seul vecteur auquel on associe son nombre d'occurence, noté $occ(\vec{s_2})$. Ensuite, pour chaque vecteur $\vec{s_1}$ correspondant à un ensemble $S_1 \subseteq V_1$, on regarde s'il existe un vecteur $\vec{s_2}$ identique. Si c'est le cas, on ajoute $occ(\vec{s_2})$ au nombre de solutions au problème.

□

Nous montrons maintenant que la technique peut être adaptée de sorte à résoudre le problème $\exists(\sigma, \varrho)$-DOMINANT lorsque $|\sigma| + |\varrho| = 3$.

Théorème 6.12. *Le problème* $\exists(\{p_1, p_2\}, \{q\})$-DOMINANT *et le problème* $\exists(\{p\}, \{q_1, q_2\})$-DOMINANT *peuvent être résolus en temps* $\mathcal{O}^*(3^{n/2}) = \mathcal{O}^*(1.7321^n)$.

Démonstration. Nous commençons la preuve en montrant que le problème $\exists(\{p_1, p_2\}, \{q\})$-DOMINANT admet un algorithme qui s'exécute en $\mathcal{O}^*(3^{n/2})$ pour le résoudre.

Soit $p_1, p_2, q \in \mathbb{N}$ et soit $G = (V, E)$ un graphe. Nous utilisons la technique Trier & Chercher. Soit $k = \lfloor n/2 \rfloor$ et soit $V_1 = \{v_1, v_2, \dots v_k\}$ et $V_2 = \{v_{k+1}, \dots, v_n\}$ une partition des sommets de G. À chaque partition P_1 de V_1 en trois sous-ensembles notés $S^1_{p_1}$, $S^1_{p_2}$, S^1_q, on associe le vecteur $\vec{s_1} = (x_1, \dots, x_k, x_{k+1}, \dots, x_n)$ défini par

$$x_i = \begin{cases} p_1 - |N(v_i) \cap (S^1_{p_1} \cup S^1_{p_2})| & \text{si } 1 \leq i \leq k \text{ et } v_i \in S^1_{p_1} \\ p_2 - |N(v_i) \cap (S^1_{p_1} \cup S^1_{p_2})| & \text{si } 1 \leq i \leq k \text{ et } v_i \in S^1_{p_2} \\ q - |N(v_i) \cap (S^1_{p_1} \cup S^1_{p_2})| & \text{si } 1 \leq i \leq k \text{ et } v_i \in S^1_q \\ |N(v_i) \cap (S^1_{p_1} \cup S^1_{p_2})| & \text{si } k+1 \leq i \leq n. \end{cases}$$

Pour chaque partition P_2 de V_2 en trois sous-ensembles notés $S^2_{p_1}$, $S^2_{p_2}$, S^2_q, l'algorithme calcule le vecteur $\vec{s_2} = (x_1, \ldots, x_k, x_{k+1}, \ldots, x_n)$ où

$$x_i = \begin{cases} |N(v_i) \cap (S^2_{p_1} \cup S^2_{p_2})| & \text{si } 1 \leq i \leq k \\ p_1 - |N(v_i) \cap (S^2_{p_1} \cup S^2_{p_2})| & \text{si } k+1 \leq i \leq n \text{ et } v_i \in S^2_{p_1} \\ p_1 - |N(v_i) \cap (S^2_{p_1} \cup S^2_{p_2})| & \text{si } k+1 \leq i \leq n \text{ et } v_i \in S^2_{p_2} \\ q - |N(v_i) \cap (S^2_{p_1} \cup S^2_{p_2})| & \text{si } k+1 \leq i \leq n \text{ et } v_i \in S^2_q. \end{cases}$$

On note que V_1 peut être partitionné dans au plus $3^{|V_1|} = 3^k$ partitions différentes en trois sous-ensembles. Il en est de même pour V_2. Ainsi, une fois le calcul de ces $2 \cdot 3^k$ vecteurs réalisé, l'algorithme trie ceux correspondant à V_2 dans l'ordre lexicographique. Puis, comme expliqué dans la preuve du théorème 6.9, pour chaque vecteur $\vec{s_1}$ (correspondant à une partition P_1 de V_1), grâce à une recherche dichotomique, l'existence d'un vecteur $\vec{s_2}$ (correspondant à une partition P_2 de V_2) tel que $\vec{s_2} = \vec{s_1}$ est testée. Le choix des vecteurs garantit que $\vec{s_2} = \vec{s_1}$ si et seulement si $S^1_{p_1} \cup S^1_{p_2} \cup S^2_{p_1} \cup S^2_{p_2}$ est un ensemble $(\{p_1, p_2\}, \{q\})$-dominant du graphe G. Comme pour un vecteur $\vec{s_1}$ donné, le vecteur $\vec{s_2} = \vec{s_1}$ peut être trouvé en temps $n \cdot \log(3^{n/2}) = n \cdot \log_3(3^{n/2}) / \log_2(3) = \mathcal{O}(n^2)$, s'il existe, dans l'ensemble des vecteurs de V_2 triés par ordre lexicographique, on en déduit que le temps d'exécution total est borné par $\mathcal{O}^*(3^{n/2})$.

La résolution du problème $\exists(\{p\}, \{q_1, q_2\})$-DOMINANT se fait de façon très similaire. On énumère toutes les partitions P_1 (respectivement P_2) de V_1 (respectivement de V_2) en trois sous-ensembles notés S^1_p, $S^1_{q_1}$, $S^1_{q_2}$ (respectivement S^2_p, $S^2_{q_1}$, $S^2_{q_2}$) auquel on associe le vecteur $\vec{s_1}$ (respectivement $\vec{s_2}$) dont les composantes sont données par :

$$x_i = \begin{cases} p - |N(v_i) \cap (S^1_p)| & \text{si } 1 \leq i \leq k \text{ et } v_i \in S^1_p \\ q_1 - |N(v_i) \cap (S^1_p)| & \text{si } 1 \leq i \leq k \text{ et } v_i \in S^1_{q_1} \\ q_2 - |N(v_i) \cap (S^1_p)| & \text{si } 1 \leq i \leq k \text{ et } v_i \in S^1_{q_2} \\ |N(v_i) \cap (S^1_p)| & \text{si } k+1 \leq i \leq n. \end{cases}$$

Pour $\vec{s_2}$, par :

$$x_i = \begin{cases} |N(v_i) \cap (S^2_p)| & \text{si } 1 \leq i \leq k \\ p - |N(v_i) \cap (S^2_p)| & \text{si } k+1 \leq i \leq n \text{ et } v_i \in S^2_p \\ q_1 - |N(v_i) \cap (S^2_p)| & \text{si } k+1 \leq i \leq n \text{ et } v_i \in S^2_{q_1}. \\ q_2 - |N(v_i) \cap (S^2_p)| & \text{si } k+1 \leq i \leq n \text{ et } v_i \in S^2_{q_2}. \end{cases}$$

□

Finalement, en combinant l'idée de la preuve du théorème 6.12 avec les techniques utilisées dans la preuve du corollaire 6.10 (page 139), on établit le corollaire suivant :

Corollaire 6.13. *Si $|\sigma| + |\varrho| = 3$, alors il est possible de résoudre les problèmes* MAX-(σ, ϱ)-DOMINANT, MIN-(σ, ϱ)-DOMINANT *et* #-(σ, ϱ)-DOMINANT *en temps $\mathcal{O}^*(3^{n/2})$.*

De même, en combinant la preuve du théorème 6.12 avec la preuve du théorème 6.11 (page 139), on établit le théorème suivant :

Théorème 6.14. *Les problèmes* $\exists(\sigma,\varrho)$-DOMINANT, MAX-(σ,ϱ)-DOMINANT, MIN-(σ,ϱ)-DOMINANT *et* $\#$-(σ,ϱ)-DOMINANT *où*

- $\sigma = \{p_1 + m\mathbb{N}, p_2 + m\mathbb{N}\}$, $\varrho = \{q + m\mathbb{N}\}$, *avec un entier* $m \geq 2$ *et des entiers* $p_1, p_2, q \in \{0, 1, \ldots, m-1\}$;
- $\sigma = \{p + m\mathbb{N}\}$, $\varrho = \{q_1 + m\mathbb{N}, q_2 + m\mathbb{N}\}$, *avec un entier* $m \geq 2$ *et des entiers* $p, q_1, q_2 \in \{0, 1, \ldots, m-1\}$;

peuvent être résolus en temps $\mathcal{O}^*(3^{n/2})$.

6.6 Conclusion et perspectives

Nous avons montré dans ce chapitre que tous les ensembles (σ,ϱ)-dominants peuvent être énumérés en temps $\mathcal{O}(c^n)$, avec $c < 2$, si l'ensemble σ est sans successeur et si, soit σ et ϱ sont finis, soit l'un des ensembles σ or ϱ est fini et $\sigma \cap \varrho = \emptyset$. Notre algorithme implique également une borne supérieure non triviale sur le nombre maximum d'ensembles (σ,ϱ)-dominants dans un graphe ne contenant pas de sommet isolé si les ensembles σ et ϱ respectent ces mêmes conditions.

De plus, nous avons observé qu'en combinant un algorithme de type Brancher & Réduire avec un mécanisme de rechargement, cela permet d'obtenir des algorithmes d'énumération relativement simples. L'analyse du temps d'exécution de ces algorithmes est également assez facile. Il n'est d'ailleurs pas impossible que cette nouvelle technique Brancher & Recharger puisse être utile pour d'autres problèmes qui demandent, par exemple, de déterminer un plus grand sous-graphe induit respectant une propriété Π.

Il est également très intéressant d'étudier la complexité du problème qui demande d'énumérer les ensembles (σ,ϱ)-dominants pour d'autres ensembles σ et ϱ, tout comme une borne combinatoire supérieure sur leur nombre. Une question naturelle serait d'obtenir un algorithme qui s'exécute en temps $\mathcal{O}^*(N_{\sigma,\varrho}(n))$ pour énumérer tous les ensembles (σ,ϱ)-dominants, avec $N_{\sigma,\varrho}(n)$ le nombre maximum d'ensembles (σ,ϱ)-dominants.

Aussi, est-il possible de caractériser tous les ensembles (σ,ϱ) pour lesquels tout graphe à n sommets, sans sommet isolé, contient au plus $\mathcal{O}(c^n)$ avec $c < 2$ ensembles (σ,ϱ)-dominants.

Finalement, les deux problèmes suivants ont également un intérêt particulier :

- σ fini et $\varrho = \mathbb{N}^*$;
- σ et ϱ sont co-finis.

Ces deux restrictions sur les ensembles σ et ϱ regroupent des problèmes bien connus. Si l'ensemble σ est fini et l'ensemble ϱ est l'ensemble des entiers naturels, on retrouve alors des problèmes tels que ENSEMBLE STABLE, COUPLAGE INDUIT, SOUS-GRAPHE r-RÉGULIER INDUIT, SOUS-GRAPHE INDUIT DE DEGRÉ BORNÉ ou encore ENSEMBLE STABLE DOMINANT. D'autre part, si les ensembles σ et ϱ sont co-finis, c'est-à-dire leurs complémentaires sont finis, nous retrouvons des problèmes tels que DOMINATION et ENSEMBLE DOMINANT TOTAL.

Chapitre 7

Autres généralisations de la domination

Dans ce dernier chapitre, nous présentons et étudions des algorithmes exacts exponentiels pour plusieurs problèmes généralisant la domination.

En premier lieu, nous nous intéressons au problème appelé Domination Partielle. Étant donné un graphe $G = (V, E)$ et un entier t, ce problème demande de trouver un ensemble dominant du graphe de sorte qu'au moins $t \leq |V|$ sommets soient dominés par l'ensemble. L'idée du problème est donc de dominer un graphe de façon partielle pour un coût moindre qu'un ensemble dominant pour tout le graphe. Nous donnerons à la section 7.1.1 un algorithme de type Brancher & Réduire pour résoudre ce problème plus rapidement que l'algorithme trivial en $\mathcal{O}^*(2^n)$ qui énumère tous les sous-ensembles de sommets. L'algorithme proposé sera dans le même esprit que les algorithmes donnés par Grandoni puis Fomin *et al.* dans [Gra06, FGK05a] pour résoudre le problème classique Domination. Néanmoins il sera un peu plus compliqué que les algorithmes pour Domination du fait de règles de réduction différentes de celles classiquement utilisées pour Domination – en particulier, un sommet sans voisin ne doit pas être systématiquement ajouté à la solution – mais aussi du fait que certains sous-problèmes pouvant être résolus en temps polynomial demanderont plus d'efforts pour les résoudre que dans le cas de la domination classique. Par conséquent, ces modifications profondes sur le fonctionnement de l'algorithme nous obligeront à procéder à une analyse du temps d'exécution différente, bien que proche, de celle de [FGK05a]. À la section 7.1.3 (page 153), nous étudierons un algorithme exact pour résoudre un problème appelé Domination Romaine. Grâce à l'algorithme établi à la section 7.1.1 nous montrerons que le problème admet un algorithme pour le résoudre dont le temps d'exécution est borné par $\mathcal{O}(1.6183^n)$. La section 7.1.4 (page 157) se consacrera au problème k-centre partiel qui est une généralisation du problème k-centre. Nous établirons des liens entre la complexité du problème k-centre partiel et le problème Domination Partielle. En particulier, nous montrerons que k-centre partiel peut être résolu en temps $\mathcal{O}^*(c^n)$ au pire des cas si et seulement s'il est possible de résoudre Domination Partielle en temps $\mathcal{O}^*(c^n)$ au pire des cas.

Au chapitre 6 nous avons présenté un algorithme pour résoudre une généralisation du problème domination appelée (σ, ϱ)-DOMINATION. Nous avons vu que ce problème permettait notamment, pour quelques ensembles particuliers pour σ et pour ϱ, de modéliser de nombreux problèmes de graphes bien connus comme DOMINATION, DOMINATION EFFICACE, ENSEMBLE STABLE DOMINANT, ENSEMBLE STABLE, etc. (voir la table 6.1 (page 122)). À la section 7.2, nous étudions une autre généralisation du problème classique de la domination où les sommets peuvent se voir attribuer des puissances qui leur permettent de dominer des sommets jusqu'à une certaine distance. À chaque puissance est associé un coût et l'objectif est de déterminer la puissance à attribuer à chaque sommet de sorte à dominer tous les sommets du graphe et à minimiser le coût total. Après avoir donné quelques preuves de NP-complétude, nous proposerons un algorithme basé sur le paradigme de la Programmation Dynamique pour résoudre ce problème de façon générale.

7.1 Domination partielle

Le problème usuel de la domination peut être parfois trop restrictif et il peut alors être demandé de déterminer un ensemble dominant tel qu'on impose la domination d'au moins t sommets d'un graphe. Pour $t = n$, avec n le nombre de sommets du graphe, nous retrouvons le problème DOMINATION ; ce nouveau problème de domination partielle est par conséquent également NP-complet et est plus ardu à résoudre que DOMINATION. Considérer des solutions partielles n'est pas une approche nouvelle. Le problème de graphes suivant a par exemple déjà été étudié :

COUVERTURE DE SOMMETS PARTIELLE / t-COUVERTURE DE SOMMETS

entrée : Un graphe $G = (V, E)$ et deux entiers positifs k et t.

question : Existe-t-il un sous-ensemble de sommets $S \subseteq V$ tel que
(i) $|S| \leq k$; et
(ii) S couvre au moins t arêtes du graphe.

Ce problème a été introduit par Bshouty et Burroughs [BB98]. Les auteurs ont montré que le problème admet un algorithme polynomial ayant un facteur d'approximation 2. Concernant la complexité à paramètre fixé, il a été montré que le problème est traitable à paramètre fixé lorsque le paramètre considéré est t [Blä03], alors qu'il est $W[1]$ complet lorsque l'on considère le paramètre k relié à la taille d'une couverture de sommets partielle [Cai05, GNW06]. Dans [Blä03], Bläser montre plus particulièrement qu'il existe un algorithme en $2^{\mathcal{O}(t)}nm \log m$ pour résoudre t-COUVERTURE DE SOMMETS sur un graphe à n sommets et m arêtes. Il donne également un algorithme en $2^{\mathcal{O}(t)}n^2 \log n$ pour résoudre le problème t-DOMINATION.

Domination Partielle / t-Domination

entrée : Un graphe $G = (V, E)$ et deux entiers positifs k et t.

question : Existe-t-il un sous-ensemble de sommets $S \subseteq V$ tel que
(i) $|S| \leq k$; et
(ii) S domine au moins t sommets du graphe.

C'est à ce dernier problème que nous allons nous atteler dans la suite afin de proposer un algorithme exponentiel pour le résoudre de façon exacte. Étant donné un graphe G, le problème d'optimisation correspondant à t-Domination réclame de trouver un plus petit ensemble S qui domine au moins t sommets du graphe. On note $\gamma_t(G)$ la taille d'un tel ensemble S.

La section 7.1.1 présente l'algorithme *Brancher & Réduire* que nous proposons. Nous donnerons ensuite à la section 7.1.2 une analyse de son temps d'exécution au pire des cas. Au cours de ces dernières années, un problème pour lequel nous avons proposé plusieurs algorithmes linéaires ou polynomiaux pour le résoudre sur des classes de graphes telles que les graphes d'intervalles, les cographes, les graphes sans astéroïde triple, les graphes avec un d-octopus, etc. [Lie04, LKLP05], nous a particulièrement intéressé. Ce problème est appelé Domination Romaine. Nous avons cherché plusieurs approches pour le résoudre de façon exacte sur un graphe quelconque et, finalement, le meilleur algorithme que nous ayons obtenu est basé sur l'algorithme pour Domination Partielle développé dans les sections suivantes. Ainsi, à la section 7.1.3 nous montrerons que l'on peut résoudre ce problème en temps $\mathcal{O}(1.6183^n)$.

7.1.1 Un algorithme Brancher & Réduire

Dans cette section, nous donnons un algorithme pour résoudre le problème t-Domination. Afin de calculer un ensemble dominant de taille minimum, Grandoni puis Fomin *et al.* [Gra06, FGK05a] étaient passés par la résolution d'un problème plus général : le problème Couverture d'ensembles. Comme déjà précisé à la section 3.4 (page 61), il existe une transformation simple qui permet de modéliser une instance du problème domination en une instance du problème de couverture d'ensembles. Grandoni avait ainsi obtenu un algorithme en $\mathcal{O}(1.9053^n)$ (en $\mathcal{O}(1.8026^n)$ au coût d'un espace exponentiel) pour résoudre Domination, puis Fomin *et al.* avait établi un algorithme dont le temps d'exécution est $\mathcal{O}(1.5263^n)$ ($\mathcal{O}(1.5137^n)$ et espace exponentiel) pour déterminer un ensemble dominant minimum d'un graphe à n sommets. L'appoche employée par ces auteurs motive notre choix à nous intéresser à un problème plus général. Ce problème est le pendant de Domination Partielle pour la couverture d'ensembles et est défini par :

Couverture d'Ensembles Partielle / t-Couverture d'Ensembles

entrée : Un univers $\mathcal{U}$, une collection $\mathcal{S}$ de sous-ensembles de $\mathcal{U}$ et deux entiers $k \in \mathbb{N}$ et $t \in \mathbb{N}$.

question : Existe-t-il $\mathcal{C} \subseteq \mathcal{S}$ tel que
(i) $|\mathcal{C}| \leq k$; et
(ii) $|\bigcup_{C \in \mathcal{C}} C| \geq t$, c'est-à-dire que la collection $\mathcal{C}$ couvre au moins t éléments de $\mathcal{U}$.

Étant donné un graphe $G = (V, E)$ et un entier t, la résolution du problème t-Domination en utilisant un algorithme pour t'-Couverture d'Ensembles se fait en considérant l'instance $(\mathcal{U}, \mathcal{S})$ donnée par
– $\mathcal{U} = V$;
– $\mathcal{S} = \{N[v] \text{ tel que } v \in V\}$;
– $t' = t$.
Un algorithme en $\mathcal{O}^*(\alpha^{|\mathcal{U}|+|\mathcal{S}|})$ pour résoudre t-Couverture d'Ensembles implique ainsi un algorithme en $\mathcal{O}^*(\alpha^{2n})$ pour résoudre t-Domination.

L'algorithme que nous donnons pour résoudre t-Couverture d'Ensembles attend en entrée une collection $\mathcal{S}$ d'ensembles dont les éléments appartiennent à un univers $\mathcal{U}(\mathcal{S}) = \bigcup_{S \in \mathcal{S}} S$. Il prend également en entrée une collection $\mathcal{L}$ d'ensembles dont les éléments appartiennent à un univers noté $\mathcal{U}(\mathcal{L}) = \bigcup_{S \in \mathcal{L}} S$. De plus, tout élément de $\mathcal{U}(\mathcal{L})$ appartient à un unique ensemble de $\mathcal{L}$, et $\mathcal{U}(\mathcal{S}) \cap \mathcal{U}(\mathcal{L}) = \emptyset$. Le dernier paramètre attendu est un entier t indiquant le nombre minimum d'éléments de $\mathcal{U}(\mathcal{S}) \cup \mathcal{U}(\mathcal{L})$ qui doivent être couvert par un nombre minimum d'ensembles de $\mathcal{S} \cup \mathcal{L}$. Voir l'algorithme **Couverture-Partielle** ci-contre.

L'opération `del` utilisée dans l'algorithme est définie par

$$\texttt{del}(\mathcal{S}, S) = \{Z \text{ tel que } Z = R \setminus S \text{ et } Z \neq \emptyset \text{ et } R \in \mathcal{S}\}.$$

L'idée générale de l'algorithme est de distinguer les sous-ensembles en deux collections $\mathcal{S}$ et $\mathcal{L}$. La collection $\mathcal{L}$ ne contient que des ensembles S pour lesquels chaque élément de S n'apparaît dans aucun autre ensemble de $\mathcal{S} \cup \mathcal{L}$. La collection $\mathcal{S}$ contient tous les autres ensembles. La raison de cette distinction est que dans certains cas le problème peut être résolu en temps polynomial et il est donc inutile de réaliser de coûteux branchements pour déterminer la solution optimale. Nous sommes capables de résoudre le problème en temps polynomial si :
– la collection $\mathcal{S}$ est vide ; dans ce cas on ajoute les ensembles de $\mathcal{L}$ « au fur et à mesure » par ordre de cardinalité décroissante ;
– la collection $\mathcal{S}$ ne contient que des ensembles de cardinalité égale à deux ; dans ce cas on se sert des ensembles de $\mathcal{L}$ et d'un couplage d'un graphe construit à partir de $\mathcal{S}$.
Ce travail polynomial sera réalisé par la fonction **poly-CouverturePartielle** que nous détaillons dans la suite. Nous montrerons également la validité de cette fonction.

Algorithme CouverturePartielle($\mathcal{S}$, $\mathcal{L}$, t)
Entrées: Une collection $\mathcal{S}$ de sous-ensembles sur un univers $\mathcal{U}(\mathcal{S})$, une collection $\mathcal{L}$ de sous-ensembles, un entier t.
Sortie : Une plus petite collection $\mathcal{S}' \subseteq \mathcal{S} \cup \mathcal{L}$ de sous-ensembles telle que au moins t éléments de $\mathcal{U}(\mathcal{S} \cup \mathcal{L})$ soient couverts par $\mathcal{S}'$.

si $t \leq 0$ **alors**
  retourner $\emptyset$
sinon si $|\mathcal{U}(\mathcal{S} \cup \mathcal{L})| < t$ **alors**
  retourner $2^{\mathbb{N}}$
sinon si $\mathcal{S} = \emptyset$ **alors**
  retourner poly-CouverturePartielle($\emptyset$, $\mathcal{L}$, t)
sinon si $\exists S_1, S_2 \in \mathcal{S}$ *tels que* $S_1 \subseteq S_2$ **alors**
  retourner CouverturePartielle($\mathcal{S} \setminus \{S_1\}$, $\mathcal{L}$, t)
sinon si $\exists S \in \mathcal{S}$ *tels que* $\forall u \in S$, *u n'appartient pas à un autre ensemble de* $\mathcal{S}$ **alors**
  retourner CouverturePartielle($\mathcal{S} \setminus \{S\}$, $\mathcal{L} \cup \{S\}$, t)
sinon
  Choisir un ensemble $S \in \mathcal{S}$ de cardinalité maximum
  si $|S| = 2$ **alors**
    retourner poly-CouverturePartielle($\mathcal{S}$,$\mathcal{L}$, t)
  sinon
    $\mathcal{S}_{\text{IN}} \leftarrow \{S\} \cup$ **CouverturePartielle**(`del`($\mathcal{S}$, S), $\mathcal{L}$, $t - |S|$)
    $\mathcal{S}_{\text{OUT}} \leftarrow$ **CouverturePartielle**($\mathcal{S} \setminus \{S\}$, $\mathcal{L}$, t)
    retourner l'ensemble $\mathcal{S}_{\text{IN}}$ ou $\mathcal{S}_{\text{OUT}}$ de plus petite cardinalité

Nous décrivons maintenant les différentes règles utilisées par l'algorithme pour déterminer une solution optimale et montrons la validité de l'algorithme.

L'algorithme dispose de deux règles d'arrêt. Si on doit couvrir moins de 0 élément alors la solution triviale est l'ensemble vide. Si l'ensemble $\mathcal{U}(\mathcal{S} \cup \mathcal{L})$ de tous les éléments contenus dans des ensembles de $\mathcal{S}$ ou de $\mathcal{L}$ est plus petit que t alors il est impossible de trouver une solution à l'instance courante. On retourne donc $2^{\mathbb{N}}$ qui représente une collection infinie d'ensembles. Dans une implémentation de l'algorithme, on pourrait simplement retourner un marqueur spécial indiquant qu'une solution de l'instance courante n'existe pas. Aussi, si on ajoute un ensemble à la collection infinie représentée par $2^{\mathbb{N}}$, on obtient encore la collection infinie $2^{\mathbb{N}}$.

L'algorithme possède aussi quatre règles de réduction.

Premièrement si la collection $\mathcal{S}$ est vide, alors les seuls ensembles qu'il est possible de choisir afin de dominer au moins t sommets sont ceux de la collection $\mathcal{L}$. L'algorithme appelle la fonction **poly-CouverturePartielle** qui permet de déterminer, s'il existe, un sous-ensemble de $\mathcal{L}$ de taille minimum permettant de couvrir au moins t éléments de $\mathcal{U}(\mathcal{L})$. Comme annoncé précédemment, nous donnons la fonction **poly-CouverturePartielle**, montrons sa validité et établissons son temps d'exécution dans la suite.

Ensuite, s'il existe deux ensembles S_1 et S_2 de $\mathcal{S}$ tels que S_1 est inclus dans S_2, alors il est possible de supprimer S_1 de la collection $\mathcal{S}$. En effet, dans toute solution optimale $\mathcal{S}'$ contenant S_1, puisque $S_1 \subseteq S_2$ la solution $\{S_2\} \cup \mathcal{S}' \setminus \{S_1\}$ est également une solution optimale.

La troisième règle de réduction indique que s'il existe un ensemble de la collection $\mathcal{S}$ pour lequel chacun de ses éléments n'appartient à aucun autre ensemble de $\mathcal{S}$, alors on supprime cet ensemble de $\mathcal{S}$ pour l'ajouter à $\mathcal{L}$.

Enfin, la dernière règle de réduction précise que si la collection $\mathcal{S}$ ne contient que des ensembles de cardinalité égale à deux, alors on utilise la fonction **poly-CouverturePartielle** pour déterminer en temps polynomial une solution optimale.

Finalement, l'algorithme possède une unique règle de branchement. Un ensemble $S \in \mathcal{S}$ de cardinalité maximum est choisi. Le branchement réalisé est trivial : soit que S appartient à la solution optimale, soit qu'il n'y appartient pas et dans ce cas il est supprimé de $\mathcal{S}$. L'algorithme résout ensuite récursivement chacun de ces deux sous-problèmes et retourne la solution optimale obtenue de ce branchement.

Notons que les sous-ensembles ajoutés successivement à la solution optimale récursivement construite ne sont pas nécessairement, dans la version présentée de l'algorithme, les ensembles originaux de la collection $\mathcal{S}$ donnée en entrée du problème. Cela est dû au fait que lorsqu'un ensemble S est choisi par la règle de branchement, l'opération `del`$(\mathcal{S}, S)$ modifie les sous-ensembles ayant une intersection non vide avec S. Néanmoins il est possible de retrouver les ensembles originaux puisqu'à un ensemble choisi correspond précisément un ensemble de l'instance originale. Notons aussi, que cela ne contredit en rien la validité des règles de réduction et de branchement car si pour un ensemble $S \in \mathcal{S}$ correspond un ensemble original S' alors $S \subseteq S'$, et pour tout $u \in S' \setminus S$, si u n'appartient pas à S cela

implique qu'il est déjà couvert par un ensemble choisi dans la solution potentielle construite jusqu'à ce stade de l'exécution.

Fonction poly-CouverturePartielle($\mathcal{S}$, $\mathcal{L}$, t)
Entrées: Une collection $\mathcal{S}$ de sous-ensembles telle que pour tout $S \in \mathcal{S}, |S| = 2$, une collection $\mathcal{L}$ de sous-ensembles pour lesquels chaque élément n'apparaît que dans un sous-ensemble, un entier t tel que $t \leq \mathcal{U}(\mathcal{S} \cup \mathcal{L})$.
Sortie : Une plus petite collection $\mathcal{S}' \subseteq \mathcal{S} \cup \mathcal{L}$ de sous-ensembles telle qu'au moins t éléments de $\mathcal{U}(\mathcal{S} \cup \mathcal{L})$ soient couverts par $\mathcal{S}'$, c.-à-d. $|\mathcal{U}(\mathcal{S}')| \geq t$.

$\mathcal{S}' \leftarrow \emptyset$
tant que $t > 0$ *et que* $\mathcal{L}$ *contient un sous-ensemble de cardinalité au moins* 2 **faire**
  Prendre un sous-ensemble $S \in \mathcal{L}$ de cardinalité maximum
  $\mathcal{L} \leftarrow \mathcal{L} \setminus S$; $\mathcal{S}' \leftarrow S \cup \mathcal{S}'$; $t \leftarrow t - |S|$
Calculer un couplage $\mathcal{C}$ de taille maximum du graphe $G = (V, E)$ où $V = \mathcal{U}(\mathcal{S})$ et $E = \{\{u, v\}$ tel que $u, v \in V$ et $\{u, v\} \in \mathcal{S}\}$
tant que $t > 0$ *et que* $\mathcal{C} \neq \emptyset$ **faire**
  Prendre une arête $\{u, v\} \in \mathcal{C}$ correspondant à un ensemble $S \in \mathcal{S}$
  $\mathcal{C} \leftarrow \mathcal{C} \setminus \big\{\{u, v\}\big\}$; $\mathcal{S} \leftarrow del(\mathcal{S}, S)$; $\mathcal{S}' \leftarrow S \cup \mathcal{S}'$; $t \leftarrow t - 2$
tant que *il existe* $S_1 \in \mathcal{S}$ *et* $S_2 \in \mathcal{S}$ *tels que* $S_1 = S_2$ **faire**
  Supprimer S_1 de $\mathcal{S}$
tant que $t > 0$ *et que* $\mathcal{S} \cup \mathcal{L} \neq \emptyset$ **faire**
  Supprimer un ensemble $S \in \mathcal{S} \cup \mathcal{L}$
  $\mathcal{S}' \leftarrow S \cup \mathcal{S}'$; $t \leftarrow t - 1$
si $t \leq 0$ **alors**
  retourner $\mathcal{S}'$
sinon
  retourner $2^{\mathbb{N}}$

Théorème 7.1. *Soit deux collections d'ensembles (éventuellement vides)* $\mathcal{S}$ *et* $\mathcal{L}$ *tel que pour tout ensemble* $S \in \mathcal{S}$ *on a* $|S| = 2$ *et tel que tout élément de* $\cup_{S \in \mathcal{L}} S$ *appartient à un unique ensemble de* $\mathcal{S} \cup \mathcal{L}$*. Soit un entier positif* t*. La fonction* ***poly-CouverturePartielle*** *calcule en temps polynomial, si elle existe, une collection* $\mathcal{S}' \subseteq (\mathcal{S} \cup \mathcal{L})$ *de plus petite taille possible telle que* $\mathcal{U}(\mathcal{S}') \geq t$*. Si une telle collection n'existe pas, la fonction* ***poly-CouverturePartielle*** *retourne la collection infinie* $2^{\mathbb{N}}$*.*

Démonstration. Le but de la fonction **poly-CouverturePartielle** est de sélectionner le moins d'ensembles possibles de $\mathcal{S} \cup \mathcal{L}$ de sorte à couvrir au moins t éléments de $\mathcal{U}(\mathcal{S} \cup \mathcal{L})$. Soit $\mathcal{S}'$ une solution optimale. Supposons que $\mathcal{S}'$ contienne un ensemble S_1 de cardinalité deux et qu'il existe un ensemble de $\mathcal{L} \setminus \mathcal{S}'$, noté S_2, dont la cardinalité est au moins deux. Alors la solution $(\mathcal{S}' \setminus \{S_1\}) \cup \{S_2\}$ est également une solution optimale. Par conséquent puisque pour tout ensemble $S \in \mathcal{S}$ on a $|S| = 2$, il est préférable de choisir tout d'abord

des ensembles de $\mathcal{L}$ de cardinalité au moins 2. On ajoute donc à la solution l'ensemble de $\mathcal{L}$ de cardinalité maximum, et qui au moins de cardinalité 2, à la solution.

Si l'on n'a pas couvert suffisament d'élément, c'est-à-dire si $t > 0$ à ce moment de l'exécution, alors pour tout $S \in \mathcal{L}$, $|S| = 1$. Comme la collection $\mathcal{S}$ n'a pas été modifiée depuis l'appel à **poly-CouverturePartielle**, l'algorithme choisit des ensembles de $\mathcal{S}$ ayant comme cardinalité au moins 2 pour couvrir des éléments. Pour déterminer la plus grande collection disjointe $\mathcal{C}$ de sous-ensembles de $\mathcal{S}$, on cherche un couplage de taille maximum dans un graphe $G = (V, E)$ défini par $V = \mathcal{U}(\mathcal{S})$ et par $E = \{\{u, v\}$ tel que $u, v \in V$ et $\{u, v\} \in \mathcal{S}\}$. Nous avons emprunté cette idée à l'algorithme de Fomin *et al.* [FGK05a].

A l'issu de l'ajout des ensembles de $\mathcal{C}$ ayant chacun cardinalité deux, il ne reste dans $\mathcal{S}$ et dans $\mathcal{L}$ que des ensembles ne contenant qu'un unique élément, qui lui même n'apparaît que dans un seul ensemble de $\mathcal{S} \cup \mathcal{L}$. On ajoute donc ces ensembles au fur et à mesure, jusqu'à satisfaire $t \leq 0$.

Finalement, la fonction nécessite un temps polynomial. En particulier le calcul du couplage peut être réaliser avec un algorithme classique s'exécutant en temps polynomial. En effet, pour un graphe $G = (V, E)$, l'algorithme de Micali et Vazirani s'exécute en temps $\mathcal{O}(\sqrt{|V|}|E|)$ [MV80, PL88]. Il est donc possible de calculer le couplage en temps $\mathcal{O}(\sqrt{|\mathcal{U}(\mathcal{S})|}|\mathcal{S}|)$. De plus, on sait que tous les ensembles de $\mathcal{S}$ ont une cardinalité égale à 2, d'où un temps borné par $\mathcal{O}(|\mathcal{S}|^{1.5})$. □

7.1.2 Analyse du temps d'exécution de l'algorithme

On étudie dans cette section la complexité en temps de l'algorithme **CouverturePartielle** pour montrer que son temps d'exécution au pire des cas est borné par $\mathcal{O}(1.2721^{|\mathcal{U}|+|\mathcal{S}|})$. Une conséquence immédiate est la possibilité de résoudre le problème Domination Partielle en temps $\mathcal{O}(1.6183^n)$. L'algorithme **CouverturePartielle** est un algorithme typique *Brancher & Réduire*. Pour établir une borne supérieure sur le temps d'exécution, il nous faut établir les récurrences correspondant aux différentes règles de l'algorithme. Tout comme nous l'avons fait au chapitre 5 pour étudier la complexité de l'algorithme **mdc** calculant une clique dominante de taille minimum dans un graphe, nous utilisons également ici une fonction de mesure non standard sur la taille de l'instance dont nous motiverons le choix au paragraphe suivant. L'analyse que nous proposerons ensuite sera reliée à celle donnée par Fomin *et al.* dans [FGK05a].

Lorsque l'algorithme branche sur un ensemble, sa cardinalité revêt une certaine importance. En effet, si l'ensemble contient beaucoup d'éléments alors sa sélection dans la solution permet de diminuer la valeur de t de façon importante. De plus, la suppression d'éléments apparaissant dans beaucoup d'ensembles, on parle alors de la fréquence d'un élément, permet de réduire la cardinalité de beaucoup d'ensembles. Rappelons que si tous les ensembles ont une cardinalité d'au plus deux alors nous sommes capables de résoudre le problème en temps polynomial. Aussi, si en réduisant la taille d'un ensemble il apparaît qu'un ensemble est contenu dans un autre alors il est possible, par application de l'une des

règles de réduction, de supprimer cet ensemble. Ainsi, la suppression d'éléments de grande fréquence semble être plus avantageuse que la suppression d'elements de fréquence faible.

Pour tenir compte de ces disparités entre les rôles que jouent les ensembles et les éléments en fonction de leur cardinalité ou fréquence, nous utilisons une fonction de mesure non standard sur la taille de l'entrée. Soit n_i le nombre d'ensembles de $\mathcal{S}$ de cardinalité i, et soit m_j le nombre d'eléments de $\mathcal{U}(\mathcal{S})$ de fréquence j. Nous considérons la mesure $\mu(\mathcal{S})$ définie par

$$\mu(\mathcal{S}) = \sum_{i \geq 1} w_i n_i + \sum_{j \geq 1} v_j m_j$$

où w_i et v_j sont des poids appartenant à l'intervalle $[0, 1]$.

C'est déjà l'utilisation d'une mesure non standard qui nous avait permis d'établir le temps d'exécution de notre algorithme calculant une clique dominante à la section 5.4.2. Là encore, l'utilisation d'une mesure non standard va nous permettre d'analyser plus finement la complexité de notre algorithme.

Dans la suite on utilise les quantités

$$\Delta w_i = \begin{cases} w_i - w_{i-1} & \text{si } i \geq 3 \\ w_2 & \text{si } i = 2 \end{cases}$$

et

$$\Delta v_i = \begin{cases} v_i - v_{i-1} & \text{si } i \geq 2 \\ w_1 & \text{si } i = 1 \end{cases}$$

La quantité Δw_i représente le gain sur la mesure μ lorsqu'un ensemble de cardinalité i voit sa cardinalité diminuer à $i - 1$. Si la cardinalité d'un ensemble est égale à 2 et qu'un élément de cet ensemble est supprimé, on obtient un ensemble de cardinalité 1. Or si S est un ensemble de cardinalité 1, soit S contient un unique élément de fréquence 1, auquel cas l'ensemble S est ajouté à $\mathcal{L}$ et est supprimé de $\mathcal{S}$; soit l'ensemble S est inclu dans un autre ensemble de $\mathcal{S}$ et dans ce cas aussi l'une des règles de réduction supprime S de $\mathcal{S}$. Par conséquent, supprimer un élément d'un ensemble de cardinalité 2 implique la suppression de l'ensemble de $\mathcal{S}$.

Concernant la quantité Δv_i, elle représente le gain sur la mesure μ lorsqu'un élément voit sa fréquence diminuer de i à $i - 1$.

Notons $T(\mu)$ le nombre de sous-problèmes récursivement résolus pour résoudre un problème de taille μ. Tout d'abord, on note que l'application des règles de réduction n'augmente pas la taille de la mesure. Regardons la règle de branchement. Soit S un ensemble de cardinatité au moins 3 choisi par l'algorithme. Pour déterminer une solution optimale de l'instance courante, deux sous-problèmes sont récursivement résolus : celui correspondant au calcul de $\mathcal{S}_{\text{IN}}$ et celui correspondant au calcul de $\mathcal{S}_{\text{OUT}}$. Le problème $\mathcal{S}_{\text{IN}}$ ajoute l'ensemble S à la solution alors que le problème $\mathcal{S}_{\text{OUT}}$ le rejette de l'instance.

Pour simplifier l'analyse et le calcul des récurrences, nous imposons dans la suite les restrictions suivantes :

- $w_1 = 0$, puisque tous les singletons sont traités par une règle de réduction ;
- $w_i = 1$ pour tout $i \geq 6$;
- $v_j = 1$ pour tout $j \geq 6$;
- $0 \leq \Delta w_i \leq \Delta w_{i-1}$ pour tout $i \geq 2$.

Considérons le sous-problème $\mathcal{S}_{\text{IN}}$ et étudions la taille du sous-problème à résoudre lors de l'appel récursif à **CouverturePartielle**. Par rapport à l'instance courante, la suppression de S implique une diminution de $w_{|S|}$. L'ajout de S à la solution entraîne que tous les éléments de S sont maintenant couverts et sont supprimés de $\mathcal{U}(\mathcal{S})$ par l'entremise de l'opération `del`$(\mathcal{S}, S)$. Si on note r_i le nombre d'éléments de S de fréquence i alors la suppression des éléments de S implique une diminution de

$$\sum_{i \geq 1} r_i v_i = \sum_{1 \leq i \leq 6} r_i v_i + r_{\geq 7}$$

puisque $v_i = 1$ pour tout $i \geq 7$, avec $r_{\geq 7}$ représentant le nombre d'éléments de fréquence au moins 7 dans S.

Regardons maintenant un ensemble R ayant une intersection non vide avec S. De par la règle de choix de S, on a $|R| \leq |S|$. Lors de l'application de `del`$(\mathcal{S}, S)$, la diminution de la cardinalité de R fait décroître la mesure d'au moins $\Delta w_{|R|} \geq \Delta w_{|S|}$. Ainsi, si on regarde tous les ensembles qui partagent des éléments avec S, la diminution de leur cardinalité contribue à faire décroître la mesure par au moins

$$\Delta w_{|S|} \sum_{i \geq 1} (i-1) r_i = \Delta w_{|S|} (\sum_{2 \leq i \leq 6} (i-1) r_i + 6 r_{\geq 7}).$$

D'après les restrictions que nous avons posées sur les poids, cette quantité est égale à 0 pour $|S| \geq 7$.

Considérons à présent le problème $\mathcal{S}_{\text{OUT}}$. De par la suppression de l'ensemble S de la collection $\mathcal{S}$, la taille du sous-problème à résoudre est diminuée de $w_{|S|}$ par rapport à la taille du problème courant. De plus, la suppression de S implique une diminution de la fréquence des éléments de S. Là encore, notons r_i le nombre d'éléments de S de fréquence i. La mesure décroît alors d'au moins

$$\sum_{i \geq 1} r_i \Delta v_i = \sum_{1 \leq i \leq 6} r_i \Delta v_i$$

puisque $\Delta v_i = 0$ pour tout $i \geq 7$.

En résumé nous obtenons la récurrence suivante :

$$T(\mu) = T(\mu - \Delta_{\text{IN}}) + T(\mu - \Delta_{\text{OUT}})$$

avec

$$\Delta_{\text{IN}} = w_{|S|} + \sum_{1\leq i\leq 6} r_i v_i + r_{\geq 7} + \Delta w_{|S|}(\sum_{2\leq i\leq 6}(i-1)r_i + 6r_{\geq 7})$$

$$\Delta_{\text{OUT}} = w_{|S|} + \sum_{1\leq i\leq 6} r_i \Delta v_i.$$

Pour résoudre ce système de récurrences, il suffit de considérer toutes les valeurs possibles de $|S|$, avec $3 \leq |S| \leq 7$, et des r_i tels que $\sum_{1\leq i\leq 6} r_i + r_{\geq 7} = |S|$.

La résolution d'un tel système se fait de façon similaire à la technique exposée à la section 5.4.2. En considérant comme valeurs de poids $w_2 = 0.4353$, $w_3 = 0.8555$, $w_4 = 0.9681$, $w_5 = 0.9921$, $v_1 = 0.6772$, $v_2 = 0.9161$, $v_3 = 0.9766$, $v_4 = 0.9930$, $v_5 = 0.9960$, on établit que $T(\mu) \leq \alpha^{\mu} \leq \alpha^{|\mathcal{U}|+|\mathcal{S}|}$ avec $\alpha < 1.2721$. Ceci nous permet d'établir le théorème suivant :

Théorème 7.2. *Le problème* t-Couverture d'Ensembles *peut être résolu en temps* $\mathcal{O}(1.2721^{|\mathcal{U}|+|\mathcal{S}|})$ *et espace polynomial.*

À cause de la réduction du problème t-Domination au problème t-Couverture d'Ensembles donnée au début de la section 7.1.1, nous obtenons le corollaire qui suit.

Corollaire 7.3. *Le problème* t-Domination *peut être résolu en temps* $\mathcal{O}(1.2721^{2n}) = \mathcal{O}(1.6183^{n})$ *et espace polynomial.*

7.1.3 Le problème de la domination romaine

On s'intéresse dans cette section à une autre variante du problème Domination : la *domination romaine*. Cette variante est motivée par un problème paraît-il historique. L'histoire veut qu'il y ait un décret de l'empereur romain Constantin le Grand datant du 4e siècle instaurant de nouvelles règles pour la protection de l'empire romain. L'empereur souhaite que chaque province soit protégée par une légion romaine présente sur son territoire ou qu'à défaut, il y ait deux légions positionnées sur au moins une des provinces voisines (voir la figure 7.1). Formellement, le problème de décision est défini de la façon suivante :

Domination Romaine

entrée : Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.

question : Existe-t-il $V_1 \subseteq V$ et $V_2 \subseteq V$ tels que

1. chaque sommet $v \in V \setminus (V_1 \cup V_2)$ a au moins un voisin dans V_2 ; et
2. $|V_1| + 2|V_2| \leq k$?

Le problème demande de déterminer pour un graphe $G = (V, E)$ une partition des sommets de G en trois sous-ensembles V_0, V_1 et V_2 tels que tout sommet de V_0 ait au moins un voisin dans V_2. Le coût induit par une telle partition est donné par la valeur de $|V_1| +$

$2|V_2|$. L'objectif du problème d'optimisation est de déterminer une partition qui minimise ce coût. Le coût minimum sur toutes les solutions possibles est appelé *nombre dominant romain* du graphe G, noté $\gamma_R(G)$, et son calcul est NP-difficile [CDHH04]. Du point de vue de la complexité à paramètre fixé, le problème a été montré $W[2]$-complet [Lie04]. Concernant l'approximation du problème, il a été montré dans [PPSSTW02] qu'il existe un schéma d'approximation polynomial (PTAS) pour résoudre DOMINATION ROMAINE sur les graphes planaires, et, pour un graphe quelconque, il existe un algorithme d'approximation de rapport $2 + 2\ln|V|$.

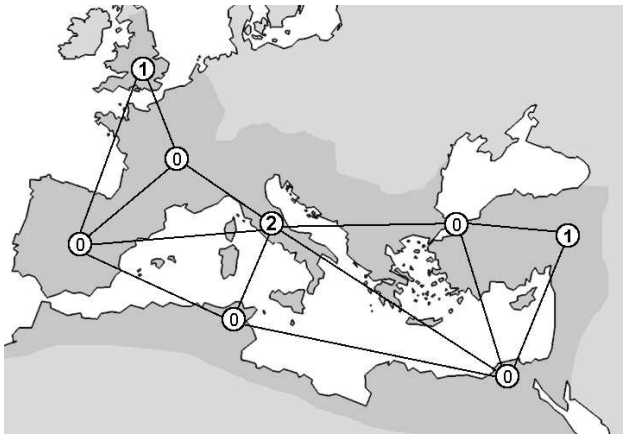


Fig. 7.1 – L'empire romain aux environs de l'an 300. Chaque province est représentée par un sommet et deux provinces sont voisines lorsqu'une arête les relie. Une solution optimale au problème DOMINATION ROMAINE, indiquant l'appartenance d'un sommet à V_0, V_1 ou V_2, est donnée par la figure.

En utilisant une approche similaire à celle de Fomin, Kratsch et Woeginger [FKW04] combinée avec le résultat de McCuaig et Shepherd rappelé ci-après, il est possible d'établir un algorithme en $\mathcal{O}(1.9602^n)$.

Théorème 7.4 ([MS89] ; voir aussi [Bla73])**.** *Si un graphe connexe $G = (V, E)$ a pour degré minimum au moins deux et si G n'est pas l'un des sept graphes exceptionnels (voir la figure 7.2) alors la taille d'un ensemble dominant minimum de G est au plus $2|V|/5$.*

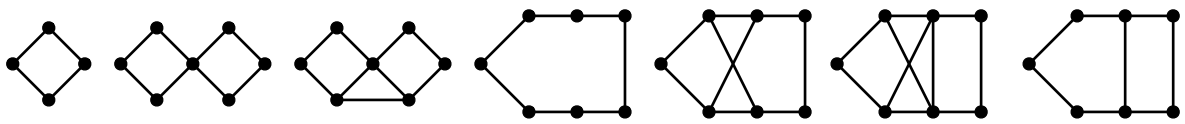


Fig. 7.2 – La collection des sept graphes exceptionnels du théorème 7.4.

L'idée de cet algorithme simple, dont une description détaillée est donnée dans [Lie04], est de traiter tous les sommets du graphe de degré 0, 1. Les sommets de degré 0 sont placés dans V_1 alors que les sommets de degré 1 nécessitent un branchement : un sommet

de degré 1 appartient soit à V_1, soit son unique voisin est dans V_2. Puis l'algorithme énumère tous les sous-ensembles de sommets de taille au plus $2|V|/5$. C'est cette dernière étape d'énumération qui réclame le plus de temps et qui implique la borne supérieure de $\mathcal{O}(1.9602^n)$ sur le temps d'exécution au pire des cas de cet algorithme. On obtient ainsi le premier algorithme plus rapide que l'algorithme naïf en $\mathcal{O}^*(2^n)$ énumérant tous les sous-ensembles de sommets.

Il est possible d'améliorer de façon significative ce résultat pour résoudre un problème plus général que nous donnons ci-après. On note immédiatement que le problème $(1,2)$-Domination Romaine est précisément le problème Domination Romaine.

(v_1, v_2)-Domination Romaine

entrée : Un graphe $G = (V, E)$, deux réels positifs v_1 et v_2, et un entier $k \in \mathbb{N}$.

question : Existe-t-il $V_1 \subseteq V$ et $V_2 \subseteq V$ tels que

1. chaque sommet $v \in V \setminus (V_1 \cup V_2)$ a au moins un voisin dans V_2 ; et
2. $v_1 \cdot |V_1| + v_2 \cdot |V_2| \leq k$?

Pour résoudre ce problème, nous allons nous servir de l'algorithme pour t-Domination établi précédemment dans ce chapitre à la section 7.1. Nous obtenons le théorème suivant et montrons dans sa preuve les liens entre ces deux problèmes.

Théorème 7.5. *Le problème* (v_1, v_2)-Domination Romaine *peut être résolu en temps* $\mathcal{O}(1.6183^n)$ *et espace polynomial.*

Démonstration. Soit $G = (V, E)$ un graphe pour lequel on cherche une solution de coût minimum au problème (v_1, v_2)-Domination Romaine. C'est-à-dire que l'on cherche deux ensembles V_1 et V_2 qui solutionnent le problème et qui minimisent la valeur $v_1 \cdot |V_1| + v_2 \cdot |V_2|$.

Considérons l'algorithme permettant de résoudre t-Domination en temps $\mathcal{O}(1.6183^n)$ pour un graphe à n sommets. Nous dénotons un appel à cet algorithme par **Domination-Partielle**(t,G). L'algorithme retourne alors un sous-ensemble des sommets du graphe G tel que au moins t sommets du graphe soient dominés.

Regardons l'algorithme **DominationRomaine**. On désigne par D_t, $0 \leq t \leq n$, les solutions optimales retournées par l'algorithme **DominationPartielle** pour résoudre le problème t-Domination sur le graphe G. Notons que pour tout t, $0 \leq t \leq n$, on a $|D_t| \leq |D_{t+1}| \leq |D_t| + 1$. En effet, si $|D_t| > |D_{t+1}|$ alors l'ensemble D_{t+1} voudrait être une solution optimale pour t-Domination de plus petite cardinalité que D_t ; une contradiction. Si $|D_{t+1}| > |D_t| + 1$ alors l'ensemble $D_t \cup v$ où v est un sommet de $V \setminus N[D_t]$ est une solution pour $(t+1)$-Domination de cardinalité $|D_t| + 1$; une contradiction sur l'optimalité de D_{t+1}.

Algorithme DominationRomaine(v_1, v_2, $G = (V, E)$)
Entrées: Deux réels v_1 et v_2, et un graphe $G = (V, E)$.
Sortie : Une solution de coût minimum au problème (v_1, v_2)-Domination Romaine pour le graphe G.

$V_1^{\text{Opt}} \leftarrow V$
$V_2^{\text{Opt}} \leftarrow \emptyset$
pour t *de* 0 *à* $|V|$ **faire**
  $D \leftarrow$ **DominationPartielle**(t,G)
  si $v_1|V \setminus N[D]| + v_2|D| \leq v_1|V_1^{Opt}| + v_2|V_2^{Opt}|$ **alors**
    $V_1^{\text{Opt}} \leftarrow V \setminus N[D]$
    $V_2^{\text{Opt}} \leftarrow D$
retourner $(V_0 = V \setminus (V_1^{\text{Opt}} \cup V_2^{\text{Opt}})\,;\, V_1 = V_1^{\text{Opt}}\,;\, V_2 = V_2^{\text{Opt}})$

Nous rappelons aussi que D_n est, par définition du problème n-Domination, un ensemble dominant minimum du graphe G. Étant donné une solution (V_0, V_1, V_2) de coût minimum au problème (v_1, v_2)-Domination Romaine, nous établissons comme borne supérieure sur la cardinalité de V_2, $|V_2| \leq |D_n|$. En effet, pour tout ensemble V_2 tel que $|V_2| > |D_n|$, le coût de la solution satisfait la relation $v_1|V \setminus N[V_2]| + v_2|V_2| > v_2|D_n| = v_1|V \setminus N[D_n]| + v_2|D_n|$. Cette dernière expression correspond précisément au coût de la solution $(V_0 = V \setminus D, V_1 = \emptyset, V_2 = D_n)$.

Soit un entier t, $0 \leq t \leq n$. Nous notons $t_{\max}$ (respectivement par $t_{\min}$) le plus grand (respectivement le plus petit) indice t' tel que $|D_{t'}| = |D_t|$. Nous allons établir le lemme important suivant :

Lemme 7.6. *L'ensemble $D_{t_{\max}}$ est une solution optimale au problème t'-Domination, si et seulement si t' vérifie $t_{\min} \leq t' \leq t_{\max}$.*

Démonstration. Par définition de $t_{\min}$ et $t_{\max}$ et du fait que la fonction qui à t associe $|D_t|$ est croissante sur $[0, n]$ (car $|D_t| \leq |D_{t+1}|$ pour tout $t \in [0, n]$), nous avons $|D_{t_{\max}}| = |D_{t'}|$ pour tout t', $t_{\min} \leq t' \leq t_{\max}$. Donc $D_{t_{\max}}$ est bien une solution optimale au problème t'-Domination pour t' tel que $t_{\min} \leq t' \leq t_{\max}$.
Supposons que $D_{t_{\max}}$ soit une solution optimale au problème $(t_{\max} + 1)$-Domination. À cause du choix de l'indice $t_{\max}$, on a $|D_{t_{\max}}| < |D_{t_{\max}+1}|$. Ainsi $D_{t_{\max}}$ ne peut être une solution optimale de $(t_{\max} + 1)$-Domination sans contredire l'optimalité de l'ensemble $D_{t_{\max}+1}$. Comme pour tout $t' \geq t_{\max} + 1$ on a $|D_{t'}| \geq |D_{t_{\max}+1}|$, on en déduit que $D_{t_{\max}}$ n'est pas une solution optimale des problèmes t'-Domination pour t' tel que $t' > t_{\max}$.
De même, si nous supposons que $D_{t_{\max}}$ est une solution optimale au problème $(t_{\max} - 1)$-Domination, nous aboutissons à une contradiction puisque pour tout $t' < t_{\min}$, on a $|D_{t'}| < |D_{t_{\max}}|$. □

Nous montrons ensuite le lemme suivant :

Lemme 7.7. *L'ensemble $D_{t_{\max}}$ est une solution optimale au problème qui demande de déterminer un ensemble de taille $|D_t| = |D_{t_{\max}}| = |D_{t_{\min}}|$ qui domine le plus de sommets possibles.*

Démonstration. Supposons qu'il existe un ensemble S tel que $|S| = |D_t|$ et $|N[S]| > |N[D_{t_{\max}}]|$. Alors cet ensemble est solution au problème $(t_{\max}+1)$-DOMINATION et sa cardinalité est plus petite que celle de $D_{t_{\max}+1}$; une contradiction sur l'optimalité de l'ensemble $D_{t_{\max}+1}$. □

D'après la définition du problème (v_1, v_2)-DOMINATION ROMAINE, connaissant l'ensemble V_2 d'une solution optimale nous pouvons en déduire la partition correspondante (V_0, V_1, V_2) : l'ensemble V_0 est l'ensemble des sommets dominés par V_2 et n'appartenant pas à V_2, c'est-à-dire $V_0 = N[V_2] \setminus V_0$, et l'ensemble V_1 est l'ensemble des sommets non dominés par V_2, c.-à-d. $V_1 = V \setminus V_2$.

Comme nous avons montré que dans une solution optimale (V_0, V_1, V_2), l'ensemble V_2 satisfait toujours $|V_2| \leq |D_n| \leq n$, pour déterminer une solution de coût minimum au problème (v_1, v_2)-DOMINATION ROMAINE, il est suffisant de regarder comme candidats pour une solution optimale les ensembles V_2 de cardinalité $|D_0| = 0$ à cardinalité $|D_n|$ qui maximisent $|N[V_2]|$. Comme le montrent les propositions précédentes, ces ensembles appartiennent à l'ensemble $\{D_t \ : \ 0 \leq t \leq n\}$. Ceci prouve la validité de l'algorithme **DominationRomaine** pour déterminer une solution de coût minimum au problème (v_1, v_2)-DOMINATION ROMAINE. Puisque l'algorithme **DominationRomaine** fait n appels à la fonction **DominationPartielle** et que le temps d'exécution de cette fonction est bornée par $\mathcal{O}(1.6183^n)$, on en déduit que le temps d'exécution de **DominationRomaine** est également borné asymptotiquement au pire des cas par cette même borne. Ceci montre le théorème 7.5. □

Comme conséquence du théorème 7.5, on obtient le corollaire suivant qui nous permet d'obtenir le meilleur algorithme exponentiel connu pour résoudre DOMINATION ROMAINE.

Corollaire 7.8. *Le problème* DOMINATION ROMAINE *peut être résolu en temps $\mathcal{O}(1.6183^n)$ et espace polynomial.*

7.1.4 Le problème k-centre partiel

Dans cette section, on étudie une généralisation du problème k-CENTRE du point de vue des algorithmes exponentiels. Avant de rappeler la définition du problème k-CENTRE, nous introduisons quelques notations. Étant donné un graphe $G = (V, E)$, une fonction de pondération $w : E \to \mathbb{R}$, un sous-ensemble $S \subseteq V$ et un sommet $v \in V$, on définit la distance d'un sommet v à l'ensemble S, noté par $w(v, S)$, de la façon suivante :

$$w(v,S) = \begin{cases} +\infty & \text{si } v \notin N[S]; \\ 0 & \text{si } v \in S; \\ \min\limits_{u \in S} w(\{u,v\}) & \text{si } v \in N[S] \setminus S. \end{cases}$$

Le problème de décision k-CENTRE est alors défini ci-après. Nous supposons que l'entier k fait partie de l'entrée du problème, contrairement à la définition habituelle de ce problème.

k-CENTRE

entrée : Un graphe $G = (V, E)$, une fonction de pondération w sur les arêtes, un entier $k \in \mathbb{N}$ et un entier $l \in \mathbb{N}$.

question : Existe-t-il un k-centre dans G, c'est-à-dire un sous-ensemble de sommets $S \subseteq V$ tel que $|S| \leq k$ et de sorte que $\max_{v \in V-S} w(v, S) \leq l$?

On désigne par $c(G)$ la plus petite valeur de $\max_{v \in V-S} w(v, S)$ parmi tous les k-centres S d'un graphe $G = (V, E)$. Le problème d'optimisation k-CENTRE demande de déterminer un k-centre S de G pour lequel $\max_{v \in V-S} w(v, S) = c(G)$.

Le problème de décision k-CENTRE est connu d'être NP-complet et d'être approximable avec un facteur 2 [Gon85, HS86] par un algorithme glouton simple. D'un autre côté, il est improbable d'obtenir une approximation en $2 - \epsilon$ pour tout $\epsilon > 0$, sauf si P = NP [HN79, Ple80].

On s'intéresse à la généralisation de k-CENTRE, le problème k-CENTRE PARTIEL. Nous supposons là encore que l'entier k fait partie de l'entrée du problème.

k-CENTRE PARTIEL

entrée : Un graphe $G = (V, E)$, une fonction de pondération w sur les arêtes, un entier $k \in \mathbb{N}$, un entier $l \in \mathbb{N}$ et un entier t.

question : Existe-t-il un k-centre partiel dans G, c'est-à-dire un sous-ensemble de sommets $S \subseteq V$ tel que
(i) $|S| \leq k$; et
(ii) si on ordonne les sommets de $V = \{v_1, \ldots, v_n\}$ de sorte que $w(v_1, S) \leq w(v_2, S) \leq \cdots \leq w(v_n, S)$, alors $w(v_t, S) \leq l$.

Le problème d'optimisation k-CENTRE PARTIEL demande de trouver un k-centre partiel S qui minimise la distance $w(v_t, S)$. On note alors cette valeur optimale par $c_{t,k}(G)$. On remarque que pour $t = n$, le problème k-CENTRE PARTIEL est précisément le problème k-CENTRE.

Une application usuelle de k-CENTRE peut être la suivante. Une chaîne de supermarchés souhaite implanter k nouveaux magasins de sorte à minimiser la distance entre les clients et le magasin le plus proche. Il faut alors décider de leur lieu d'implantation.

Une application du problème k-CENTRE PARTIEL concerne la mise en service de technologies haut débit du genre ADSL (*Asymmetric Digital Subscriber Line*). Le débit maximum proposé à l'utilisateur dépend directement de sa distance au nœud de raccordement

d'abonnés (NRA) équipé de la technologie. Plus la distance augmente, plus le débit diminue jusqu'à une certaine distance à partir de laquelle le client ne peut se voir proposer un accès au réseau à cause d'un trop fort affaiblissement. Déterminer l'emplacement de k NRA avec ADSL, de sorte à minimiser la distance au client, tout en couvrant au moins $t \leq n$ clients est un problème directement modélisable par une instance de k-CENTRE PARTIEL.

Dans la suite nous étudions les relations entre le problème k-CENTRE PARTIEL et le problème t-DOMINATION donné à la section 7.1 (page 144). L'algorithme **Dom** donné sur cette page montre qu'on peut résoudre le problème t-DOMINATION en utilisant un algorithme qui résout le problème k-CENTRE PARTIEL. Réciproquement, l'algorithme **Centre** explique comment on peut résoudre k-CENTRE PARTIEL en se servant d'un algorithme pour résoudre t-DOMINATION.

Algorithme Dom($G = (V, E)$,t)
Entrée: Un graphe G, un entier t. Un algorithme qui résout k-CENTRE PARTIEL .
Sortie : Le nombre dominant $\gamma_t(G)$.

Soit $G' = (V', E')$ avec
- $V' \leftarrow V$
- $E' \leftarrow \{\{u, v\} : u \in V', v \in V', u \neq v\}$

Soit $w(\{u, v\}) = \begin{cases} 1 & \text{si } \{u, v\} \in E \\ +\infty & \text{sinon} \end{cases}$

$k \leftarrow 0$
tant que $k \leq n$ **faire**
  $k \leftarrow k + 1$
  Résoudre k-CENTRE PARTIEL sur l'entrée $(G' = (V', E'), w, t, k)$
  si *S est une solution telle que* $w(v_t, S) \leq 1$ **alors**
    retourner « $\gamma_t(G) = k$ »

Algorithme Centre($G = (V, E)$, w, t, k)
Entrées: Un graphe $G = (V, E)$, une fonction de pondération $w : E \to \mathbb{R}$, deux entiers $t, k \in \mathbb{N}$. Un algorithme qui résout t-DOMINATION.
Sortie : La valeur optimale $c_{t,k}(G)$.

Trier les arêtes de E de sorte que $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$
$i \leftarrow 0$
tant que $i \leq m$ **faire**
  $i \leftarrow i + 1$
  Soit $G_i = (V, E_i)$ où $E_i = \{e_j \in E : j \leq i\}$
  Résoudre t-DOMINATION sur l'entrée $(G_i = (V, E_i), t)$
  si *S est une solution telle que* $|S| \leq k$ **alors**
    retourner « $c_{t,k}(G) = w(e_i)$ »

Lemme 7.9. *Soit un graphe $G = (V, E)$. Soit $G' = (V', E')$ le graphe défini par :*
- $V' = V$;
- $E' = \{\{u, v\} : u \in V', v \in V', u \neq v\}$.

Soit une fonction de pondération w telle que

$$w(\{u, v\}) = \begin{cases} 1 & si\ \{u, v\} \in E \\ +\infty & sinon. \end{cases}$$

Soit k, $0 \leq k \leq n$, le plus petit entier tel que $c_{t,k}(G') \leq 1$. Alors $\gamma_t(G) = k$.

Démonstration. Soit k le plus petit entier pour lequel $c_{t,k}(G') \leq 1$.

Montrons que $(c_{t,k}(G') \leq 1) \Rightarrow (\gamma_t(G) \leq k)$. Supposons que $S \subseteq V'$ soit un ensemble de taille k. On ordonne les sommets de G' de sorte que $V' = \{v_1, \ldots, v_n\}$ avec $w(v_1, S) \leq w(v_2, S) \leq \cdots \leq w(v_n, S)$. Ainsi, $w(v_t, S) \leq 1$. Soit v_j un sommet de V' avec $j \leq t$. Soit v_j appartient à S, auquel cas $w(v_j, S) = 0$, soit v_j a un voisin v dans S tel que $w(v_j, v) = 1$. On en déduit donc, par la construction de G', que S domine au moins t sommets de G. Ainsi, $\gamma_t(G) \leq |S| = k$.

Supposons maintenant qu'il existe un entier k', strictement inférieur à k, tel que $\gamma_t(G) = k'$. Montrons que $(\gamma_t(G) = k') \Rightarrow (c_{t,k'}(G') \leq 1)$. Supposons que G possède un ensemble S de taille k' qui domine au moins t sommets. Ordonnons les sommets de G' de sorte que $V' = \{v_1, \ldots, v_n\}$ avec $w(v_1, S) \leq w(v_2, S) \leq \cdots \leq w(v_n, S)$. Puisque S domine au moins t sommets de G, il s'ensuit que S domine également au moins t sommets dans G'. Pour tout sommet v_j dominé par S dans G', on a $w(v_j, S) \leq 1$ puisque soit v_j appartient à S, soit il existe une arête $\{v_j, v\} \in E$ avec $v \in S$ et telle que $w(v_j, v) = 1$. Par conséquent, S est une solution au problème k'-CENTRE PARTIEL pour le graphe $G' = (V', E')$, la fonction de pondération w et tel que $w(v_t, S) \leq 1$. Comme $c_{t,k'}(G') \leq w(v_t, S)$, on en déduit que $c_{t,k'}(G') \leq 1$.

En conséquence, puisque k est le plus petit entier pour lequel $c_{t,k}(G') \leq 1$, l'existence d'un entier $k' < k$ tel que $\gamma_t(G) = k'$ est impossible sous peine de contradiction. □

Lemme 7.10. *Soit un graphe $G = (V, E)$, une fonction de pondération $w : E \to \mathbb{R}$ et deux entiers $t, k \in \mathbb{N}$. Supposons que les arêtes de $E = e_1, e_2, \ldots, e_m$ soient ordonnées de sorte que $w(e_1) \leq w(e_2) \leq \cdots \leq w(e_m)$. Soit $G_i = (V_i, E_i)$ le graphe défini pour tout i, $1 \leq i \leq m$, par :*
- $V_i = V$;
- $E_i = \{e_j \in E : j \leq i\}$.

Soit i, $0 \leq i \leq m$, le plus petit entier tel que $\gamma_t(G_i) \leq k$. Alors $c_{t,k}(G) = w(e_i)$.

Démonstration. Soit i le plus petit entier pour lequel $\gamma_t(G_i) \leq k$.

Montrons que $\gamma_t(G_i) \leq k \Rightarrow c_{t,k}(G) \leq w(e_i)$. Soit $S \subseteq V_i$ tel que S domine au moins t sommets dans G_i et tel que $|S| \leq k$. On ordonne les sommets de G de sorte que $V = \{v_1, \ldots, v_n\}$ avec $w(v_1, S) \leq w(v_2, S) \leq \cdots \leq w(v_n, S)$. Puisque $V_i = V$ et que $E_i \subseteq E$, on

a $|N_G[S]| \geq t$. Pour tout sommet v_j dominé par S, soit $v_j \in S$ et alors $w(v_j, S) = 0$, soit $v_j \in N(S)$ et dans ce cas $w(v_j, S) \leq w(e_i)$ puisque $w(e_i)$ est l'arête de plus grand poids présente dans E_i. Ainsi, on en déduit que $c_{t,k}(G) \leq w(e_i)$.

Supposons maintenant qu'il existe un entier i' tel que $c_{t,k}(G) = w(e_{i'}) < w(e_i)$. Comme les arêtes sont ordonnées par poids croissant, ceci implique que $i' < i$. Montrons que $c_{t,k}(G) = w(e_{i'}) \Rightarrow \gamma_t(G_{i'}) \leq k$. Soit $S \subseteq V$ une solution à k-CENTRE PARTIEL pour G et tel que $|S| = k$. Les sommets de G sont ordonnés de façon à ce que $V = \{v_1, \ldots, v_n\}$ avec $w(v_1, S) \leq w(v_2, S) \leq \cdots \leq w(v_n, S)$. Ainsi, $w(v_t, S) \leq w(e_{i'})$. Soit v_j un sommet de $V_{i'}$ avec $j \leq t$. Soit v_j appartient à S, soit v_j a un voisin v dans S tel que $w(v_j, v) \leq w(e_{i'})$. On en déduit donc, par la construction de G_i qui contient toutes les arêtes de poids au plus $w(e_{i'})$, que S domine au moins t sommets de G_i. Ainsi, $\gamma_t(G) \leq |S| = k$.

En conséquence, puisque i est le plus petit entier pour lequel $\gamma_t(G_i) \leq k$, l'existence d'un entier $i' < i$ tel que $c_{t,k}(G) = w(e_{i'})$ est impossible sans aboutir à une contradiction. □

On établit le théorème suivant :

Théorème 7.11. *Si on dispose d'un algorithme qui résout en temps $\mathcal{O}^*(\alpha^n)$ le problème k-*CENTRE PARTIEL*, alors l'algorithme **Dom** résout t-*DOMINATION *en temps $\mathcal{O}^*(\alpha^n)$.*

*Si on dispose d'un algorithme qui résout en temps $\mathcal{O}^*(\beta^n)$ le problème t-*DOMINATION*, alors l'algorithme **Centre** résout k-*CENTRE PARTIEL *en temps $\mathcal{O}^*(\beta^n)$.*

Démonstration. Les preuves de validité des algorithmes **Dom** et **Centre** sont respectivement assurées par les lemmes 7.9 et 7.10. De plus, les algorithmes peuvent être facilement modifiés de sorte à retourner l'ensemble S pour lequel $\gamma_t(G)$ ou $c_{t,k}(G)$ plutôt que de simplement retourner les valeurs de $\gamma_t(G)$ et $c_{t,k}(G)$.

Regardons l'analyse du temps d'exécution de **Dom**. Par hypothèse, on a un algorithme qui résout k-CENTRE PARTIEL en temps $\mathcal{O}^*(\alpha^n)$ sur un graphe à n sommets. L'algorithme **Dom** invoque au plus n fois cet algorithme. Ainsi, le temps d'exécution de **Dom** est borné par $\mathcal{O}^*(\alpha^n)$.

Pour l'algorithme **Centre**, on a supposé l'existence d'un algorithme pour résoudre t-DOMINATION en temps $\mathcal{O}^*(\beta^n)$ sur un graphe à n sommets. L'algorithme **Centre** fait au plus $m \leq n^2$ appels à cet algorithme. Ainsi, le temps d'exécution de **Centre** est borné par $\mathcal{O}^*(\beta^n)$. □

Le théorème précédent nous permet de déduire le corollaire suivant :

Corollaire 7.12. *Il existe un algorithme qui résout k-*CENTRE PARTIEL *en temps $\mathcal{O}^*(c^n)$ si et seulement s'il existe un algorithme qui résoud t-*DOMINATION *en temps $\mathcal{O}^*(c^n)$, où c est une constante positive.*

Finalement, nous concluons cette section par le corollaire suivant :

Corollaire 7.13. *Il existe un algorithme qui résout k-*CENTRE PARTIEL *en temps $\mathcal{O}^*(1.6183^n)$.*

Démonstration. Ce corollaire est obtenu en combinant le théorème 7.11 et le corollaire 7.3 (page 153). □

7.2 Domination avec des puissances variables

Cette section s'intéresse à une autre généralisation du problème de la domination pour laquelle les sommets peuvent être affectés d'une puissance. La puissance affectée à un sommet indique jusqu'à quelle distance ce sommet peut dominer d'autres sommets. Étant donné un graphe $G = (V, E)$ et un sommet v de V, on note $N^p[v]$ l'ensemble des sommets situés à distance au plus p du sommet v, formellement $N^p[v]$ représente l'ensemble $\{u \in V \ : \ d(v, u) \leq p\}$. Par convention, on pose $N^0[v] = \{v\}$. Ainsi, si un sommet v du graphe se voit attribuer une puissance $p(v)$ alors tous les sommets de l'ensemble $N^{p(v)}[v]$ sont réputés être dominés. Dans le cas classique de la domination, la seule puissance autorisée est $p = 1$: en effet, un sommet de l'ensemble dominant ne peut dominer que ses voisins immédiats. Pour le problème Domination Romaine présenté à la section 7.1.3, les puissances autorisées appartiennent à l'ensemble $P = \{0, 1\}$. Nous avons vu que, de par la définition du problème Domination Romaine, les deux puissances de P n'ont pas le même coût. Il est très naturel de considérer des coûts différents en fonction de la puissance attribuée. Habituellement, ce type de problèmes sert à l'implantation d'émetteurs terrestres permettant de diffuser un signal. Un émetteur avec une grande puissance permet de diffuser dans un rayon plus important qu'un émetteur avec une faible puissance. D'un autre côté, un tel émetteur coûte aussi plus cher. Ces types de problèmes réels suggèrent donc de s'intéresser à une généralisation du problème de la domination appelée Domination avec des Puissances Variables. Il s'agit également d'une généralisation du problème connu sous le nom *Broadcast Domination*, problème qui impose dans la définition qui suit que $c(i) = i$ pour tout $i \in P$ [Erw04]. Enfin, cette généralisation a également fait l'objet de travaux dans [Sch06] où le problème est montré d'être NP-complet, même lorsque le graphe donné en entrée est planaire, biparti ou cordal et que l'ensemble des puissances est de taille bornée. Il y est également proposé un algorithme polynomial pour résoudre ce problème sur un graphe de largeur arborescente bornée lorsque l'ensemble des puissances est borné.

Nous utiliserons par la suite l'abréviation DomPuisVar pour désigner ce problème.

Domination avec des Puissances Variables (DomPuisVar)

entrée : Un graphe $G = (V, E)$, une liste de puissances $P \subseteq \mathbb{N}$, une fonction de coût $c : P \to \mathbb{R}^+$ et un réel $k \in \mathbb{N}$.

question : Existe-il un sous-ensemble de sommets $D \subseteq V$ et une fonction $p : D \to P$ qui à chaque sommet de D associe une puissance de P et tels que
(i) $\bigcup_{v \in D} N^{p(v)}[v] = V$, et
(ii) $\sum_{v \in D} c(p(v)) \leq k$?

Étant donné un sous-ensemble $D \subseteq V$, une liste de puissances P et une fonction de coût c associée à cet ensemble de puissance, on appelera *coût d'une solution* la valeur $\sum_{v \in D} c(p(v))$ notée $c(D, p)$ ou simplement $c(D)$ s'il n'y a pas d'ambiguïté sur la fonction

p considérée. Le problème d'optimisation correspondant à DomPuisVar demande de déterminer une solution, c'est-à-dire un ensemble D et une fonction p, de coût total $c(D, p)$ minimum.

Dans la suite de cette section, lorsque nous manipulerons la fonction de coût c, nous supposerons que toutes les opérations arithmétiques pourront être réalisées en temps constant.

Nous introduisons maintenant la définition suivante qui nous sera utile dans les prochaines sections.

Définition 7.2.1. Soit un graphe $G = (V, E)$, une liste de puissances P et un sous-ensemble $D \subseteq V$ pour lequel on dispose d'une fonction p qui associe à chaque sommet de D une puissance de P. Soit un sommet $v \in D$ et soit w un sommet situé à distance $d(v, w) \leq p(v)$ de v. On dit alors que w a une *puissance induite (par v)* de $p(v) - d(v, w)$.

7.2.1 Résultats de complexité

On constate que pour un ensemble de puissances P réduit au singleton $\{1\}$ tel que le coût est défini par $c(1) = c_1$, avec c_1 une constante strictement positive, nous retrouvons le problème habituel de la Domination. Ceci permet d'en déduire immédiatement que DomPuisVar est un problème NP-difficile.

Une observation similaire peut être faite si la fonction de coût c est décroissante, c'est-à-dire si $c(i) \geq c(j)$ pour tout $i, j \in P$ avec $i < j$. Supposons que P soit fini et posons $p_{\max} = \max_{p \in P} p$. Soit D et p définissant une solution optimale. Alors, pour tout sommet $v \in D$ tel que $p(v) \neq p_{\max}$, la solution qui consiste à attribuer à v la puissance $p_{\max}$ en remplacement de $p(v)$ est également une solution optimale puisque $c(p_{\max}) \leq c(p(v))$. Ainsi, si la fonction de coût est décroissante alors seule la puissance $p_{\max}$ nécessite d'être considérée, puisque toute autre puissance pourrait être substituée par la puissance la plus forte et de coût moindre. Là encore, le problème reste NP-difficile par réduction vers Domination en considérant $P = \{1\}$ et $c(1)$. Néanmoins, étant donné un graphe $G = (V, E)$, on note que le problème peut alors être résolu en temps exponentiel grâce aux algorithmes classiques développés pour le problème Domination en considérant le graphe $G^{p_{\max}} = (V, \{\{u, v\} \text{ tel que } u, v \in V \text{ et } d(u, v) \leq p_{\max}\})$.

D'autre part, si $P = \mathbb{N}^*$ et si $c(i) = i$ pour tout $i \in P$, Heggernes et Lokshtanov [HL06] ont récemment montré que le problème peut être résolu en temps $\mathcal{O}(n^6)$. Le résultat de Heggernes et Lokshtanov est important puisque depuis l'introduction du problème et les premiers travaux sur ce cas particulier du problème ([Erw04, HMMU04, DEHHH06]), il n'était pas connu si le problème était NP-difficile ou dans P. Les auteurs de [HL06] indiquent également que l'approche utilisée peut être étendue au cas où la fonction c respecte $c(i) + c(j) \geq c(i + j)$.

Suite aux travaux de Heggernes et Lokshtanov, on pourrait avoir l'intuition que si l'ensemble P des puissances est non borné alors le problème admet un algorithme polynomial pour le résoudre, alors que si P est borné, comme c'est le cas pour Domination ou Domination Romaine, le problème serait NP-complet. Malheureusement, nous allons montrer

dans la suite que cette « dichotomie » sur la difficulté du problème n'est pas vraie. Il semble plutôt que Heggernes et Lokshtanov ont réussi à obtenir un algorithme polynomial grâce aux propriétés de la fonction de coût.

Le théorème suivant montre que pour un certain choix de l'ensemble de puissance P, avec P non borné, le problème est NP-complet.

Théorème 7.14. *Si l'ensemble des puissances P est égal à l'ensemble $\mathbb{N}^*$ et si la fonction de coût satisfait $c(i) = 2^i$ pour tout $i \in P$ alors le problème* DomPuisVar *est NP-complet.*

Démonstration. Le problème Couverture de Sommets est NP-complet, même lorsqu'il est restreint aux graphes cubiques [GP95]. Nous allons montrer le théorème en utilisant une réduction depuis Couverture de Sommets restreint aux graphes cubiques.

Le problème DomPuisVar tel que défini dans l'énoncé du théorème est dans NP. Etant donné une fonction $p : D \to P$ qui à chaque sommet de D associe une puissance de P, on peut vérifier un temps polynomial que $\bigcup_{v \in D} N^{p(v)}[v] = V$ et calculer en temps polynomial la valeur du coût total $c(D, p)$.

Soit un graphe $G = (V, E)$ et un entier k définissant une instance de Couverture de Sommets. Nous allons construire un graphe $G' = (V', E')$ tel que G' admet une solution de coût au plus $k' = 4k$ pour DomPuisVar si et seulement si G possède une couverture de sommets de taille au plus k. De plus, on suppose que l'instance de DomPuisVar n'autorise que l'ensemble de puissances $P = \mathbb{N}^*$ avec pour coût $c(i) = 2^i$, pour tout $i \in P$.

Le graphe $G' = (V', E')$ est spécifié par (voir aussi les figures 7.3 et 7.4) :

– $V' = V \cup \left\{ s^1_{uv}, s^{\mathring{1}}_{uv}, s^2_{uv}, s^{\mathring{2}}_{uv} \ : \ \{u, v\} \in E \right\}$;

– $E' = \left\{ \{u, s^1_{uv}\}, \{u, s^2_{uv}\}, \{v, s^1_{uv}\}, \{v, s^2_{uv}\}, \{s^1_{uv}, s^{\mathring{1}}_{uv}\}, \{s^2_{uv}, s^{\mathring{2}}_{uv}\} \ : \ \{u, v\} \in E \right\}$.

On note que cette instance peut être construite en temps polynomial par rapport à la taille de l'entrée du problème Couverture de Sommets.

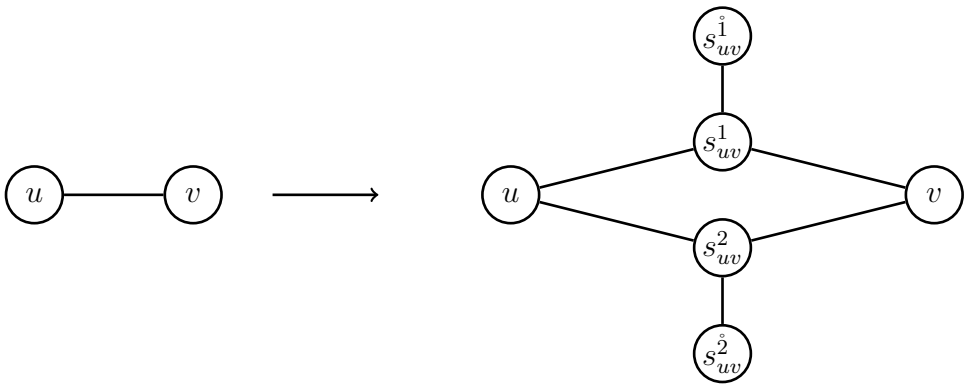


Fig. 7.3 – Illustration du remplacement local réalisé pour chaque arête $\{u, v\}$ de G afin de construire le graphe G'.

« $\Leftarrow$ ». Supposons que S soit une couverture de sommets de taille au plus k pour le graphe G. On définit alors $D = S$ et pour tout $u \in D$ on pose $p(u) = 2$. On affirme

que l'ensemble D et la fonction p sont une solution à DomPuisVar de coût au plus $4k$ pour le graphe G'. Comme $|S| \leq k$ et que pour tout $u \in S$ on a $p(u) = 2$, on en déduit que $\sum_{v \in D} c(p(v)) \leq k2^2 = 4k$. Le coût est donc bien majoré par $4k$. On sait que S est une couverture de sommets de G ; ainsi pour toute arête $\{u, v\}$ de G, au moins l'une des extrémités appartient à S. On note que, par construction de G', si $\{u, v\}$ était une arête de G, alors u et v se trouvent à distance au plus 2 dans G'. Par conséquent, l'ensemble D, pour lequel tous les sommets ont une puissance égale à 2, dominent tous les sommets de V. L'ensemble V' contient également quatre nouveaux sommets s^1_{uv},$s^{\mathring{1}}_{uv}$, s^2_{uv}, $s^{\mathring{2}}_{uv}$ pour chaque arête de $\{u, v\}$ de G. Par construction, ces quatre nouveaux sommets sont tous situés à distance au plus 2 de u ou de v. Ils sont donc également dominés par les sommets de D. On en déduit que l'on a bien une solution de coût au plus $4k$.

« $\Rightarrow$ ». Réciproquement, supposons que l'on ait une solution D au problème DomPuisVar de coût au plus $4k$ pour G' et montrons que dans ce cas G admet une couverture de sommets de taille au plus k. Dans la suite nous appellerons *sommets originaux* les sommets V de G' et *sommets nouveaux* les sommets $V' \setminus V$ de G'. Bien sûr, l'ensemble D est un sous-ensemble de V'.

Assertion 2. *Soit D une solution contenant des sommets nouveaux, alors on peut trouver en temps polynomial une solution D' de coût $c(D') \leq c(D)$ qui ne contient aucun sommet nouveau.*

Démonstration. Pour démontrer la proposition, nous distinguons les cas suivants :

- Il existe un sommet nouveau $s_{uv} \in D$ avec $p(s_{uv}) \geq 3$. Soit u et v les deux sommets originaux correspondant à s_{uv}. C'est-à-dire qu'il existe une arête $\{u, v\} \in E$ pour laquelle le sommet s_{uv} a été créé. Considérons la solution $D' = (D \setminus \{s_{uv}\}) \cup \{u, v\}, p')$ où p' est la fonction p avec $p'(u) = \max(p(s_{uv}) - 1, p(u))$ et $p'(v) = \max(p(s_{uv}) - 1, p(v))$. On a $c(D', p') \leq c(D, p) - c(p(s_{uv})) + 2c(p(s_{uv}) - 1) = c(D) - 2^{p(s_{uv})} + 2 \cdot 2^{p(s_{uv})-1} = c(D, p)$. Ainsi le coût $c(D', p')$ n'est pas supérieur à $c(D, p)$. De plus, comme $p(s_{uv}) \geq 3$, l'attribution au sommet u et au sommet v d'une puissance au moins égale à $p(s_{uv}) - 1$ permet de dominer tous les sommets situés à distance au plus $p(s_{uv}) - 1$ de ces deux sommets, c'est-à-dire au moins tous les sommets précédemment dominés par s_{uv} avec une puissance $p(s_{uv})$.
- Il existe un sommet nouveau $s_{uv} \in D$ avec $p(s_{uv}) = 2$.
  - Si ce sommet est un sommet de degré 1 (c'est-à-dire qu'il est du type $s^{\mathring{i}}_{uv}$, $i \in \{1, 2\}$) alors les seuls sommets dominés par s_{uv} sont les quatre sommets situés à distance au plus 2. On propose alors de remplacer D et p par $D' = (D \setminus \{s_{uv}\}) \cup \{u\}$ et $p' = p$ avec $p'(u) = \max(2, p(u))$. Là encore, on a $c(D', p') \leq c(D, p)$.
  - Si le nouveau sommet est de degré 3 alors il est du type s^i_{uv}, $i \in \{1, 2\}$. Sans perte de généralité, supposons qu'il est du type s^1_{uv}. Il s'ensuit que le sommet $s^{\mathring{2}}_{uv}$ doit également être dominé. Si $\{s^{\mathring{2}}_{uv}, s^2_{uv}\} \cap D = \emptyset$ alors les sommets u, v, $s^{\mathring{1}}_{uv}$ et s^1_{uv} sont tous déjà dominés par la solution $D' = (D \setminus \{s^1_{uv}\})$ et $p' = p$ qui est de coût moindre. Et si $\{s^{\mathring{2}}_{uv}, s^2_{uv}\} \cap D \neq \emptyset$ dans ce cas, et sans perte de généralité, on suppose que $s^2_{uv} \in D$ (si $s^{\mathring{2}}_{uv} \in D$, alors on peut remplacer dans D le sommet $s^{\mathring{2}}_{uv}$

par le sommet s^2_{uv} en lui attribuant comme puissance celle induite par $\mathring{s}^2_{uv}$ sur lui). Il suffit alors de considérer la solution $D' = (D \setminus \{s^1_{uv}, s^2_{uv}\}) \cup \{u, v\}$ et $p' = p$ avec $p'(u) = \max(2, p(u))$ et $p'(v) = \max(1, p(v))$. (Remarque : on note qu'à ce moment de la transformation, il n'existe plus de sommets nouveaux dont la puissance est supérieure ou égale à 3. Par conséquent, $1 \leq p(s^2_{uv}) \leq 2$ et donc la puissance induite en u et en v par s^1_{uv} et s^2_{uv} n'excède pas 1.)

– Il existe un sommet nouveau $s_{uv} \in D$ avec $p(s_{uv}) = 1$. Là encore, sans perte de généralité nous pouvons supposer que ce sommet est du type s^i_{uv}, $i \in \{1, 2\}$. En effet, si $\mathring{s}^i_{uv} \in D$, alors il est possible de remplacer dans D le sommet $\mathring{s}^i_{uv}$ par le sommet s^i_{uv} avec $p(s^i_{uv}) = p(\mathring{s}^i_{uv})$. De plus, on suppose que s^i_{uv} est s^1_{uv}.
Si $\{\mathring{s}^2_{uv}, s^2_{uv}\} \cap D = \emptyset$ alors un sommet doit induire une puissance d'au moins 2 sur u ou sur v. Il s'ensuit que les sommets $\mathring{s}^1_{uv}$ et s^1_{uv} sont dans ce cas déjà dominés et on peut donc suppprimer la puissance de 1 attribuée à s^1_{uv}. Sinon, on suppose sans perte de généralité que $s^2_{uv} \in D$ et que $p(s^2_{uv}) = 1$. Dans ce cas, on considère la solution de moindre poids définie par $D' = (D \setminus \{s^1_{uv}, s^2_{uv}\}) \cup \{u\}$ et $p' = p$ avec $p'(u) = \max(p(u), 2)$. Bien sûr, $c(p(s^1_{uv})) + c(p(s^2_{uv})) = 4 = 2^2 \leq \max(p(u), 2)$.
Dans chacun des cas étudiés précédemment, on a proposé une transformation de la solution originale de sorte à obtenir une nouvelle solution de moindre poids ne contenant pas de nouveaux sommets.

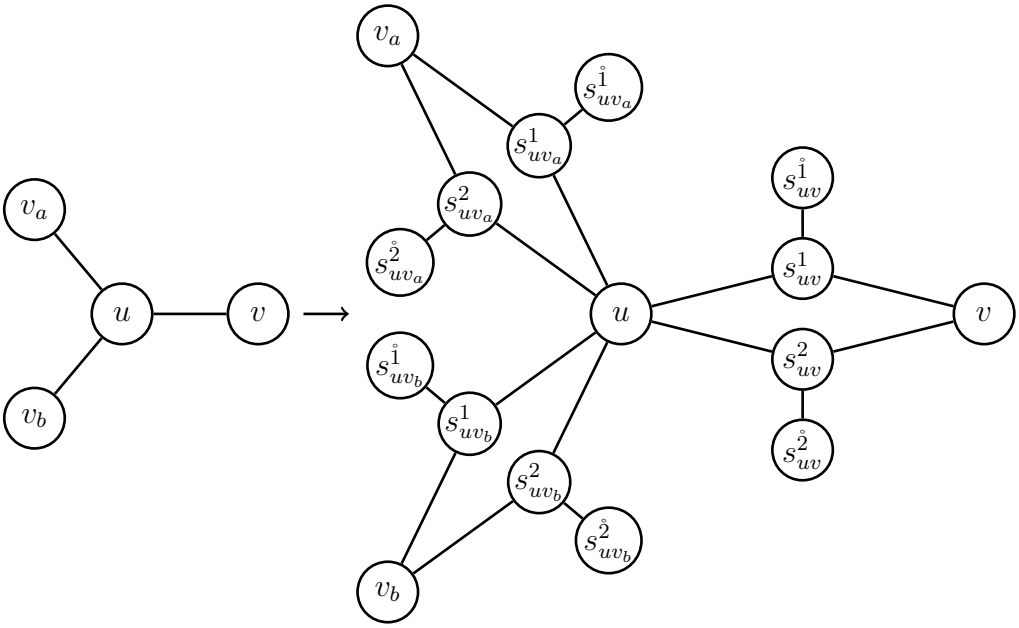


Fig. 7.4 – Illustration du remplacement local utilisé dans la preuve.

□

On montre maintenant la proposition suivante :

Assertion 3. *Soit D une solution contenant uniquement des sommets originaux alors on peut construire en temps polynomial une solution D' de coût moindre ne contenant que des sommets originaux et dont la puissance n'excède pas* 2.

Démonstration. Pour démontrer cette proposition nous distinguons les cas suivants :

- Il existe un sommet original $u \in D$ tel que $p(u) \geq 4$. Comme le graphe G est cubique, le sommet u possède trois sommets originaux situés à distance 2 : v_1, v_2 et v_3.
  On regarde la solution $D' = (D \setminus \{u\}) \cup \{v_1, v_2, v_3\}$, $p' = p$ avec $p'(v_i) = \max(p(u) - 2, p(v_i))$, $i \in \{1, 2, 3\}$.
  On a $c(D', p') \leq c(D, p) + 3c(p(u) - 2) - c(p(u)) = c(D, p) + 3 \cdot 2^{p(u)-2} - 2^{p(u)} = c(D, p) + (3/4 - 1) \cdot 2^{p(u)} \leq c(D, p)$.
- Il existe un sommet original $u \in D$ tel que $p(u) = 3$. En attribuant une puissance égale à 3 à un sommet original u, les trois sommets originaux , nommés v_1, v_2 et v_3, situés à distance 2 de u obtiennent une puissance induite par u égale à 1. Comme la solution ne contient pas de sommets nouveaux, les sommets nouveaux (du type $\mathring{s}^i_{v_j w}$, $i \in \{1, 2\}$, $j \in \{1, 2, 3\}$) situés à distance 2 de v_1, v_2 et v_3 doivent être dominés par des sommets originaux. Ceci implique que les sommets v_1, v_2 et v_3 sont déjà dominés par d'autres sommets que u (de même pour les sommets $s^i_{v_j w}$, $i \in \{1, 2\}$, $j \in \{1, 2, 3\}$). Par conséquent, on peut réduire la puissance de u de $p(u) = 3$ à $p(u) = 2$.

□

Assertion 4. *Soit D une solution contenant uniquement des sommets originaux dont la puissance n'excède pas* 2 *alors on peut construire en temps polynomial une solution D' de coût moindre ne contenant que des sommets originaux dont la puissance est* 2.

Démonstration. Si un sommet original u a pour puissance $p(u) = 1$, alors les sommets nouveaux situés à distance 2 de u doivent être dominés. Comme la solution ne contient que des sommets originaux, il s'ensuit que tous les sommets nouveaux situés à distance au plus 2 de u sont déjà dominés et que u est aussi dominé car il obtient une puissance induite égale à 1 de ses voisins originaux situés à distance 2 dans G'. □

Nous avons montré que si l'on dispose d'une solution D de coût au plus $4k$ pour G' alors on peut construire en temps polynomial une solution D' de coût au plus $4k$ ne contenant que des sommets originaux de puissance 2.

Puisque tous les sommets de type $\mathring{s}^i_{uv}$ doivent être dominés, et qu'il existe un tel sommet dans G' à chaque fois que le graphe G possède une arête $\{u, v\}$, il s'ensuit que pour toute arête $\{u, v\}$ de G, au moins l'un des sommets u ou v appartient à D. Par conséquent, l'ensemble D est une couverture de sommets pour G. De plus, pour tout sommet $u \in D$, $p(u) = 2$ et $c(p(u)) = 4$. Ceci implique que si la solution pour G' est de coût au plus $4k$ alors $|D| \leq k$. Ainsi, on a bien une couverture de sommets de taille au plus k pour G.

Ceci termine notre démonstration montrant que le problème pour lequel l'ensemble des puissances est égal à l'ensemble $\mathbb{N}^*$ et la fonction de coût est définie par $c(i) = 2^i$ pour tout $i \in \mathbb{N}^*$ est NP-complet. □

Dans le but de généraliser le théorème 7.14 (page 164), nous montrons maintenant que le problème est NP-complet si l'ensemble des puissances P est infini et si la fonction de coût n'est plus restreinte à 2^i, pour toute puissance i, mais égale à d^i pour un entier $d \geq 3$.

Théorème 7.15. *Si l'ensemble des puissances P est égal à l'ensemble $\mathbb{N}^*$ et si la fonction de coût satisfait $c(i) = d^i$ pour tout $i \in P$, avec $d \geq 2$ un entier fixé, alors le problème* DomPuisVar *est NP-complet.*

Démonstration. La preuve de ce théorème est plus simple que celle du théorème 7.14 qui démontre le cas où d est égal à 2. Supposons que d soit supérieur ou égal à 3.

Là encore, le problème DomPuisVar appartient à NP. Pour montrer sa NP-complétude, nous faisons une réduction depuis le problème NP-complet Domination restreint aux graphes cubiques [GJ79].

Soit G un graphe cubique. On montre que G admet un ensemble dominant de taille au plus k si et seulement si le graphe G admet une solution au problème DomPuisVar tel que défini dans l'énoncé du théorème de coût au plus $d \cdot k$.

« $\Rightarrow$ ». Soit D un ensemble dominant de G de taille k, alors l'ensemble D pour lequel on affecte a chacun de ses sommets une puissance égale à 1 est une solution au problème DomPuisVar de coût $d \cdot k$.

« $\Leftarrow$ ». Réciproquement, soit D et p une solution à DomPuisVar pour G de coût $c(D, p) = d \cdot k$. Nous montrons que l'on peut construire une solution de coût au plus $d \cdot k$ dont chacun des sommets a pour puissance au plus 1. En effet, soit un sommet $u \in D$ de puissance supérieure ou égale à 2 et soit v_1, v_2, v_3 ses trois voisins dans G. Alors la solution définie par D' et p' avec $D' = D \setminus \{u\} \cup \{v_1, v_2, v_3\}$ et $p' = p$, $p'(v_i) = \max(p(v_i), p(u) - 1)$, $i \in \{1, 2, 3\}$, est de coût au plus $c(D, p)$.

L'idée principale utilisée dans cette preuve est que pour un sommet u d'un graphe cubique, le nombre des sommets situés à distance i de u augmente moins vite que le coût nécessaire pour dominer ces sommets à distance i en utilisant une puissance plus forte en u. Par exemple, dans un graphe cubique, un sommet u a 3 voisins et 6 sommets à distance 2. Pour dominer ces 6 sommets en utilisant u, il faut attribuer à u une puissance d'au moins 2 et de coût au moins égale à $d^2 \geq 9$, alors qu'il était également possible d'attribuer simplement à ses trois voisins une puissance d'au moins 1 et de coût total seulement égal à $3d^1$ avec $3d^1 \leq d^2$. À cause du choix pour la fonction de coût $c(i) = d^i$, avec $d \geq 3$, nous avons toujours la relation $3 \cdot d^{i-1} \leq d^i$, pour tout $i \geq 2$, ce qui justifie le fait qu'il existe toujours une solution optimale D dont les puissances affectées aux sommets de D sont égales à 1.

Puisque pour tout $u \in D$ on a $p(u) = 1$ et $c(D, p) \leq d \cdot k$, on en déduit que l'ensemble D est un ensemble dominant pour le graphe G dont la cardinalité est au plus k. □

7.2.2 Un algorithme exact pour le problème

Nous proposons dans cette section un algorithme exact exponentiel non trivial pour résoudre le problème DomPuisVar.

7.2.2.1 L'algorithme naïf

Un algorithme simple et naïf pour résoudre ce problème demande de tester toutes les puissances possibles pour chacun des sommets du graphe puis de garder une solution de moindre coût. Cet algorithme est décrit par le pseudo-code de **DomPuisVarNaïf** donné sur cette page.

Algorithme DomPuisVarNaïf($G = (V, E)$, P, c)
Entrées: Un graphe $G = (V, E)$, une liste P de puissances, une fonction de coût $c : P \to \mathbb{R}^+$.
Sortie : Un solution optimale $D \subseteq V$ et une fonction $p : D \to P$.

```
D ← V
pour tout v ∈ D, p(v) = min{P}
q ← Σ_{v∈D} p(v)
pour chaque sous-ensembles X ⊆ V faire
    pour chaque fonction p' : X → P faire
        si ⋃_{v∈X} N^{p'(v)}[v] = V et Σ_{v∈X} c(p'(v)) < q alors
            D ← X
            p ← p'
            q ← Σ_{v∈D} p(v)
retourner l'ensemble D et la fonction p
```

Comme l'ensemble des puissances disponibles peut être non fini, il n'est donc pas toujours possible de réaliser l'énumération de toute les fonctions p' dans l'algorithme **DomPuisVarNaïf**. Néanmoins, cette approche reste possible. Pour cela, on considère le graphe G connexe donné en entrée. Si le graphe n'est pas connexe, alors on résout le problème sur chacune de ses composantes connexes. On note $\text{diam}(G)$ le diamètre du graphe (voir section 1.2.1 (page 14) pour un rappel de la définition). Soit $d \in P$ tel que $c(d) = \min\{c(i)$ tel que $i \in P$ et $d \geq \text{diam}(G)\}$. Puis on considère l'ensemble de puissances $P' = \{i \in P$ tel que $i < \text{diam}(G)\} \cup d$. Comme le diamètre de G ne peut être plus grand que n, et même $n - 1$ car le pire des cas est un chemin composé de n sommets, on en déduit que la cardinalité de P' n'excède pas n. De plus, il est naturel de se restreindre à l'ensemble P' puisque les puissances supérieures au diamètre de G n'ont de sens que pour celles ayant le plus petit coût. De par cette restriction, on en déduit que si la cardinalité de P n'est pas bornée, alors il suffit de se restreindre à l'ensemble de puissances P' dont la cardinalité est au plus le nombre n de sommets du graphe (comme pour tout graphe connexe on a $\text{diam}(G) \leq n$). On obtient alors un algorithme trivial qui au pire des cas s'exécute en $\mathcal{O}^*(n^n)$ et qui essaye d'attribuer à chaque sommet l'une des puissances de P'. Si la cardinalité de P est bornée par une constante alors nous pouvons résoudre le problème en temps $\mathcal{O}^*(|P|^n)$ en essayant d'attribuer à chaque sommet du graphe l'une des puissances de P. Il faut ensuite à chaque fois vérifier que l'on aboutit à une solution au problème DomPuisVar et garder une solution de moindre coût afin de retourner une solution optimale.

7.2.2.2 Un algorithme non trivial

Une question importante que l'on se pose demande d'obtenir un algorithme exact plus rapide que cet algorithme naïf pour résoudre cette généralisation de la domination. Nous avons déjà montré que le problème est NP-complet pour quelques cas particuliers et donc de façon générale le problème est NP-complet. C'est pourquoi un algorithme polynomial est impossible à obtenir, sauf si P = NP. Nous donnons dans le théorème suivant un algorithme basé sur le paradigme de la programmation dynamique pour résoudre ce problème et montrons que ce nouvel algorithme est sensiblement plus rapide que l'algorithme trivial.

Théorème 7.16. *Le problème* DomPuisVar *peut être résolu en temps* $\mathcal{O}(n^3 \cdot 2^n)$ *et espace exponentiel.*

Démonstration. Soit un graphe $G = (V, E)$ connexe, un ensemble de puissances P et une fonction de coût $c : P \to \mathbb{R}^+$. Comme pour l'algorithme trivial, si l'ensemble P contient plus de n puissances, où n est le nombre de sommets du graphe, alors on peut, sans perte de généralité, se restreindre à un ensemble P' dont la cardinalité n'excéde pas n.

Pour toute puissance $i \in P'$, on note S^i la collection d'ensembles $\{N^i[v]$ tel que $v \in V\}$ où pour un sommet $v \in V$, $N^i[v]$ désigne l'ensemble des sommets dont la distance à v est au plus i ; c'est-à-dire $N^i[v] = \{u \in V$ tels que $d(u, v) \leq i\}$. On remarque que pour tout $i \in P$, l'ensemble S^i peut être construit en temps polynomial et que $|S^i| \leq n$.

Soit $\mathsf{Opt}[X]$ défini pour tout ensemble $X \subseteq V$. La valeur de $\mathsf{Opt}[X]$ est égale au coût minimum $c(D)$ d'une solution $D \subseteq V$ au problème DomPuisVar pour dominer l'ensemble X. Les valeurs de $\mathsf{Opt}[X]$ sont évaluées en énumérant les ensembles X par ordre de cardinalité croissante et sont initialisées à « $+\infty$ ». Pour tout ensemble X, $\mathsf{Opt}[X]$ est donné par :

$$\begin{aligned} \mathsf{Opt}[\emptyset] &\leftarrow 0 \\ \mathsf{Opt}[X] &\leftarrow \min_{\substack{i \in P' \\ S \in S^i}} \left\{ \mathsf{Opt}[X \setminus S] + c(i) \right\} \end{aligned}$$

Étant donné un ensemble X et les valeurs de $\mathsf{Opt}[X']$ pour tout ensemble $X' \subseteq X$, $\mathsf{Opt}[X]$ peut être calculé en temps $\mathcal{O}(n^3)$ car (i) $\sum_{i \in P'} |S^i| \leq n^2$, pour un ensemble S, puis (ii) l'opération $X \setminus S$ peut être implémentée en $\mathcal{O}(n)$ et ensuite (iii) la recherche de $\mathsf{Opt}[X \setminus S]$ se fait également en $\mathcal{O}(n)$. Comme l'algorithme doit calculer $\mathsf{Opt}[X]$ pour les 2^n sous-ensembles, il s'ensuit que le temps d'exécution total est borné par $\mathcal{O}(n^3 \cdot 2^n) = \mathcal{O}^*(2^n)$. □

7.3 Conclusion et perspectives

Au cours de ces années de thèse, le problème Domination Romaine a suscité mon intérêt pour le résoudre de façon exacte. Finalement la meilleure approche obtenue pour attaquer ce problème est d'utiliser un algorithme que nous avons initialement développé

pour résoudre le problème Domination Partielle. Nous pouvons ainsi résoudre Domination Romaine en temps $\mathcal{O}(1.6183^n)$. Une question intéressante serait de développer un algorithme dont le temps d'exécution au pire des cas est borné par $\mathcal{O}(c^n)$ avec $c < 1.6$.

De façon générale, il serait attrayant d'obtenir un algorithme plus rapide pour Domination Partielle. Bien sûr, ce problème semble « plus difficile à résoudre » que le problème classique Domination, qui n'est qu'un cas particulier de Domination Partielle et pour lequel le meilleur algorithme publié s'exécute en temps $\mathcal{O}(1.5263^n)$ avec un espace polynomial.

Une voie qui pourrait être prometteuse serait de s'attaquer directement au problème k-centre partiel. Nous avons montré qu'un algorithme en $\mathcal{O}^*(c^n)$ pour résoudre ce problème permettrait immédiatement de résoudre Domination Partielle en temps $\mathcal{O}^*(c^n)$. Ce même lien existe entre les problèmes k-centre et Domination. Ainsi, tout comme Fomin *et al.* ont obtenu un algorithme exponentiel pour résoudre Domination à travers une réduction au problème Couverture d'ensembles [FGK05a], il peut être opportun d'étudier le problème k-centre pour obtenir directement un algorithme pour Domination.

Nous avons établi que le problème Domination avec des Puissances Variables peut être résolu en temps $\mathcal{O}^*(2^n)$ en nécessitant un espace exponentiel. Une question naturelle serait d'obtenir un algorithme aussi rapide mais n'utilisant qu'un espace polynomial. Un algorithme de type Brancher & Réduire pourrait peut-être permettre d'obtenir un tel algorithme. De plus, même si l'on s'autorise un espace exponentiel, proposer un algorithme en $\mathcal{O}^*(c^n)$ avec $c < 2$ est également un défi non trivial.

Pour des fonctions de coût particulières, la conception de tels algorithmes devrait être facilitée. Un important travail de recherche pourrait être entrepris autour de ce problème afin de déterminer les fonctions de coût pour lesquelles le problème admet un algorithme polynomial et celles pour lesquelles le problème reste NP-complet et de proposer des algorithmes pour résoudre ces problèmes particuliers.

Conclusion

Les travaux exposés au fil de cette thèse ont trait au problème DOMINATION ainsi qu'à quelques-unes de ses variantes et généralisations. Lorsque cette thèse a débuté sous la direction de Dieter Kratsch, voici trois ans, l'objectif était de s'intéresser aux algorithmes exponentiels, sans préciser les types de problèmes auxquels nous nous attacherions. Cet intérêt avait été motivé, entre autre, par la parution de deux états de l'art par Woeginger [Woe03, Woe04]. Les nombreuses publications pour les problèmes de domination dans les graphes, en particulier pour en étudier la complexité et pour obtenir des algorithmes polynomiaux permettant de les résoudre pour certaines instances ont suscité plusieurs milliers de publications durant ces trentes dernières années. C'est donc assez naturellement que nous nous sommes penchés sur ce problème, pour cette fois proposer des algorithmes exponentiels. Riche de leurs nombreuses variantes et généralisations, ces problèmes de domination ont motivé nos travaux.

Nous avons mis en œuvre dans cette thèse plusieurs techniques pour concevoir et analyser des algorithmes exponentiels pour les résoudre (voir la figure 8.1). Quelques-unes de ces techniques sont déjà connues, comme Brancher & Réduire, et d'autres totalement nouvelles. Aussi, au chapitre 4, nous avons donné une approche générale basée sur la largeur arborescente d'un graphe pour résoudre le problème DOMINATION sur différentes classes de graphes pour lesquelles ce problème reste NP-difficile. L'utilisation de cette approche a requis d'établir des bornes linéaires supérieures sur la largeur arborescente en fonction du degré maximum. En utilisant le paradigme de la programmation dynamique, nous avons aussi montré dans ce chapitre qu'un ensemble dominant de taille minimum peut être calculé en temps $\mathcal{O}^*(2^{n-z})$ pour tout graphe ayant un ensemble stable de taille z. Une conséquence fut l'obtention d'un algorithme pour résoudre DOMINATION sur les graphes bipartis. Une autre conséquence non triviale de ce résultat fut l'obtention d'un algorithme en $\mathcal{O}(1.7088^n)$ pour résoudre ce même problème mais cette fois sur un graphe quelconque. Ce dernier algorithme requiert un espace exponentiel et est le meilleur connu après celui de Fomin *et al.* [FGK05a].

La technique classique Brancher & Réduire, combinée avec une analyse appelée Mesurer pour Conquérir, a ensuite permis d'obtenir au chapitre 5 un algorithme calculant une clique dominante d'un graphe en temps $\mathcal{O}(1.3387^n)$. En utilisant la technique Mémorisation, nous avons montré que ce temps pouvait être amélioré à $\mathcal{O}(1.3234^n)$ avec la nécessité de disposer d'un espace exponentiel. Nous avons proposé des bornes inférieures sur le temps d'exécution de notre algorithme ainsi que sur celui de Culberson *et al.* [CGA05]

résolvant uniquement le problème d'existence. Ces bornes inférieures sont particulièrement utiles lorsqu'on considère des algorithmes construits sur le paradigme Brancher & Réduire. L'analyse de tels algorithmes est difficile et les bornes supérieures établies restent souvent surestimées. Des bornes inférieures informent donc sur la précision de l'analyse mais aussi, elles permettent de comparer le temps d'exécution asymptotique de deux algorithmes exponentiels et donnent des indications sur les possibilités existantes pour améliorer l'analyse. Une question non triviale est de savoir s'il est possible de systématiser le calcul de ces bornes et éventuellement de les trouver automatiquement. Il s'agit alors essentiellement d'exhiber des (sous-)graphes demandant à l'algorithme un grand temps de calcul. Bien sûr, une question encore plus intéressante serait d'améliorer la précision de l'analyse au pire des cas des algorithmes Brancher & Réduire ; c'est déjà l'objectif entrepris par l'analyse de type Mesurer pour Conquérir.

Au chapitre 6, nous nous sommes intéressés au problème demandant d'énumérer les ensembles (σ, ϱ)-dominants d'un graphe. Grâce à une technique totalement originale, appelée Brancher & Recharger, nous avons proposé un algorithme en temps $\mathcal{O}^*(c^n)$, pour un certain $c < 2$, énumérant tous ces ensembles sous certaines conditions sur les ensembles σ et ϱ. Des exemples présentés dans ce chapitre indiquent que sans ces conditions, il existe des graphes pour lesquels le nombre d'ensembles (σ, ϱ)-dominants est de l'ordre de 2^n. Une conséquence intéressante de notre algorithme est l'obtention de bornes combinatoires supérieures sur le nombres de ces ensembles dans un graphe. Finalement, en employant la technique peu connue Trier & Chercher, nous avons établi des algorithmes permettant de compter et de trouver des ensembles $(\{p\}, \{q\})$-dominants de taille maximum et minimum en temps $\mathcal{O}^*(2^{n/2})$. Une astuce nous ayant ensuite autorisée à étendre la technique pour des ensembles σ et ϱ non réduits à des singletons.

Le dernier chapitre s'est intéressé à d'autres généralisations tels que des problèmes de domination partielle. L'algorithme Brancher & Réduire obtenu pour ce problème a également impliqué que le calcul d'un ensemble dominant romain de coût minimum pouvait être réaliser en temps $\mathcal{O}(1.6183^n)$. Enfin, nous avons montré que le problème k-CENTRE PARTIEL pouvait être résolu en $\mathcal{O}^*(c^n)$ si et seulement s'il était possible de résoudre le problème de domination partielle dans le même temps. Finalement, la seconde partie de ce chapitre s'est concentrée sur une généralisation appelée DOMINATION AVEC DES PUISSANCES VARIABLES. En plus de quelques preuves de NP-complétude, un algorithme s'exécutant en $\mathcal{O}^*(2^n)$, et basé sur le paradigme de la Programmation Dynamique, a été donné pour résoudre ce problème.

Cette thèse nous a donné une meilleure compréhension de ce domaine de recherche, des techniques existantes, mais aussi la possibilité d'apporter des techniques nouvelles. Les résultats obtenus, certainement améliorables, laissent envisager de grands espoirs pour l'obtention d'algorithmes exacts encore plus rapides. En 2006, Razgon a publié un algorithme qui s'exécute en $\mathcal{O}(1.1034^n)$ pour résoudre le problème NP-difficile ENSEMBLE STABLE sur les graphes de degré maximum 3. Un tel temps d'exécution nous motive à continuer nos recherches sur le problème DOMINATION.

Néanmoins, on dénombre des problèmes NP-complets dans divers domaines, il serait donc profitable d'investir l'expertise acquise pour la résolution de problèmes de dominations vers d'autres problèmes. C'est déjà ce que nous avons fait en nous intéressant à d'autres problèmes de graphes. Nous avons récemment, à la conférence MFCS 2007, exposé un algorithme pour résoudre un problème d'affectation de fréquence [KKL07]. Ce problème NP-complet appelé étiquetage-$L(2,1)$ demande d'affecter aux sommets d'un graphe un entier de l'intervalle $[0 \dots k]$ de sorte que deux sommets adjacents aient une valeur dont la différence soit au moins 2, et deux sommets ayant un voisin commun aient une valeur différente.

En plus des questions soulevées dans les conclusions des chapitres, des questions plus générales restent ouvertes, comme par exemple l'amélioration de la précision des analyses des algorithmes Brancher & Réduire. En outre, des expérimentations pourraient être menées. L'intuition est que les bornes d'exécution établies sont des bornes au pire des cas ; on pense donc que, pour des graphes issus d'applications réelles, les algorithmes exponentiels pourraient être capables de traiter des entrées avec un grand nombre de sommets. Bien sûr, pour des analyses expérimentales et des implémentations pratiques d'algorithmes de type Brancher & Réduire, l'introduction de techniques de séparation et d'évaluation (en anglais, *Branch & Bound*) pourrait s'avérer bénéfique. L'étude expérimentale du comportement des algorithmes exponentiels pourrait débuter par des tests sur des graphes aléatoires. De plus, la comparaison d'algorithmes nécessitant un espace polynomial ou exponentiel devrait nous renseigner sur l'importance de l'espace utilisé.

Finalement, l'exploration de nouvelles techniques de conceptions d'algorithmes pourrait aider à résoudre des problèmes pour lesquels il semble difficile de faire mieux que l'algorithme naïf. Il serait également intéressant d'étudier la combinaison des algorithmes exponentiels et des algorithmes d'approximation pour obtenir, par exemple, un algorithme d'appoximation avec un facteur constant pour Domination et dont le temps d'exécution serait $\mathcal{O}(c^n)$ avec une constante c proche de 1. Si nous regardons cette idée en considérant un algorithme exponentiel exact alors, en supprimant quelques coûteuses règles de branchement, la complexité au pire des cas de l'algorithme pourrait être améliorée. En contre-partie la condition demandant de déterminer une solution exacte serait relâchée. L'étude des règles de branchement permettrait d'établir un rapport d'approximation pour ce nouvel algorithme. Par exemple, supposons que l'on dispose d'un algorithme pour le problème Domination contenant une règle de branchement demandant de brancher en ajoutant à la solution un sommet ou de brancher en ajoutant deux sommets. Si on remplace cette règle par une règle de réduction demandant l'ajout des deux sommets, alors nous pouvons obtenir un algorithme d'approximation de rapport deux. Récemment, Björklund et Husfeldt ont proposé une famille d'algorithmes exponentiels pour approcher une solution au problème Nombre Chromatique [BH06], abondant ainsi l'idée d'obtenir de tels algorithmes.

D'une façon similaire, il serait intéressant de se questionner sur la possibilité de combiner algorithmes exponentiels et algorithmes randomisés. On pourrait alors obtenir des algorithmes exponentiels s'exécutant en temps $\mathcal{O}(c^n)$, avec c une constante proche de 1, et

qui retourneraient une solution correcte avec une certaine probabilité ; ou dont le temps d'exécution attendu pourrait être borné par $\mathcal{O}(c^n)$. D'un autre côté, ces choix probabilistes pourraient aussi être utiles dans des implémentations. Par exemple, lorsqu'un algorithme exponentiel doit brancher sur un sommet choisi parmi une collection de sommets candidats, le choix pourrait être réalisé de façon aléatoire. L'objectif étant d'éviter, avec une certaine probabilité, d'obtenir des instances « difficiles » comme celles nous ayant servi à établir les bornes inférieures présentées dans cette thèse.

Fig. 8.1 – Résumé des algorithmes exponentiels présentés dans cette thèse.

▷ problème Domination :
- sur les graphes cercles : $\mathcal{O}(1.4956^n)$;
- sur les graphes 4-cordaux : $\mathcal{O}(1.4913^n)$;
- sur les graphes faiblement cordaux : $\mathcal{O}(1.4842^n)$;
- sur les graphes cordaux : $\mathcal{O}(1.4173^n)$;
- sur les graphes c-denses : $\mathcal{O}(1.2303^{n(1+\sqrt{1-2c})})$;
- sur les graphes ayant un ensemble stable de taille z : $\mathcal{O}^*(2^{n-z})$;
- sur les graphes bipartis : $\mathcal{O}(1.4143^n)$;
- sur les graphes arbitraires : $\mathcal{O}(1.7088^n)$;

▷ problème Clique Dominante de taille minimum : $\mathcal{O}(1.3234^n)$;

▷ problème (σ, ϱ)-Domination (existence, énumération, dénombrement, minimisation, maximisation) :
- si σ est sans successeur et les deux ensembles σ et ϱ sont finis : $\mathcal{O}^*(c^n)$, avec $c < 2$ dépendant de $\max\{\max\sigma, \max\varrho\}$;
- si σ est sans successeur, l'un des ensembles est fini et $\sigma \cap \varrho = \emptyset$: $\mathcal{O}^*(c^n)$, avec $c < 2$ dépendant de $\min\{\max\sigma, \max\varrho\}$;

▷ problème (σ, ϱ)-Domination (existence, dénombrement, minimisation, maximisation) :
- si $\sigma = \{p\}$ et $\varrho = \{q\}$: $\mathcal{O}^*(2^{n/2})$;
- si $\sigma = p + m\mathbb{N}$ et $\varrho = q + m\mathbb{N}$: $\mathcal{O}^*(2^{n/2})$;
- si $|\sigma| + |\varrho| = 3$, $\sigma, \varrho \neq \emptyset$: $\mathcal{O}^*(3^{n/2})$;
- si $\sigma = \{p_1 + m\mathbb{N}, p_2 + m\mathbb{N}\}$ et $\varrho = \{q + m\mathbb{N}\}$: $\mathcal{O}^*(3^{n/2})$;
- si $\sigma = \{p + m\mathbb{N}\}$ et $\varrho = \{q_1 + m\mathbb{N}, q_2 + m\mathbb{N}\}$: $\mathcal{O}^*(3^{n/2})$;

▷ problème Domination Partielle : $\mathcal{O}(1.6183^n)$;

▷ problème Domination Romaine : $\mathcal{O}(1.6183^n)$;

▷ problème Domination avec des Puissances Variables : $\mathcal{O}^*(2^n)$.

Bibliographie

[ABFKN02] Alber, J., H.L. Bodlaender, H. Fernau, T. Kloks, R. Niedermeier, Fixed parameter algorithms for dominating set and related problems on planar graphs, *Algorithmica* **33** (2002), p. 461–493. [p. 6, 72, 74, 80, 82]

[AN02] Alber, J., R. Niedermeier, Improved Tree Decomposition Based Algorithms for Domination-like Problems, *Proceedings of LATIN 2002*, *LNCS* **2286**, (2002), p. 221–233. [p. 72, 80, 82]

[And03] Anderson, J.A., *Discrete Mathematics with Combinatorics*. 2^{e} édition. Prentice Hall, 2003. [p. 31, 54]

[ALS91] Arnborg, S., J. Lagergren, D. Seese, Easy problems for tree-decomposable graphs, *Journal of Algorithms* **12** (1991), p. 308–340. [p. 98]

[AP89] Arnborg, S., A. Proskurowski, Linear time algorithms for NP-hard problems restricted to partial k-trees, *Discrete Applied Mathematics* **23** (1989), p. 11–24. [p. 59]

[ACGKMSP99] Ausiello, G., P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi, *Complexity and approximation : combinatorial optimization problems and their approximability properties*. Springer-Verlag, 1999. [p. 25]

[Bei99] Beigel, R., Finding maximum independent sets in sparse and general graphs, *Proceedings of SODA 1999*, p. 856–857. [p. 4, 40]

[BE05] Beigel, R., D. Eppstein, 3-coloring in time $\mathcal{O}(1.3289^n)$, *Journal of Algorithms* **54** (2005), p. 168–204. [p. 41]

[BBC00] Berry, A., J.P. Bordat, O. Cogis, Generating All the Minimal Separators of a Graph, *International Journal of Foundations of Computer Science* **11** (2000), p. 397–403. [p. 84, 85]

[Ber84] Bertossi, A.A., Dominating sets for split and bipartite graphs, *Information Processing Letters* **19** (1984), p. 37–40. [p. 59, 77, 90]

[BH06] Björklund, A., T. Husfeldt, Inclusion-exclusion algorithms for counting set partitions, *Proceedings of the* 47$^{\text{th}}$ *IEEE Symposium on Foundations of Computer Science, FOCS 2006*, p. 575–582. [p. 39, 41, 43, 54, 55, 67, 126, 175]

[BHK07] Björklund, A., T. Husfeldt, M. Koivisto, Set partitioning via inclusion-exclusion, http://www.cs.lu.se/home/Thore_Husfeldt/, article soumis. [p. 55]

[Bla93] Blair, J.R.S., B.W. Peyton, An introduction to chordal graphs and clique trees, *Graph theory and sparse matrix computation*, The IMA Volumes in Mathematics and its Applications **56** , Springer, 1993, p. 1–29. [p. 81]

[Bla73] Blank, M., An estimate of the external stability of a graph without suspended vertices, *Prikl Math i Programmirovanie Vyp* **10** (1973), p. 3–11. [p. 154]

[Blä03] Bläser, M., Computing small partial coverings, *Information Processing Letters* **85** (2003), p. 327–331. [p. 144]

[Bod93] Bodlaender, H.L., A tourist guide through treewidth, *Acta Cybernetica* **11** (1993), p. 1–23. [p. 72]

[Bod98] Bodlaender, H.L., A partial k-arboretum of graphs with bounded treewidth, *Theoretical Computer Science* **209** (1998), p. 1–45. [p. 72]

[Bod05] Bodlaender, H.L., Discovering Treewidth, technical report UU-CS-2005-018, institute of information and computing sciences, Université d'Utrecht, Pays-Bas. [p. 72, 80]

[BGHK95] Bodlaender, H.L., J.R. Gilbert, H. Hafsteinsson, T. Kloks, Approximating treewidth, pathwidth, frontsize and shortest elimination tree, *Journal of Algorithms* **18** (1995), p. 238–255. [p. 86]

[BM93] Bodlaender, H.L., R.H. Möhring, The pathwidth and treewidth of cographs, *SIAM Journal on Discrete Mathematics* **6** (1993), p. 181–188. [p. 72]

[BJ82] Booth, K.S., J.H. Johnson, Dominating Sets in Chordal Graphs, *SIAM Journal on Computing* **11** (1982), p. 191–199. [p. 59]

[BH92] Boppana, R., M.M. Halldórsson, Approximating Maximum Independent Sets by Excluding Subgraphs, *Proceedings of SWAT 1990*, *LNCS* **447**, (1990), p. 13–25. [p. 30]

[BT01] Bouchitté, V., I. Todinca, Treewidth and minimum fill-in : grouping the minimal separators, *SIAM Journal on Computing* **31** (2001), p. 212 – 232. [p. 87]

[BK87] Brandstädt, A., D. Kratsch, On domination problems for permutation and other graphs, *Theoretical Computer Science* **54** (1987), p. 181 – 198. [p. 98, 120]

[BLS99] Brandstädt, A., V.B. Le, J. Spinrad, *Graph Classes : A Survey.* SIAM Monographs on Discrete Mathematics and Applications, SIAM, Philadelphia (1999) [p. 17, 19, 21, 80]

[BK04] Brueggemann, T., W. Kern, An improved local search algorithm for 3-SAT, *Theoretical Computer Science* 329 (2004), p. 303 – 313. [p. 41, 44]

[BB98] Bshouty, N.H., L. Burroughs, Massaging a linear programming solution to give a 2-approximation for a generalization of the Vertex Cover problem. *Proceedings of STACS 1998*, *LNCS* **1373**, (1998), p. 298–308. [p. 144]

[Bys04] Byskov J.M., Enumerating maximal independent sets with applications to graph colouring, *Operations Research Letters* **32** (2004), p. 547–556. [p. 126]

[BMS05] Byskov J.M., B.A. Madsen, B. Skjernaa, On the number of maximal bipartite subgraphs of a graph, *Journal of Graph Theory* **48** (2005), p. 127–132. [p. 126]

[Cai05] Cai, L., Parameterized complexity of cardinality constrained optimization problems, manuscrit, mai 2005. [p. 144]

[Cam94] Cameron, P.J., *Combinatorics : Topics, Techniques, Algorithms*. Cambridge University Press, 1994. [p. 31, 45, 54]

[CG05] Chandran, L.S., F. Grandoni, Refined memorization for vertex cover, *Information Processing Letters* **93** (2005), p. 125–131. [p. 48]

[Cha98] Chang, M.S., Efficient algorithms for the domination problems on interval and circular-arc graphs, *SIAM Journal on Computing* **27** (1998), p. 1671–1694. [p. 59, 69]

[CKX05] Chen, J., I.A. Kanj, G. Xia, Labeled Search Trees and Amortized Analysis : Improved Upper Bounds for NP-Hard Problems, *Algorithmica* **43** (2005), p. 245–273. [p. 47]

[CC04] Chlebík, M, J. Chlebíková, Approximation Hardness of Dominating Set Problems, *Proceedings of ESA 2004*, *LNCS* **3221**, (2004), p. 192–203. [p. 60]

[Chr71] Christofides, N., An algorithm for the chromatic number of a graph, *The Computer Journal* **14** (1971), p. 38–39. [p. 39]

[CDHH04] Cockayne, E.J., P.A. Jr. Dreyer, S.M. Hedetniemi, S.T. Hedetniemi, Roman domination in graphs, *Discrete Mathematics* **278** (2004), p. 11–22. [p. 23, 42, 154]

[Coo71] Cook, S., The Complexity of Theorem Proving Procedures, *Proceedings of the third Annual ACM Symposium on Theory of Computing*, ACM, New York, **2570**, p. 151–158. [p. 1, 14]

[CLRS02] Cormen, T., C. Leiserson, R. Rivest, C. Stein, *Introduction à l'algorithmique*. 2^{e} édition. Dunod, 2002. [p. 12, 42, 45, 50, 131]

[CK90] Cozzens, M.B., L.L. Kelleher, Dominating cliques in graphs, *Discrete Mathematics* **86** (1990), p. 101–116. [p. 98]

[CHL01] Crochemore, M., C. Hancart, T. Lecroq, *Algorithmique du texte*. Vuibert, 2001. [p. 42]

[CGA05] Culberson, J., Y. Gao, C. Anton, Phase Transitions of Dominating Clique Problem and Their Implications to Satisfiability Search, *Proceedings of IJCAI 2005*, p. 78–83. [p. 7, 44, 99, 100, 104, 173]

[Dah06] Dahlöf,V., Exact algorithms for exact satisfiability problems, thèse de doctorat, Linköping, Suède, 2006. [p. 137]

[DJB04] Dahlöf,V., P. Jonsson, R. Beigel, Algorithms for four variants of the exact satisfiability problem, *Theoretical Computer Science* **320** (2004), p. 373–394. [p. 137]

[DLL62] Davis,M., G. Logemann, D. Loveland, A machine program for theorem-proving, *Communications of the ACM* **5** (1962), p. 394–397. [p. 4, 39, 41, 44]

[DP60] Davis,M., H. Putnam, A computing procedure for quantification theory, *Journal of the ACM* **7** (1960), p. 201–215. [p. 4, 39, 41]

[DFFPR06] Dehne, F, M.R. Fellows, H. Fernau, E. Prieto, F. Rosamond, Nonblocker : parameterized algorithmics for minimum dominating set, *Proceedings of SOFSEM 2006*, *LNCS* **3831**, (2006), p. 237–245. [p. 29]

[Die05] Diestel, R., *Graph Theory*. Springer-Verlag, Heidelberg, 2005. [p. 16]

[Dij59] Dijkstra, E.W., A note on two problems in connexion with graphs, *Numerische Mathematik* **1** (1959), p. 269–271. [p. 42]

[Dor07a] Dorn, F., How to use planarity efficiently : new tree-decomposition based algorithms, à paraître dans *proceedings of WG 2007*, (2007). [p. 70]

[Dor07b] Dorn, F., Designing Subexponential Algorithms : Problems, Techniques & Structures, thèse de doctorat, Université de Bergen, Norvège, Juillet 2007. [p. 70]

[DF95a] Downey, R.G., M.R. Fellows, Fixed-Parameter Tractability and Completeness I : Basic Results, *Journal on Computing* **24** (1995), p. 873–921. [p. 29]

[DF95b] Downey, R.G., M.R. Fellows, Fixed-parameter tractability and completeness II : On completeness for $W[1]$, *Theoretical Computer Science* **141** (1995), p. 109–131. [p. 30]

[DF99] Downey, R.G., M.R. Fellows, *Parameterized complexity*, Springer-Verlag, New York, 1999. [p. 25, 26, 29, 35, 58, 65]

[DEHHH06] Dunbar, J.E., D.J. Erwin, T.W. Haynes, S.M. Hedetniemi, S.T. Hedetniemi, Broadcasts in graphs, *Discrete Applied Mathematics* **154** (2006), p. 59–75. [p. 163]

[Epp03] Eppstein D., Small maximal independent sets and faster exact graph coloring, *Journal of Graph Algorithms and Applications* **7** (2003), p. 131–140. [p. 126]

[Epp06] Eppstein D., Quasiconvex analysis of backtracking algorithms, *ACM Transactions on Algorithms* **2** (2006), p. 492–509. [p. 47, 64]

[Epp07] Eppstein D., The traveling salesman problem for cubic graphs, *Journal of Graph Algorithms and Applications* **11** (2007), p. 61–81. [p. 48]

[Erw04] Erwin D.J., Dominating broadcasts in graphs, *Bulletin of the Institute of Combinatorics and its Applications* **42** (2004), p. 89–105. [p. 162, 163]

[Eul1736] Euler L., Solutio problematis ad geometriam situs pertinentis, *Opera Omnia* **7** (1736), p. 128–140. [p. 2]

[FK85] Farber, M., M. Keil, Domination in Permutation Graphs, *Journal of Algorithms* **6** (1985), p. 309–321. [p. 59]

[Fei98] Feige,U., A threshold of $\ln n$ for approximation set cover, *Journal of ACM* **45** (1998), p. 634–652. [p. 35, 60]

[Fei00] Feige,U., Coping with the NP-Hardness of the Graph Bandwidth Problem, *Proceedings of SWAT 2000*, *LNCS* **1851**, (2000), p. 129–145. [p. 41, 44]

[SF98] Sedgewick, R., P. Flajolet, *Introduction à l'analyse des algorithmes*, International Thomson Publishing, 1998. [p. 45]

[FG06] Flum, J., M. Grohe, *Parameterized Complexity Theory*, Springer-Verlag, New York, 2006. [p. 26, 29]

[FKW04] Fomin, F.V., D. Kratsch, G.J. Woeginger, Exact (exponential) algorithms for the dominating set problem, *Proceedings of WG 2004*, *LNCS* **3353**, (2004), p. 245–256. [p. 60, 61, 65, 69, 90, 91, 95, 98, 126, 154]

[FGP06] Fomin, F.V., S. Gaspers et A.V. Pyatkin, Finding a minimum feedback vertex set in time $O(1.7548^n)$, *Proceedings of IWPEC 2006*, *LNCS* **4169**, (2006), p. 184–191. [p. 41, 44, 126]

[FGKKL07] Fomin, F.V., P.A. Golovach, J. Kratochvíl, D. Kratsch, M. Liedloff, Branch and Recharge : Exact Algorithms for Generalized Domination, *Proceedings of WADS 2007*, *LNCS* **4619**, (2007), p. 508–519. [p. 8]

[FGK05a] Fomin, F.V., F. Grandoni, D. Kratsch, Measure and conquer : Domination - A case study, *Proceedings of ICALP 2005*, *LNCS* **3380**, (2006), p. 192–203. [p. 44, 61, 63, 64, 65, 69, 70, 73, 74, 75, 95, 98, 100, 107, 120, 143, 145, 150, 171, 173]

[FGK05b] Fomin, F.V., F. Grandoni, D. Kratsch, Some new techniques in design and analysis of exact (exponential) algorithms, *Bulletin of the EATCS* **87** (2005), p. 47–77. [p. 38, 49, 64, 74, 107]

[FGK06a] Fomin, F.V., F. Grandoni, D. Kratsch, Measure and conquer : A simple $\mathcal{O}(2^{0.288n})$ independent set algorithm, *Proceedings of SODA 2006*, (2006), p. 18–25. [p. 4, 40, 44]

[FGK06b] Fomin, F.V., F. Grandoni, D. Kratsch, Solving Connected Dominating Set Faster than 2^n, *Proceedings of FSTTCS 2006*, *LNCS* **4337**, (2006), p. 152–163. [p. 97, 99]

[FGPS05] Fomin, F.V., F. Grandoni, A.V. Pyatkin, A.A. Stepanov, Bounding the Number of Minimal Dominating Sets : a Measure and Conquer Approach, *Proceedings of ISAAC 2005*, *LNCS* **3827**, (2006), p. 573–582. [p. 65, 67, 126, 136]

[FH06] Fomin, F.V., K. Høie, Pathwidth of cubic graphs and exact algorithms, *Information Processing Letters* **97**, (2006), p. 191–196 [p. 65, 95]

[FS07] Fomin, F.V., A.A. Stepanov, Counting Minimum Weighted Dominating Sets, *Proceedings of COCOON 2007*, to appear. [p. 67]

[FG65] Fulkerson, D.R., O.A. Gross, Incidence matrices and interval graphs, *Pacific Journal of Mathematics* **15**, (1965), p. 835–855 [p. 120]

[GJ79] Garey, M.R., D.S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness*. W.H. Freeman and Company, San Francisco, 1979. [p. 3, 12, 14, 16, 57, 59, 69, 168]

[G02] Gasarch, W., Guest column : The P = ?NP Poll, *SIGACT News Complexity Theory Column* **36** (2002). [p. 2]

[GKL06] Gaspers, S., D. Kratsch, M. Liedloff, Exponential Time Algorithms for the Minimum Dominating Set Problem on Some Graph Classes, *Proceedings of SWAT 2006*, *LNCS* **4059**, (2006), p. 148–159. [p. 6, 89, 90]

[GKLT07] Gaspers, S., D. Kratsch, M. Liedloff, I. Todinca, Exponential Time Algorithms for the Minimum Dominating Set Problem on Some Graph Classes, article sousmis à *ACM Transactions on Algorithms*. [p. 6]

[GL06] Gaspers, S., M. Liedloff, A branch-and-reduce algorithm for finding a minimum independent dominating set in graphs, *Proceedings of WG 2006*, *LNCS* **4271**, (2006), p. 78–89. [p. 97]

[GL07] Gaspers, S., M. Liedloff, A Branch-and-Reduce Algorithm for Finding a Minimum Independent Dominating Set in Graphs, Rapport Technique 344, Department of Informatics, University of Bergen, Bergen, Norway, (janvier 2007). [p. 95]

[Gav72] Gavril, F., Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph, *SIAM Journal on Computing* **1** (1972), p. 180–187. [p. 120]

[Gol80] Golumbic, M.C., *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, New York, 1980. [p. 16, 79, 80, 84]

[Gon85] Gonzalez, T.F., Clustering to minimize the maximum intercluster distance, *Theoretical Computer Science* **38** (1985), p. 293–306. [p. 158]

[GKP03] Graham, R.L., D.E. Knuth, O. Patashnik, *Mathématiques concrètes : Fondations pour l'informatique*. Vuibert, 2003. [p. 31]

[Gra04] Grandoni, F., Exact Algorithms for Hard Graph Problems, thèse de doctorat, Université de Rome "Tor Vergata", Rome, Italie, Mars 2004. [p. 61]

[Gra06] Grandoni, F., A note on the complexity of minimum dominating set, *Journal of Discrete Algorithms* **4** (2006), p. 209–214. [p. 61, 63, 64, 98, 143, 145]

[GP95] Greenlaw, R., R. Petreschi, Cubic graphs, *ACM Computing Surveys* **27** (1995), p. 471–495. [p. 164]

[GNW06] Guo, J., R. Niedermeier, S. Wernicke, Parameterized complexity of vertex cover variants, à paraître dans *Theory of Computing Systems*. [p. 144]

[GRS06] Gupta,S., V. Raman et S. Saurabh, Fast exponential algorithms for Maximum r-regular induced subgraph problems, *Proceedings of FSTTCS 2006*, *LNCS* **4337**, (2006), p. 139–151. [p. 126, 128]

[HHS98a] Haynes, T.W., S.T. Hedetniemi, P.J. Slater, *Fundamentals of Domination in Graphs*. Marcel Dekker, New York, 1998. [p. 3, 24, 60, 97]

[HHS98b] Haynes, T.W., S.T. Hedetniemi, P.J. Slater, *Domination in Graphs : Advanced Topics*. Marcel Dekker, New York, 1998. [p. 3, 24, 60, 97]

[HM97] Hayward, R., H. Meyniel, Weakly Triangulated Graphs I : Co-perfect orderability, *Discrete Applied Mathematics* **73** (1997), p. 199–210. [p. 88]

[Heg06] Heggernes, P., Minimal triangulations of graphs : A survey, *Discrete Mathematics* **306** (2006), p. 297–317. [p. 72]

[HL06] Heggernes, P., D. Lokshtanov, Optimal broadcast domination in polynomial time, *Discrete Mathematics* **306** (2006), p. 3267–3280. [p. 163]

[HK62] Held, M., R.M. Karp, A dynamic programming approach to sequencing problems, *Journal of SIAM*, p. 196–210, 1962. [p. 4, 38, 41, 42, 51]

[HS86] Hochbaum, D.S., D.B. Shmoys, A unified approach to approximation algorithms for bottleneck problems, *Journal of the ACM* **33** (1986), p. 533–550. [p. 158]

[HS74] Horowitz, E., S. Sahni, Computing Partitions with Applications to the Knapsack Problem, *Journal of the ACM* **21** (1974), p. 277–292. [p. 4, 41, 43, 53, 136]

[HMMU04] Horton, S.B., C.N. Meneses, A. Mukherjee, M.E. Uluçakli, A computational study of the broadcast domination problem, Rapport Technique 2004-45, DIMACS Center for Discrete Mathematics and Theoretical Computer Science (2004), p. 277–292. [p. 163]

[HN79] Hsu, W.L., G.L. Nemhauser, Easy and hard bottleneck location problems, *Discete Applied Mathematics* **1** (1979), p. 209–216. [p. 158]

[HT91] Hsu, W.L., K.H. Tsai, Linear time algorithms on circular-arc graphs, *Information Processing Letters* **40** (1991), p. 123–129. [p. 59, 96]

[Huf52] Huffman, D.A., A Method for the Construction of Minimum-Redundancy Codes, *Proceedings of the IRE* **40** (1952), p. 1098–1101. [p. 42]

[IPZ01] Impagliazzo, R., R. Paturi, F. Zane, Which problems have strongly exponential complexity ?, *Journal of Computer and Sytem Sciences* **63** (2001), p. 512–530. [p. 60]

[ISGCI] Information System on Graph Class Inclusions, http://wwwteo.informatik.uni-rostock.de/isgci/. [p. 16]

[Iwa04] Iwama, K., Worst-case upper bounds for kSAT, *Bulletin of the EATCS* **82** (2004), p. 61–71. [p. 38]

[Jia86] Jian, T., An $\mathcal{O}(2^{0.304n})$ algorithm for solving maximum independent set problem, *IEEE Trans. Computers* **35**, (1986), p. 847–851. [p. 4, 39]

[Joh74] Johnson, D.S., Approximation algorithms for combinatorial problems, *Journal of Computer and System Sciences* **9** (1974), p. 256–278. [p. 58]

[Joh87] Johnson, D.S., The NP-completeness column : an ongoing guide, *Journal of Algorithms* **8** (1987), p. 285–303. [p. 16]

[JYP88] Johnson, D.S., M. Yannakakis, C.H. Papadimitriou, On generating all maximal independent sets, *Information Processing Letters*, **27**, (1988), p. 119–123. [p. 39, 52, 99]

[Kar82] Karp, R.M., Dynamic programming meets the principle of inclusion and exclusion, *Operations Research Letters* **1** (1982), p. 49–51. [p. 43]

[Kei93] Keil, J.M., The complexity of domination problems in circle graphs, *Discrete Applied Mathematics* **42** (1993), p. 51–63. [p. 59, 98]

[KT06] Kleinberg, J., É. Tardos, *Algorithm Design.* Addison Wesley, 2006. [p. 12, 42, 50]

[Klo93] Kloks, T., *Treewidth*, PhD. Thesis, Universiteit Utrecht, Pays-Bas, 1993. [p. 72]

[Klo94] Kloks, T., *Treewidth. Computations and approximation*, *LNCS* **842**, Springer-Verlag, Berlin, 1994. [p. 71]

[Klo96] Kloks, T., Treewidth of Circle Graphs, *International Journal of Foundations of Computer Science* **7** (1996), p. 111–120. [p. 89]

[KK98] Kloks, T., D. Kratsch, Listing All Minimal Separators of a Graph, *SIAM Journal on Computing* **27** (1998), p. 605–613. [p. 84]

[Knu97] Knuth, D.E., *The Art of Computer Programming, Volume 3 : Sorting and Searching*. Addison-Wesley, 1997 (second edition). [p. 42, 48, 53]

[Koi06] Koivisto, M., An $\mathcal{O}^*(2^n)$ Algorithm for Graph Coloring and Other Partitioning Problems via Inclusion-Exclusion, *Proceedings of the* 47^{th} *IEEE Symposium on Foundations of Computer Science, FOCS 2006*, p. 583–590. [p. 39, 41, 43, 54, 55, 67]

[KKL07] Kratochvíl, J., D. Kratsch, M. Liedloff, Exact algorithms for $L(2,1)$-labeling of graphs, *Proceedings of MFCS 2007*, *LNCS* **4708**, (2007), p. 513–524. [p. 175]

[Kra98] Kratsch, D., Algorithms, in *Domination in Graphs : Advanced Topics*, T. Haynes, S. Hedetniemi, P. Slater, (eds.), Marcel Dekker, 1998, p. 191–231. [p. 98]

[Kra00] Kratsch, D., Domination and total domination in asteroidal triple-free graphs, *Discrete Applied Mathematics* **99** (2000), p. 111–123. [p. 59, 69]

[Kra05] Kratsch, D., Communication personnelle, 2005. [p. 67]

[KDL94] Kratsch, D., P. Damaschke, A. Lubiw, Dominating cliques in chordal graphs, *Discrete Mathematics* **128** (1994), p. 269–275. [p. 98]

[KL06] Kratsch, D., M. Liedloff, An Exact Algorithm for the Minimum Dominating Clique Problem, *Proceedings of IWPEC 2006*, *LNCS* **4169**, (2006), p. 130–141. [p. 7]

[KL07] Kratsch, D., M. Liedloff, An Exact Algorithm for the Minimum Dominating Clique Problem, à paraître dans *Theoretical Computer Science*. [p. 7]

[KS93] Kratsch, D., L. Stewart, Domination on cocomparability graphs, *SIAM Journal on Discrete Mathematics* **6** (1993), p. 400–417. [p. 59, 69, 98]

[Kru56] Kruskal, J.B., On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem, *Proceedings of the American Mathematical Society* **7** (1956), p. 48–50. [p. 42]

[KL99] Kullmann, O., H. Luckhardt, Algorithms for SAT/TAUT decision based on various measures, manuscrit, février 1999. http://www.cs.swan.ac.uk/~csoliver/ [p. 45]

[Law76] Lawler, E.L., A note on the complexity of the chromatic number problem, *Information Processing Letters* **5** (1976), p. 66–67. [p. 39, 126]

[Lie03] Liedloff, M., *Problèmes NP-complets : introduction au domaine et preuves de NP-complétude*. Rapport de Travail d'Etude et de Recherche, Université Paul Verlaine - Metz, 2003. [p. 12, 58]

[Lie04] Liedloff, M., *Domination Romaine*. Rapport de Diplôme d'Études Approfondies, DEA informatique de Lorraine, École doctorale IAEM Lorraine, Laboratoire d'Informatique Théorique et Appliquée de l'Université Paul Verlaine - Metz, juin 2004. [p. 145, 154]

[LKLP05] Liedloff, M., T. Kloks, J. Liu, S.-L. Peng, Roman Domination over Some Graph Classes, *Proceedings of WG 2005*, *LNCS* **3787**, (2005), p. 103–114. [p. 42, 145]

[MS89] McCuaig, W., B. Shepherd, Domination in graphs with minimum degree two, *Journal of Graph Theory* **13**, (1989), p. 749–762. [p. 154]

[MV80] Micali, S., V.V. Vazirani, An $\mathcal{O}(\sqrt{|V|}|E|)$ Algorithm for Finding Maximum Matching in General Graphs, *Proceedings of FOCS 1980*, p. 17–27. [p. 150]

[MM65] Moon, J.W., L. Moser, On cliques in graphs, *Israel Journal of Mathematics* **3**, (1965), p. 23–28. [p. 39, 44, 52, 80, 99, 126, 136]

[MR95] Motwani, R., P. Raghavan, *Randomized Algorithms*. Cambridge University Press, 1995. [p. 25]

[Nie06] Niedermeier, R., *Invitation to Fixed Parameter Algorithms*. Oxford University Press, U.S.A., 2006. [p. 26, 29, 80]

[NR03] Niedermeier, R., P. Rossmanith, On efficient fixed-parameter algorithms for weighted vertex cover, *Journal of Algorithms* **47**, (2003), p. 63–77. [p. 48]

[Pap94] Papadimitriou, C H., *Computational Complexity*. Addison Wesley, 1994. [p. 12, 60]

[PPSSTW02] Pagourtzis, A., P. Penna, K. Schlude, K. Steinhöfel, D.S. Taylor, P. Widmayer, Server Placements, Roman Domination and other Dominating Set Variants, *Proceedings of IFIP International Conference on Theoretical Computer Science : Foundations of Information Technology in the Era of Networking and Mobile Computing*, (2002), p. 280–291. [p. 154]

[PL88] Peterson, P.A., M.C. Loui, The general maximum matching algorithm of Micali and Vazirani, *Algorithmica* **3**, (1988), p. 511–533. [p. 150]

[Ple80] Plesník, J., On the computational complexity of centers locating in a graph, *Aplikace Matematiky* **25** (1980), p. 445–452. [p. 158]

[PFT92] Press, W H., B.P. Flannery, S.A. Teukolsky,*Numerical Recipes in C : The Art of Scientific Computing*. 2^{e} édition. Cambridge University Press, 1992. [p. 45]

[Pri57] Prim, R.C., Shortest connection networks and some generalisations, *Bell System Technical Journal* **36** (1957), p. 1389–1401. [p. 42]

[RS04] Randerath, B., I. Schiermeyer, Exact algorithms for Minimum Dominating Set, Technical Report zaik-469, Zentrum für Angewandte Informatik, Köln, Germany, April 2004. [p. 61, 97, 98]

[RS97] Raz, R., S. Safra, A sub-constant error-probability low-degree test, and sub-constant error-probability PCP characterization of NP, *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, (1997), p. 475–484. [p. 35, 60]

[Raz06a] Razgon, I., Exact computation of maximum induced forest, *Proceedings of SWAT 2006*, *LNCS* **4059**, (2006), p. 160–171. [p. 44]

[Raz06b] Razgon, I., A Faster Solving of the Maximum Independent Set Problem for Graphs with Maximal Degree 3, *Proceedings of the Second ACiD Workshop*, (2006), p. 131–142. [p. 40]

[Ree96] Reed, B., Paths, stars and the number three, *Combinatorics, Probability and Computing* **5** (1996), p. 277–295. [p. 60, 126]

[Rio58] Riordan, J., *An Introduction to Combinatorial Analysis*. John Wiley & Sons, 1958. Dover Publications, 2003. [p. 31, 54]

[RS86] Robertson, N., P.D. Seymour, Graph Minors. II. Algorithmic Aspects of Tree-Width, *Journal of Algorithms* **7** (1986), p. 309–322. [p. 71]

[Rob86] Robson, J.M., Algorithms for maximum independent sets, *Journal of Algorithms* **7** (1986), p. 425–440. [p. 4, 39, 42, 44, 47, 48, 91, 117, 118]

[Rob01] Robson, J.M., Finding a maximum independent set in time $\mathcal{O}(2^{n/4})$, Technical Report 1251-01, LaBRI, Université Bordeaux I, 2001. [p. 40]

[Roo06] van Rooij, J., Design by Measure and Conquer : An $\mathcal{O}(1.5086^n)$ algorithm for minimum dominating set and similar problems, thèse de Master, Department of Information and Computer Science, Université d'Utrecht, Pays-Bas, INF/SCR.06.05, septembre 2006. [p. 29, 64, 65, 66]

[Ros06] Rosen, K.H., *Discrete Mathematics and Its Applications*. 6$^{\text{e}}$ édition. McGraw-Hill, 2006. [p. 31, 54]

[Sch95] Schiermeyer, I., Problems remaining NP-complete for sparse or dense graphs, *Discussiones Mathematicae. Graph Theory* **15** (1995), p. 33–41. [p. 76]

[Sch06] Schnupp, M., Broadcast Domination with flexible powers, Diplomarbeit, Fakultät für Mathematik und Informatik, Université de Iéna, Allemagne, avril 2006. [p. 162]

[Sch05] Schöning, U., Algorithmics in exponential time, *Proceedings of the 22nd International Symposium on Theoretical Aspects of Computer Science (STACS 2005)*, *LNCS* **3404**, (2005), p. 36–43. [p. 38]

[SS81] Schroeppel, R., A. Shamir, A $T = \mathcal{O}(2^{n/2})$, $S = \mathcal{O}(2^{n/4})$ algorithm for certain NP-complete problems, *SIAM Journal on Computing* **3** (1981), p. 456–464. [p. 43, 136]

[SH75] Shamos, M.I., D. Hoey, Closest-point problem, *Proceedings of the* 16$^{\text{th}}$ *IEEE Symposium on Foundations of Computer Science, FOCS 1975*, p. 151–162. [p. 42]

[ST90] Shindo, M., E. Tomita, Simple algorithms for finding a maximum clique and its worst case time complexity, *Systems and computers in Japan* **21**, (1990), p. 1–13. [p. 4, 40]

[Spi03] Spinrad, J.P., *Efficient Graph Representation, Fields Institute Monographs*. American Mathematical Society, Providence, Rhode Island, 2003. [p. 80]

[Tel94] Telle, J.A., Complexity of domination-type problems in graphs, *Nordic Journal of Computing* **1**, (1994), p. 157–171. [p. 7, 59, 121, 124, 125, 137]

[TT77] Tarjan, R.E., A.E. Trojanowski, Finding a maximum independent set, *SIAM Journal on Computing*, **6**, (1977), p. 537–546. [p. 39, 42, 109]

[Vaz04] Vazirani, V.V., *Approximation Algorithms*. Springer, 2003. [p. 25]

[Vil06] Villanger, Y., Improved exponential-time algorithms for treewidth and minimum fill-in, *Proceedings of LATIN 2006*, *LNCS* **3887**, (2006), p. 800–811. [p. 41]

[Wah04] Wahlström, M., Exact algorithms for finding minimum transversals in rank-3 hypergraphs, *Journal of Algorithms*, **51**, (2004), p. 107–121. [p. 41]

[Wah07] Wahlström, M., Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems, thèse de doctorat, Linköping, Suède, 2007. [p. 41]

[Wes96] West, D., *Introduction to Graph Theory*. Prentice Hall, 1996. [p. 16]

[W-J94] Wickham-Jones, T., *Mathematica Graphics : Techniques and Applications*. Springer-Verlag, 1994. [p. 42]

[Wil05] Williams, R., A new algorithm for optimal 2-constraint satisfaction and its implications, *Theoretical Computer Science*, **348**, (2005), p. 357–365. [p. 41]

[Woe03] Woeginger, G.J., Exact algorithms for NP-hard problems : A survey, *Combinatorial Optimization - Eureka, You Shrink!*, *LNCS* **2570**, (2003), p. 185–207. [p. 3, 12, 30, 38, 41, 42, 126, 136, 173]

[Woe04] Woeginger, G.J., Space and time complexity of exact algorithms : Some open problems, *Proceedings of the 1st International Workshop on Parameterized and Exact Computation (IWPEC 2004)*, *LNCS* **3162**, (2004), p. 281–290. [p. 38, 173]

[YS02] Yato, T., T. Seta, Complexity and completeness of finding another solution and its application to puzzles, *Proceedings of the National Meeting of the Information Processing Society of Japan (IPSJ 2002)*. [p. 1]

Index

Liste des symboles

$\alpha(G)$	taille maximum d'un ensemble stable de G
$\mathcal{O}(\cdot)$	borne asymptotique supérieure sur le temps d'exécution d'un algorithme
$\mathcal{O}^*(\cdot)$	borne asymptotique supérieure sur le temps d'exécution d'un algorithme qui supprime les facteurs polynomiaux
$\chi(G)$	nombre minimum de couleurs nécessaires pour colorier G
$\Delta(G)$	degré maximum d'un sommet du graphe G
$\delta(G)$	degré minimum d'un sommet du graphe G
$\emptyset$	ensemble vide
$\gamma(G)$	taille minimum d'un ensemble dominant de G
$\gamma(G)$	taille minimum d'un ensemble dominant de G
$\gamma_{\mathsf{R}}(G)$	taille minimum d'un ensemble dominant romain de G
$\kappa(G)$	nombre minimum de cliques formant une partition des sommes de G
$\Omega(\cdot)$	borne asymptotique inférieure sur le temps d'exécution d'un algorithme
$\omega(G)$	taille maximum d'une clique de G
$\overline{G}$	complémentaire du graphe G
$\leq_p$	notation pour une réduction polynomiale
$\mathrm{Card}(E)$	cardinalité d'un ensemble E
$\Theta(\cdot)$	borne approchée asymptotique sur le temps d'exécution d'un algorithme
C_l	cycle avec l sommets
$d_G(v)$	degré du sommet v dans G

$diam(G)$	diamètre d'un graphe G
$exc(u)$	excentricité d'un sommet u
$G = (V, E)$	notation habituelle pour désigner un graphe avec V l'ensemble de ses sommets et E l'ensemble de ses arêtes
K_l	graphe complet à l sommets
$n!$	nombre de permutations d'ensemble à n éléments
$N_G(v)$	voisinage ouvert d'un sommet v dans un graphe G
$N_G[v]$	voisinage fermé d'un sommet v dans un graphe G
P_l	chemin avec l sommets