Algorithmes exacts et exponentiels pour les problèmes NP-difficiles sur les graphes et hypergraphes

Manfred Cochefert

École Doctorale
IAEM

Université
de Lorraine

Laboratoire
LITA

THÈSE Informatique
Présentée par

COCHEFERT Manfred

Algorithmes Exacts et Exponentiels pour les Problèmes NP-difficiles sur les Graphes et Hypergraphes

Directeur de Thèse

Pr. KRATSCH Dieter

Président	Dr.	MAFFRAY	Frédéric	G-SCOP, Grenoble INP, Grenoble
Examinateur	Pr.	KRATSCH	Dieter	LITA, Université de Lorraine, Metz
Examinateur	Pr.	MARION	Jean-Yves	LORIA, Université de Lorraine, Nancy
Examinateur	Pr.	NOURINE	Lhouari	LIMOS, Université Blaise Pascal, Clermont-Ferrand
Rapporteur	Pr.	HABIB	Michel	LIAFA, Université Paris Diderot, Paris
Rapporteur	Dr.	HAVET	Frédéric	Directeur de Recherches CNRS, Université Sophia Antipolis, Nice

Résumé

Dans cette Thèse, nous nous intéressons à la résolution exacte de problèmes NP-difficiles sur les graphes et les hypergraphes. Les problèmes que nous étudions regroupent dans un premier temps des variantes du problème classique du Nombre Chromatique. Les variantes de ce problème se distinguent par la difficulté introduite par les relations entre les classes de couleurs, ou la difficulté de reconnaissance des classes de couleurs elles-mêmes. Puis nous ferons le lien avec les problèmes de transversaux sur les hypergraphes. Plus particulièrement, il s'agira de s'intéresser à l'Énumération de Transversaux Minimaux dans un hypergraphe de rang borné. Outre la résolution exacte, nous nous intéressons à la résolution à paramètre fixe. Le problème de Racine Carrée de Graphe est un problème important en Théorie des Graphes. Nous proposons et montrons la solubilité à paramètre fixe de deux problèmes d'optimisation reliés. Finalement, nous nous intéresserons à la résolution de problèmes de graphe, soit en lien avec les problèmes de colorations, soit pour montrer les performances possibles de différents algorithmes en fonction de l'espace mémoire disponible.
Dans cette Thèse, nous aurons à coeur d'appliquer judicieusement la grande majorité des techniques essentielles en Algorithmique Exacte Exponentielle. Principalement, nous appliquerons la *Programmation Dynamique* ou le principe d'*Inclusion-Exclusion* pour les problèmes de coloration. La technique de *Programmation Dynamique* se retrouvera pour d'autres problèmes de cette Thèse, aux côtés d'autres méthodes comme la technique de *Branchement* ou de *Mesurer et Conquérir*.

Mots-Clés. Algorithmes Exacts Exponentiels, Algorithmes Paramétrés, Graphes, Hypergraphes, NP-difficiles, Colorations, Transversaux, Racines de Graphes, Programmation Dynamique, Inclusion-Exclusion, Branchement, Mesurer et Conquérir

Abstract

In this Thesis, we are interested in the exact computation of NP-hard problems on graphs and hypergraphs. Firstly, we study several variants of colorings. Those variants appear harder than the famous Chromatic Number problem, by adding difficulty in recognizing the color classes, or more often by introducing various relationships between them. Then we link to problems of transversals in hypergraphs. More precisely, we are interested in enumerating minimal transversals in bounded ranked hypergraphs. Besides the exact computation, we are also interested in fixed-parameter tractability. For this area, we study two optimization versions of the famous Square Root of Graphs problem. Finally, we will be interested in solving other problems of graphs related to colorings, or in order to compare efficiencies of algorithms depending on the memory space available.
In this Thesis, we will apply most of major techniques in designing exact exponential algorithms. The main techniques we use are *Dynamic Programming*, *Inclusion-Exclusion*, *Branching*, or *Measure and Conquer*.

Keywords . Exact Exponential Algorithms, Fixed-Parameter Algorithms, Graphs, Hypergraphs, NP-hard, Colorings, Transversals, Hitting-Sets, Roots of Graphs, Dynamic Programming, Inclusion-Exclusion, Branching, Measure and Conquer

Table des matières

Remerciements

Cette thèse est un document qui retrace les résultats d'un travail qui a duré trois années consécutives à une longue période de formation universitaire. Tout ce temps passé à l'Université de Metz, puis l'Université de Lorraine, m'a permis de rencontrer ou de travailler avec de nombreuses personnes.

Mes premiers remerciements vont aux membres du jury qui ont tous entièrement assisté à ma soutenance. Merci aux rapporteurs qui ont contribué à améliorer la qualité de la rédaction, grâce à leur lecture attentive et consciencieuse du manuscrit. Merci à tous les membres, pour avoir accepté et accompli leur mission, et m'avoir finalement décerné mon titre de Docteur Informatique de l'Université de Lorraine.

Je remercie ensuite chaleureusement mon directeur de thèse, Dieter Kratsch, qui m'a accompagné tout au long de cette longue entreprise. Dans un premier temps, il a su susciter mon engouement pour la recherche, et cette branche d'informatique théorique si singulière. Avec son aide, j'ai pu décrocher cette thèse. Il m'a orienté dans mes recherches, puis accompagné dans mes travaux. Son livre de 2010 intitulé *Algorithmes Exacts Exponentiels* est sans aucun doute la meilleure référence sur les techniques algorithmiques. Je souhaite encore à ce jour pouvoir maîtriser chaque détail de ce livre. Dieter m'a apporté tout son soutien au sein du laboratoire, et son influence dans le monde de la recherche m'a permis de nouer beaucoup de relations. Grâce à lui, j'ai pu rencontrer beaucoup de monde, de France, de Norvège, d'Allemagne, d'Italie, d'Angleterre, d'Europe ou d'ailleurs. Je remercie chaque personne avec qui nous avons travaillé pour leurs échanges riches, multiculturels, et nos résultats fructueux dont cette thèse retrace l'essentiel. Merci également à Dieter et sa femme pour m'avoir accueilli à Iena.

Une thèse est un travail personnel mais c'est surtout l'occasion d'échanges au sein de groupes de travail. En particulier, certaines personnes sont capables de vous faire progresser dans votre maîtrise. Par une simple phrase ou un échange très long, le déclic suscité par une explication anodine peut vous faire gagner beaucoup de temps. A ce titre, je tiens à remercier très chaleureusement Serge Gaspers, Petr Golovach, Fabrizio Grandoni, Mathieu Liedloff, et Daniël Paulusma.
Je tiens à remercier toutes les personnes avec lesquelles j'ai pu travailler, en particulier le groupe AMT, composé de chercheurs en provenance de Aachen, Trier et Metz. Chaleureusement, je remercie Mathieu Chapelle, Henning Fernau, et Daniel Meister pour nos échanges fructueux. Je remercie également Nicolas Bourgeois, Jean-François Couturier, Bruno Escoffier, Romain Letourneur, et Anthony Perez pour leur contribution. Finalement, mes pensées vont à Pinar Heggernes, Felix Reidl, Peter Rossmanith, Somnath Sikdar, et Pim van't Hof pour tous nos échanges.
Je remercie également les très nombreux doctorants ou chercheurs que j'ai pu rencontrer tout au long de mon parcours, qu'ils aient été mes professeurs, que j'ai pu rencontrer lors de conférences ou de groupes de travail, qu'ils partagent ma spécialité, qu'ils recherchent dans l'informatique, ou qu'ils recherchent dans une toute autre discipline scientifique. En plus d'être de grands chercheurs, certaines personnes sont devenus au fil du temps de bons amis.

Réaliser une thèse nécessite beaucoup de soutien logistique et de moyens. A ce titre, je tiens à remercier tous les membres du LITA et de l'école doctorale IAEM pour leur soutien durant ma thèse. Je souhaite également remercier Ioan Todinca qui m'a apporté du soutien lors de ma visite à Orléans.

Etre un chercheur c'est aussi avoir une vie personnelle, et être doctorant c'est aussi un lot d'épreuves et des difficultés. Pour cela j'ai une pensée toute particulière pour mes amis, pour leur soutien et les

moments partagés en leur compagnie. Finalement, mes derniers remerciements iront à ma femme Marie et notre fille Mélodie née durant cette thèse. Que serais-je sans vous ? Je vous remercie pour votre amour, ainsi que toute la famille pour votre soutien inconditionnel.

Très amicalement,
Manfred Cochefert

Introduction

Sommaire

Présentation de la Thèse

L'Informatique est la science de l'information. D'un point de vue académique, l'Informatique est un domaine scientifique très vaste qui regroupe un ensemble de disciplines très variées. L'objectif commun de ces disciplines est le traitement automatique de données par un ordinateur.
L'Informatique Théorique est une discipline de l'Informatique qui vise la résolution de méta-problèmes informatiques par un ordinateur. Un méta-problème est un problème informatique très général, qui, moyennant quelques variations, peut définir une très grande famille d'autres problèmes informatiques. Un nombre extraordinairement grand de problèmes informatiques peut donc être résolu en résolvant une combinaison particulière de méta-problèmes sur des jeux de données spécifiques. Dans cette optique, les chercheurs ont formalisé des méta-problèmes informatiques, et les données sur lesquelles sont appliqués les méta-problèmes peuvent être représentées au sein de structures bien formalisées comme les graphes.
On identifie ensuite au sein de l'Informatique Théorique différents champs disciplinaires. Par exemple, la Théorie des Graphes vise à exploiter les propriétés structurelles des données organisées en graphes pour la résolution de problèmes. Ce champ disciplinaire est en particulier à l'origine de la caractérisation des graphes en classes de graphes. La Théorie des Hypergraphes vise la même approche mais lorsque les données sont organisées en hypergraphes.
Ma thèse s'inscrit dans le champ disciplinaire de l'Algorithmique. Ce champ disciplinaire de l'Informatique Théorique vise à fournir des méthodes de résolution de problèmes informatiques, méthodes que l'on appelle algorithmes. Un algorithme est une séquence particulière d'instructions très élémentaires.
Grâce aux résultats issus de la Théorie de la Complexité, autre champ disciplinaire de l'Informatique Théorique, on sait qu'il existe en informatique des problèmes très difficiles pour un ordinateur. Typiquement, ce sont des problèmes qui ne sont pas solvables dans un délai raisonnable d'une vie humaine sitôt que la taille des données grandit.
Pour ces problèmes difficiles, il existe alors différentes approches de résolution, chacune débouchant sur une spécialité du champ disciplinaire qu'est l'Algorithmique. Par exemple, pour l'Algorithmique d'Approximation, l'objectif est d'obtenir la meilleure solution qui puisse être calculée en un temps raisonnable. La durée de calcul et la qualité garantie de la solution peuvent alors se retrouver étroitement liées.

Pour ma part, ma thèse s'inscrit dans les spécialités Algorithmique Exacte Exponentielle et Algorithmique Paramétrée de l'Algorithmique, et s'appuie sur les résultats de Théorie des Graphes, Théorie des Hypergraphes et Théorie de la Complexité. Les résultats obtenus pourront par ailleurs avoir des conséquences dans les champs disciplinaires sur lesquels ils s'appuient.
L'Algorithmique Exacte Exponentielle est une spécialité de l'Algorithmique qui vise à résoudre les problèmes difficiles de l'Informatique Théorique de manière exacte, c'est-à-dire qu'on ne cherche non pas à sacrifier la qualité de la solution mais on cherche à obtenir la solution optimale du problème, quitte à imposer à l'ordinateur un temps de calcul extrêmement grand pour obtenir cette solution. L'objectif cependant est de réduire ce temps de calcul au minimum, nous cherchons donc à fournir des méthodes qui soient les plus efficaces possibles, et on mesure la qualité d'une méthode à son temps d'exécution asymptotique.
L'Algorithmique Paramétrée est une autre spécialité de l'Algorithmique qui vise les mêmes objectifs que l'Algorithmique Exacte Exponentielle. Cependant, l'objectif premier de cette spécialité est d'identifier, de comprendre et d'extraire la nature même de la difficulté du problème étudié. Une fois un paramètre choisi, l'objectif est alors de fournir des méthodes de résolution les plus efficaces possible mais qui dépendent le plus exclusivement possible de ce paramètre.
Ces deux spécialités sont essentielles, et suscitent chacune un grand intérêt dans le monde de la recherche.

Exemple. Pour que cette introduction générale soit la plus compréhensible pour tous les lecteurs, je souhaiterais illustrer la portée de ma thèse à travers un exemple très concret de la vie courante. Admettons que je sois un ouvrier du bâtiment, et que je souhaite construire tout seul une réplique exacte de la Tour Eiffel. En qualité d'ouvrier qui est sur le point de construire je suis comparable à un ordinateur, je suis capable d'exécuter des opérations très élémentaires, chacune d'une durée égale et incompressible. Pour construire une Tour Eiffel, je vais exécuter dans un certain ordre une séquence d'opérations très élémentaires, telles que prendre un marteau, taper sur un clou, serrer un boulon... La durée de construction de la Tour Eiffel va alors dépendre du nombre d'opérations élémentaires que j'aurai à effectuer. Mon objectif est bien entendu de construire la Tour Eiffel le plus rapidement possible, car ensuite, j'aimerais construire dans mon jardin une réplique exacte du Tower Bridge. Convenons qu'il s'agit là d'un problème très difficile pour nous, il est très probable que cela soit impossible dans le temps imparti d'une vie d'homme, même si je disposais d'opérations élémentaires plus complexes qui utiliseraient des engins de construction modernes. Je n'ai d'ailleurs pas eu à commencer à construire pour le savoir, car c'est un ami qui m'avait prévenu que c'était difficile, un ami qui connaît la Théorie de la Complexité. En discutant avec un collègue lui aussi féru de construction, il m'apprend que grâce aux méthodes fournies par l'Algorithmique d'Approximation, si je me donne une durée de construction de dix ans, je pourrai certainement construire un édifice qui ressemble d'assez près à une Tour Eiffel. Cependant, cet édifice ne sera très probablement jamais l'édifice que je souhaite construire. Il m'informe également que si je me donne cinq ans, j'aurai probablement un édifice moins ressemblant, et si je me donne vingt ans de construction, j'aurai quelque chose d'un peu plus ressemblant. Mes amis sont gentils, mais je suis quelqu'un de très têtu, et ma femme exige absolument que cet édifice soit la réplique exacte de la Tour Eiffel. Mes amis sont en plus très occupés, je ne peux pas leur demander un coup de main. De toute façon, il faudrait qu'ils soient un million à m'aider pour réduire significativement le temps de construction. Qu'à cela ne tienne, je vais commencer à construire, mon fils et les générations suivantes prendront le relais. Cependant, je vais bien sûr essayer de leur rendre la tâche le plus facile possible, je vais donc développer une méthode qui prenne le moins de temps possible. Ce sera un résultat d'Algorithmique Exacte Exponentielle. Ayant commencé depuis un an, nous

avons raison c'est vraiment un problème très difficile, il va me falloir encore un siècle ou deux de construction. Cependant, à force de construire, je me rends compte de la vraie nature du problème. Je peux construire quelque chose de très large et très haut facilement, la vraie difficulté réside dans le nombre de boulons à serrer, car il faut les serrer dans un certain ordre, et je ne sais pas dans quel ordre les serrer. Si j'avais une dizaine de boulons, je pourrais éventuellement tous les tester dans tous les ordres possibles, mais vu que la Tour Eiffel en possède quelques millions, c'est la nature de la difficulté du problème, voilà un résultat d'Algorithmique Paramétrée.

Dans la spécialité d'Algorithmique Exacte Exponentielle ou Algorithmique Paramétrée, on peut se poser la question de l'utilité de tels travaux. La question est de savoir à quoi ou comment pourront être appliqués de tels résultats ? Outre une meilleure compréhension des problèmes difficiles, avons-nous des applications concrètes ?
Avant de répondre à cette question, j'aimerais mentionner une interrogation très simple et pourtant très fondamentale dans notre discipline de l'Informatique Théorique, à savoir la frontière entre les problèmes simples et les problèmes difficiles. L'Institut de Mathématiques Clay offre un million de dollars à qui pourra statuer sur cette frontière, la classe de complexité **P** est-elle la même que la classe **NP**, ou est-il tout simplement impossible d'y répondre ? Cette question n'a pas trouvé de réponse depuis les fondements de l'Informatique. Si $\mathbf{P} = \mathbf{NP}$, alors une très grande majorité des résultats d'Algorithmique Exacte Exponentielle s'effondrent, car asymptotiquement nos méthodes deviendraient très mauvaises. Si $\mathbf{P} \neq \mathbf{NP}$, alors les résultats de notre spécialité auront largement gagné en solidité et en fondements. Finalement si on ne peut tout simplement pas répondre à cette question, c'est-à-dire si l'on prouve que cette question est indécidable, alors on pourra poser comme axiome que nos résultats sont solides, d'autant plus que certaines méthodes pour résoudre des problèmes faciles peuvent dans des cas pratiques se révéler moins efficaces que des méthodes pour résoudre d'autres problèmes difficiles. La tendance des chercheurs de notre discipline est de penser qu'il est possible de démontrer que $\mathbf{P} \neq \mathbf{NP}$. En d'autres termes, il y aurait en informatique une frontière tangible entre les problèmes faciles et les problèmes difficiles. Cette tendance motive l'existence et les résultats de nos spécialités Algorithmique Exacte Exponentielle et Algorithmique Paramétrée. Il y a donc une justification naturelle à obtenir des résultats tels que présentés dans cette thèse ne serait-ce que par l'existence de ces problèmes difficiles.
En ce qui concerne l'utilité de l'étude des problèmes d'Informatique Théorique, il faut remarquer que ces problèmes ont en majorité une justification d'application concrète et immédiate en biologie, médecine, transports, fouille de données, informatique décisionnelle, intelligence artificielle... Même les problèmes difficiles avaient bien souvent une application concrète bien avant que l'on statue de sa complexité et qu'on juge ces problèmes difficiles. Il est possible que certains problèmes difficiles apparaissent comme des variantes artificielles de problèmes préexistants. Bien qu'ils puissent sembler préfabriqués, leur existence et leur étude ne sont néanmoins pas injustifiées sitôt qu'elles permettent d'améliorer la compréhension, cerner les limites, la nature et l'extension des problèmes originels, ainsi que de développer des méthodes de résolution.
Par ailleurs, les méthodes d'Algorithmique Exacte Exponentielle s'améliorent parallèlement au fait que les limites des ordinateurs sont sans cesse repoussées. De 1970 à 2010, la puissance des ordinateurs communs a suivi les Lois de Moore. Par exemple, en 1990, un ordinateur muni d'un processeur Intel 80486 cadencé à 50 MHz n'était pas capable en une seconde de calculer le nombre chromatique d'un graphe quelconque à $n = 20$ sommets avec l'algorithme de Lawler en temps $O^*(2.44^n)$. Tandis qu'un ordinateur actuel avec un processeur mono-coeur cadencé à 3 GHz est capable de calculer dans cette même seconde le nombre chromatique d'un graphe avec plus de $n = 30$ sommets grâce à l'algorithme de Bjorklünd, Husfeldt et Koivisto qui s'exécute en temps $O^*(2^n)$.

Bien que l'on observe un progrès, la taille des données sur lesquelles on peut résoudre des problèmes difficiles reste toutefois très petite. Les applications réelles concernent des graphes avec des centaines de milliers de villes pour un réseau de transport, des milliards de sommets pour un réseau social, 5.10^{22} atomes pour un gramme de carbone 12 en chimie, ou 10^{80} atomes dans l'univers. La portée des méthodes d'Algorithmique Exacte Exponentielle reste cependant à relativiser avec ce qu'est réellement capable de calculer un ordinateur moderne commun, qui n'est pas forcément capable de résoudre un problème facile sur des données de plus grande taille.
Par exemple, pour une application en temps réel cadencée à 25 images par seconde, et sans astuce particulière, un ordinateur cadencé à 3 GHz ne peut pas déterminer une clique maximum d'un graphe planaire de plus de $n = 100$ sommets. Dans cette classe de graphe, ce problème est pourtant facile, il est de décider si trois ou quatre sommets sont tous reliés entre eux. D'un autre côté, un supercalculateur actuel de type Tianhe-2 de l'université chinoise de technologie de défense, avec une capacité de calcul de 33 PFlops (33.10^{15} opérations à la seconde), est théoriquement capable de résoudre en une seconde le problème difficile de clique maximum de n'importe quel graphe de moins de $n = 200$ sommets avec l'algorithme de Robson qui s'exécute en temps $O^*(1.21^n)$.
La frontière n'est donc pas si grande en ce qui concerne l'application des méthodes que ce soit pour des problèmes faciles ou difficiles. Les applications concrètes pour les problèmes difficiles existent, et il tient tout autant à notre spécialité de développer des méthodes toujours plus efficaces pour résoudre ces problèmes difficiles, qu'à la technologie de s'améliorer pour concevoir des ordinateurs de demain toujours plus puissants. L'objet de ma thèse est en définitive de construire des méthodes les plus efficaces possible pour résoudre des problèmes difficiles sur des données comme les graphes ou les hypergraphes.

Résultats de la Thèse

En 1971, Cook formule le théorème qui porte son nom selon lequel le problème SAT est NP-complet [46]. La difficulté d'un problème informatique vient d'être inventée. Il y a des problèmes décisionnels, auxquels on doit répondre par oui ou par non, pour lesquels il n'existe pas d'algorithmes polynomiaux pour les résoudre. En 1972, Richard Karp s'apppuya sur ce résultat pour établir la liste des célèbres 21 problèmes NP-complets de Karp [121]. Dès lors, la liste des problèmes NP-complets s'allonge, la difficulté des problèmes devient la Théorie de la NP-complétude au sein de la Théorie de la Complexité. En 1979, Garey et Johnson publient un livre qui introduit cette théorie et deviendra une des plus grosses références en Informatique Théorique [94]. On dit alors qu'un problème est NP-difficile en particulier si sa version décisionnelle est NP-complet.
Les problèmes NP-difficiles font depuis lors l'objet d'une étude intense. Cette théorie créé divers champs disciplinaires, parmi lesquelles l'Algorithmique Exacte Exponentielle. En 1976, Lawler publie un des plus célèbres algorithmes de programmation dynamique en Algorithmique Exacte Exponentielle. Cet algorithme permet de calculer le nombre chromatique d'un graphe en temps $O^*(2.44^n)$ [136]. Ce résultat ne sera amélioré que bien des années plus tard en 2009 [20]. En 2003, Woeginger publie une étude et liste les meilleurs algorithmes exacts exponentiels connus pour résoudre certains problèmes NP-difficiles [188]. Dans cette étude, il mentionne une série de problèmes ouverts qui suscite un engouement certain dans le monde de la recherche pour cette spécialité [54]. En 2010, Fomin et Kratsch compilent dans leur livre les grandes techniques algorithmiques connues pour construire des algorithmes exacts exponentiels pour des problèmes NP-difficiles [90].
L'étude des problèmes NP-difficiles a également créé la spécialité de l'Algorithmique Paramétrée. Cette spécialité voit son essor à partir de 1989. Cette discipline est jeune, mais elle suscite depuis ses débuts beaucoup d'intérêts dans le monde de la recherche [65, 164, 165]. L'objectif de cette

spécialité est de confiner dans un paramètre l'explosion combinatoire d'un problème NP-difficile. Ainsi, pour résoudre un problème NP-difficile, on cherche un algorithme de résolution à paramètre fixe, c'est-à-dire que le temps d'exécution exponentiel ne dépend que de ce paramètre, à un facteur polynomial en la taille de l'entrée près. Cette spécialité améliore grandement notre compréhension sur la nature des problèmes étudiés.

Dans cette thèse, nous étudions différents problèmes NP-difficiles sur les graphes ou les hypergraphes. Au regard de l'intérêt toujours plus croissant de ces spécialités, nous nous attacherons à résoudre de manière exacte ou à paramètre fixe les problèmes étudiés. Dans cette thèse, nous établissons entre autres les résultats suivants :

1. Nous étudions la résolution exacte de différentes variantes de problèmes de coloration. Dans un premier temps, nous construisons deux algorithmes de *Programmation Dynamique Classique* pour calculer respectivement le Nombre Grundy et le Nombre Grundy Partiel d'un graphe. Ces algorithmes s'exécutent en temps $O^*(2.44^n)$ et $O^*(4^n)$ avec espace exponentiel.
2. Nous construisons des algorithmes de *Programmation Dynamique Arborescente* pour les problèmes de coloration du Nombre a-Chromatique et du Nombre Harmonieux Chromatique ainsi que d'autres problèmes reliés. En particulier, nous construisons pour ces problèmes les premiers algorithmes de résolution *single-epoxentiel*, respectivement en temps et espace $O^*(2^n)$ et $O^*(8^n)$ pour les arbres et les graphes de genre borné.
3. Finalement, nous étendons l'application du principe d'*Inclusion-Exclusion* à deux problèmes de coloration qui se présentent comme des variantes plus contraintes que le problème du Nombre Chromatique. Nous proposons deux algorithmes d'*Inclusion-Exclusion* pour les problèmes du Nombre b-Chromatique et du Nombre Clique-Chromatique, respectivement en temps $O^*(3^n)$ et $O^*(2^n)$.

4. Nous étudions le problème d'énumération des transversaux minimaux d'un hypergraphe de rang $k \in \mathbb{N}$, communément appelé la version énumération du problème k-Hitting Set. Nous proposons un algorithme de *Branchement* pour ce problème lorsque $k = 3$. Grâce à une analyse affinée qui utilise la méthode de *Mesurer et Conquérir*, nous montrons que notre algorithme s'exécute en temps $O^*(1.6755^n)$.
5. Nous établissons des bornes inférieures pour des algorithmes d'énumération des transversaux minimaux d'un hypergraphe de rang $k \in \mathbb{N}_{\geq 2}$. Puis nous proposons des algorithmes de *Branchement* pour le problème de k-Hitting Set en temps $O^*(\gamma_k^n)$, où $\gamma_k < 2$ est une constante qui dépend de $k \in \mathbb{N}_{\geq 3}$.
6. Finalement, nous étendons le champ d'application de la technique de *Compression Iterative* à nos problèmes de k-Hitting Set. La mise en commun des résultats montre que dans cette Thèse, nous proposons pour tout $3 \leq k \leq 6$, un algorithme d'énumération des transversaux d'un hypergraphe de rang k plus efficace que tous les algorithmes connus jusqu'à présent.

7. Nous étudions la résolution paramétrée du problème de graphe appelé Racine Carrée. Initialement, nous montrons qu'il y a un algorithme en temps polynomial pour décider si un graphe de degré maximum $\Delta(G) \leq 6$ admet une *racine carrée*, et la construit si elle existe.
8. Nous montrons que le problème de décider si un graphe admet une racine carrée qui est un arbre avec au plus $k \in \mathbb{N}$ adjonctions d'arêtes est un problème FPT de paramètre k. Pour se faire, nous construisons un noyau généralisé quadratique pour ce problème, c'est-à-dire que

nous réduisons une instance du problème originel à une instance de $O(k^2)$ sommets d'un autre problème plus général.

9. Finalement, nous montrons que le problème de décider si un graphe admet une racine carrée qui peut être obtenue à partir de lui-même par au plus $k \in \mathbb{N}$ suppressions d'arêtes est un problème FPT de paramètre k. Nous construirons également un algorithme exact exponentiel en temps $O^*(3^{m/3})$ pour calculer une racine carrée d'un graphe si elle existe.

10. Nous étudions une variante du problème Motif de Graphe, la version optimisation du problème Ensemble Tropical Connexe. Nous montrons que sous l'hypothèse de complexité **ETH**, on ne peut pas construire d'algorithme *sous-exponentiel* pour résoudre ce problème, même lorsqu'on se restreint aux arbres. En conséquence, nous proposons un algorithme balancé par trois algorithmes différents et qui s'exécute en temps $O^*(1.5359^n)$.
11. Nous proposons un algorithme de *Branchement* en temps $O^*(1.2721^n)$ pour résoudre le problème Ensemble Tropical Connexe sur les arbres. Nous montrerons que cet algorithme est asymptotiquement plus efficace qu'un algorithme d'énumération optimal de toutes les solutions minimales (en temps $O^*(3^{n/3})$).
12. Finalement, nous proposons deux algorithmes exacts pour le même problème mais dont la contrainte d'espace mémoire disponible est différente. Pour le problème de Domination Romaine Faible, nous construisons deux algorithmes respectivement en temps $O^*(2^n)$ et $O^*(2.23^n)$ avec espace exponentiel et polynomial.

Chapitres de la Thèse

Chapitre 1

Ce premier chapitre correspond aux préliminaires nécessaires à la lecture de cette thèse. Il introduit les notions utilisées en Théorie des Graphes, et Théorie des Hypergraphes, et des classes de graphes connues. Il définit les notations, les mesures du temps d'exécution d'un algorithme. Il introduit les différentes versions d'un problème, et les classes de complexité des problèmes. Il introduit une première approche avec les techniques algorithmiques.

Chapitre 2

Calculer le nombre chromatique d'un graphe est un problème fondamental en informatique. Le meilleur algorithme de résolution exacte était pendant longtemps un algorithme de *Programmation Dynamique* en temps $O^*(2.44^n)$, proposé par Lawler en 1976 [136]. En 2009, Bjorklünd, Husfeldt et Koivisto ont montré comment appliquer le principe d'*Inclusion-Exclusion* et construire un algorithme en temps $O^*(2^n)$ pour calculer le nombre chromatique d'un graphe [20].
Dans ce Chapitre, nous nous consacrons à l'étude et à la résolution exacte de différents problèmes de coloration, chacun pouvant être considéré comme une variante du problème originel, le problème du Nombre Chromatique. Ces problèmes se différencient par une contrainte supplémentaire qui reflète des relations fortes entre les classes de couleurs, ou par une difficulté accrue d'identification d'une classe de couleur potentielle.
L'Algorithme de Lawler de 1976 est un algorithme de *Programmation Dynamique Classique*. Nous montrons dans les Sections 2.2 et 2.3 que cette technique peut être appliquée respectivement aux problèmes du Nombre Grundy et du Nombre Grundy Partiel d'un graphe. Nos algorithmes s'exécutent respectivement en temps $O^*(2.44^n)$ et $O^*(4^n)$ avec espace $O^*(2^n)$ et $O^*(3^n)$.

En Introduction de ce Chapitre, nous montrerons qu'il existe un algorithme de *Programmation Dynamique Arborescente* pour le problème du Nombre Chromatique. Cet algorithme s'exécute en temps *sous-exponentiel* lorsqu'il est appliqué à un graphe planaire. Dans la Section 2.4, nous utilisons cette même technique de *Programmation Dynamique Arborescente* pour la résolution du problème du Nombre a-Chromatique d'un graphe. Notre algorithme calcule le nombre a-chromatique d'un arbre en temps et espace $O^*(2^n)$, ou d'un graphe de genre borné en temps et espace $O^*(8^n)$. Notre résultat constitue donc le premier algorithme en temps *single-exponentiel* pour ce problème et sur ces classes de graphe. Dans la Section 2.7, nous montrerons comment adapter notre algorithme, afin d'obtenir des résultats similaires pour les problèmes du Nombre Pseudo-a-Chromatique et du Nombre Chromatique Harmonieux d'un graphe.
Nous nous attacherons à appliquer ou étendre le principe d'*Inclusion-Exclusion* à d'autres problèmes de coloration que ceux présentés par Bjorklünd, Husfeldt et Koivisto en 2009 [20]. En Section 2.5, nous montrons comment le problème du Nombre b-Chromatique d'un graphe peut être résolu en temps $O^*(3^n)$ avec cette technique. La méthode utilisée s'appuie sur un algorithme utilisant le principe d'*Inclusion-Exclusion* pour résoudre des problèmes généralisés de Partitions Étiquetées.
Finalement, nous montrons comment le problème du Nombre Clique-Chromatique peut être résolu en temps $O^*(2^n)$ également en utilisant ce principe d'*Inclusion-Exclusion*. D'un point de vue de la Théorie de la Complexité, ce problème est pourtant le problème le plus difficile rencontré dans cette thèse, car il est $\sum_2^p$-difficile, puisque décider si une coloration est une clique-coloration ne peut être résolu en temps polynomial. Ce problème de coloration peut être vu comme un problème d'hypergraphes, car il s'agit pour le complément de chaque classe de couleur d'une clique-coloration d'être un transversal de l'hypergraphe des cliques maximales du graphe. Décider la 2-clique-colorabilité d'un graphe constitue alors pour nous le lien avec le Chapitre qui suit, celui de l'énumération des transversaux minimaux d'un hypergraphe de rang borné.
Une partie des résultats de ce Chapitre ont fait l'objet d'une publication en 2014 [42].

Chapitre 3

Les hypergraphes sont une généralisation naturelle des graphes. Un graphe est constitué d'arêtes, que l'on peut considérer comme des sous-ensembles de sommets. Les hypergraphes sont constitués d'hyperarêtes, et forment une famille de sous-ensembles de sommets. Le rang $k \in \mathbb{N}$ d'un hypergraphe représente la taille maximum d'une hyperarête qui le compose. Puisque les arêtes d'un graphe sont des couples de sommets, les graphes sont donc exactement les hypergraphes de rang $k = 2$.
Dans ce Chapitre, nous étudions des algorithmes pour le problème d'énumération des transversaux minimaux dans des hypergraphes de rang $k \geq 2$ borné, appelé la version énumération du problème k-Hitting Set. Dans cette Thèse, nous proposons des algorithmes qui sont plus efficaces que ceux que nous avons publié en 2014 pour les problèmes k-Hitting Set et co$k$-Hitting Set lorsque $3 \leq k \leq 6$ [42]. La Section 3.2 est consacrée à l'étude des propriétés des algorithmes de *Branchement*, technique principalement utilisée tout au long de ce Chapitre, ainsi qu'à l'introduction des outils que nous utiliserons pour nos analyses de temps d'exécution.
En Introduction de ce Chapitre, Sous-Section 3.1.2, nous proposons une borne inférieure sur le temps d'exécution d'un algorithme pour la version énumération du problème k-Hitting Set. Puis dans la Section 3.3, nous construisons un algorithme en temps $O^*(1.6755^n)$ pour l'énumération des transversaux minimaux d'un hypergraphe de rang $k = 3$. Cet algorithme est un algorithme de *Branchement*, et nous utilisons la méthode *Mesurer et Conquérir* pour affiner l'analyse de son temps d'exécution. Ces deux techniques avaient été utilisées par Walström pour la version optimisation du problème 3-Hitting Set en 2004 [186].
Dans la Section 3.4, nous améliorons les algorithmes publiés en 2014 pour le problème k-Hitting

Set. Nous qualifions nos algorithmes de généralisation à deux hyperarêtes des algorithmes publiés en 2014, car notre technique de *Branchement* est intrinsèquement reliée à celle que nous avions déjà mise en œuvre auparavant. Pour $k = 6$, notre algorithme d'énumération est plus efficace que celui utilisé jusqu'à présent pour la version optimisation du problème 6-Hitting Set.
Finalement, en Section 3.5, nous montrons comment la technique de *Compression Iterative*, technique principalement utilisée pour la résolution paramétrée des problèmes, peut être utilisée pour notre problème. Cette technique avait déjà été appliquée pour les versions optimisation et dénombrement du problème k-Hitting Set, nous la mettons ici en œuvre pour sa version énumération.
Les résultats obtenus en Section 3.3 pour $k = 3$ nous permettent, après l'algorithme proposé en Section 3.4, d'améliorer encore le temps d'exécution d'un algorithme pour l'énumération des transversaux minimaux dans un hypergraphe de rang $k = 4$.
Tous les résultats de ce Chapitre sont résumés dans la Table 3.2 de la Conclusion.

Chapitre 4

Le carré d'un graphe est obtenu en reliant tous les sommets ayant un voisin commun. Etant donné un graphe G, on appelle racine carrée de G un graphe H tel que G est le carré de H. Déterminer l'existence, une ou plusieurs racines carrées d'un graphe est un problème difficile. Cette notion a été définie par la Théorie des Graphes, et constitue un problème à fort caractère structurel.
Dans ce Chapitre, nous étudions des algorithmes paramétrés pour déterminer si elle existe une racine carrée d'un graphe. En premier lieu, nous construisons en Section 4.3 pour les graphes de degré maximum $\Delta(G) \leq 6$, un algorithme polynomial pour le problème de Racine Carrée. Les résultats de cette Section s'appuyent sur les propriétés structurelles reliées au problème étudié que nous étayons en Section 4.2.
Dans la Section 4.4, nous montrons que le problème de décider si une racine carrée d'un graphe peut être obtenue à partir d'un arbre avec au plus $k \in \mathbb{N}$ adjonctions d'arêtes est un problème FPT de paramètre k. Ce problème est appelé Racine Carrée Minimum. Notre preuve s'appuye sur la réduction de l'instance du problème à une instance d'un problème plus général, où les arêtes du graphe sont étiquetées. Nous montrerons que la taille de notre *noyau généralisé*, ou encore appelé *binoyau*, est quadratique en k.
Finalement, dans la Section 4.5, nous montrons que le problème de décider si une racine carrée d'un graphe peut être obtenue à partir de lui-même et par au plus $k \in \mathbb{N}$ suppressions d'arêtes est un problème FPT de paramètre k. Ce problème est appelé Racine Carrée Maximum, et nous proposons un algorithme en temps $O^*(3^{m/3})$ pour résoudre de manière exacte ce problème.
Les résultats présentés dans ce Chapitre ont été publiés en 2013, et constituent les premiers résultats d'Algorithmique Paramétrée pour le problème de Racine Carrée de graphes [41].

Chapitre 5

Dans ce Chapitre, nous construisons des algorithmes exacts exponentiels pour résoudre différents problèmes de graphes. Les algorithmes proposés jusqu'à présent utilisent un espace soit exponentiel s'il s'agit de *Programmation Dynamique Classique* ou d'utiliser le principe d'*Inclusion-Exclusion*, soit polynomial s'il s'agit d'une technique de *Branchement*. Dans ce Chapitre, un de nos fils conducteurs est de proposer des alternatives à nos algorithmes selon les contraintes d'environnement au sein duquel les problèmes sont étudiés.
En première partie, Section 5.2, nous étudions le problème d'Ensemble Tropical Connexe, et nous serons contraints par une forte propriété de complexité. En effet nous montrerons en Sous-Section 5.2.2 qu'il n'y a pas, sous une forte hypothèse de complexité, d'algorithme de résolution en

temps *sous-exponentiel* même lorsqu'on se restreint aux arbres. De ce résultat, nous construisons en Sous-Section 5.2.3 trois alternatives de résolution différentes. Nous balancerons alors nos algorithmes, pour construire un algorithme exact exponentiel pour ce problème en temps $O^*(1.5359^n)$.
Dans la Section 5.3, nous étudierons également le problème d'Ensemble Tropical Connexe, mais nous nous restreindrons au seul cas des arbres. Pour cette classe de graphes, nous construirons un algorithme d'énumération de toutes les solutions minimales à ce problème. En montrant un résultat combinatoire, nous montrerons que le temps d'exécution de $O^*(3^{n/3})$ de cet algorithme est optimal. Ce temps d'exécution constituera alors un challenge pour résoudre la version optimisation du problème, que nous relèverons en construisant en Section 5.3.3 un algorithme de *Branchement* en temps $O^*(1.2721^n)$.
Finalement, dans la Section 5.4, nous proposerons deux alternatives pour résoudre un même problème, le problème de Domination Romaine Faible, qui diffèrent selon la contrainte d'espace mémoire utilisée. Lorsque l'espace mémoire utilisé est polynomial au regard du nombre de sommets du graphe, nous proposons un algorithme en temps $O^*(2.23^n)$ pour résoudre ce problème. Si l'on dispose d'un espace mémoire exponentiel, nous proposons un algorithme en temps $O^*(2^n)$ en utilisant une étape de *Pré-traitement* qui utilise la *Programmation Dynamique Classique*.
Les résultats présentés dans ce Chapitre ont fait l'objet d'une publication en 2013 [33] et d'un article accepté pour prochaine publication.

Chapitre 1

Préliminaires

Ce chapitre présente les définitions et les notations utilisées tout au long de cette thèse. Pour apporter de la clarté dans la lecture, les termes que nous utilisons pour la première fois apparaîtront sous la forme *définition*. Les termes que nous utilisons pour la première fois et qui revêtent une importance supérieure apparaîtront sous la forme *définition*.
Nous supposons que le lecteur possède les prérequis de Mathématiques Discrètes. Nous nous référons au livre *Introduction to Algorithms* de Cormen, Leiserson, Rivest et Stein pour toute notion complémentaire d'Algorithmique [141], et nous nous référons au livre de Fomin et Kratsch pour des détails plus fournis sur les techniques d'Algorithmique Exacte Exponentielle [90].
Le lecteur est supposé familier avec la Théorie de la Complexite. Nous nous référons au livre de Garey et Johnson [94] et au livre Arora et Barak [11] pour des explications plus détaillées concernant les classes de complexité des problèmes informatiques.
Le lecteur est également supposé familier avec les notions principales de Théorie des Graphes. Nous nous référons au livre de Diestel pour toute nécessité complémentaire sur les graphes [59]. Nous nous référons au livre *Hypergraphs : combinatorics of finite sets* de Claude Berge pour toute terminologie supplémentaire relative aux hypergraphes [17].
Dans cette thèse, nous analysons les algorithmes uniquement dans le modèle de calcul pourvu d'une machine ***RAM*** à processeur unique. Les *graphes* que nous considérons sont uniquement les *graphes finis*, *non orientés*, *sans boucles* et *sans arêtes multiples*.

Sommaire

1.1 Symboles et Notations

On note $\emptyset$ l'ensemble vide, $\mathbb{N}$ l'ensemble des entiers naturels, et $\mathbb{R}$ l'ensemble des nombres réels. Pour un ensemble de nombres $\mathbb{X}$, et lorsque cela a du sens, on notera $\mathbb{X}_{\text{op } c}$ le sous-ensemble de $\mathbb{X}$ formé par les nombres $x \in \mathbb{X}$ satisfaisant x op c, pour un *opérateur* de comparaison op $\in \{<, >, \leq, \geq, \neq\}$. On notera de plus $\mathbb{X}^*$, $\mathbb{X}_+$ et $\mathbb{X}_+^*$ les ensembles respectifs $\mathbb{X}_{\neq 0}$, $\mathbb{X}_{\geq 0}$ et $\mathbb{X}_{>0}$.
Soit X un ensemble dénombrable d'objets et de cardinalité $n \in \mathbb{N}$. Pour deux entiers $a \leq b \in \mathbb{N}$, on note $\{a, \ldots, b\}$ l'ensemble des entiers naturels compris entre a et b, donc supérieurs ou égal à a et inférieurs ou égal à b. Et si de plus $b \leq n$, on note $\{x_a, \ldots, x_b\} \subseteq X$ le sous-ensemble de X formé par les objets dont les indices sont compris entre a et b.
Pour $n \in \mathbb{N}^*$, on note $\mathfrak{S}_n$ l'ensemble des permutations π de l'ensemble d'entiers naturels $\{1, \ldots, n\}$.
Soient A et B deux *ensembles*. On note $A \subseteq B$ si A est *inclus ou égal* à B. On note $A \subset B$ si A est *strictement inclus* dans B. On note $A \cup B$ ou $A + B$ l'*union* de A et B. On note $A \cap B$ leur *intersection*, on note $A \backslash B$ ou $A - B$ la *différence* de A par B, et on note $A \Delta B$ la *différence symétrique* de A et B. De plus, on note $A \times B$ le *produit cartésien* de A par B.
Soient $\mathcal{U}$ un *univers*, et $A \subseteq \mathcal{U}$ un sous-ensemble de l'univers. Usuellement, on note $2^{\mathcal{U}}$ l'*ensemble des parties* de $\mathcal{U}$, qui est la famille de tous les sous-ensembles de $\mathcal{U}$. En particulier, $A \subseteq \mathcal{U} \Leftrightarrow A \in 2^{\mathcal{U}}$.
On note $|A|$ la *cardinalité* de A, aussi appelée la *taille* de A, et on note $\overline{A}$ le *complémentaire* de A dans $\mathcal{U}$, c'est-à-dire $A \cup \overline{A} = \mathcal{U}$ et $A \cap \overline{A} = \emptyset$.
Soient $\mathcal{U}$ un univers, $A \subseteq \mathcal{U}$ et $\mathcal{F} \subseteq 2^{\mathcal{U}}$ une *famille de sous-ensembles* de $\mathcal{U}$. On dit que A est *maximal* dans $\mathcal{F}$ si pour tout sous-ensemble $B \subseteq \overline{A}$, $A \cup B \notin \mathcal{F}$. De même, on dit que A est *minimal* dans $\mathcal{F}$ si pour tout sous-ensemble $B \subset A$, $B \notin \mathcal{F}$. On dit que A est *maximum* dans $\mathcal{F}$ si pour tout sous-ensemble $B \in \mathcal{F}$, on a $|A| \geq |B|$. De même, on dit que A est *minimum* dans $\mathcal{F}$ si pour tout sous-ensemble $B \in \mathcal{F}$, on a $|A| \leq |B|$. Dans la suite de la thèse, nous utiliserons fréquemment ces notions de manière implicite. La famille $\mathcal{F}$ sera induise par une propriété sur les sous-ensembles de l'univers $\mathcal{U}$.
Soit $\mathcal{U}$ un univers, et soit $\mathcal{F} \subseteq 2^{\mathcal{U}}$ une famille de sous-ensembles de $\mathcal{U}$. On note $\mathcal{F}_\downarrow$ la *clôture descendante* de $\mathcal{F}$, et $\mathcal{F}_\uparrow$ la *clôture ascendante* de $\mathcal{F}$, qui sont définies comme suit : $\mathcal{F}_\downarrow = \{X \subseteq \mathcal{U} \ : \ \exists\ Y \in \mathcal{F},\ X \subseteq Y\}$ et $\mathcal{F}_\uparrow = \{X \subseteq \mathcal{U} \ : \ \exists\ Y \in \mathcal{F},\ Y \subseteq X\}$. Ces familles sont donc composées respectivement de tous les sous-ensembles et super-ensembles des objets appartenant à $\mathcal{F}$.
Une *couverture* d'un univers $\mathcal{U}$ est une famille $\mathcal{F}^* \subseteq 2^{\mathcal{U}}$ de sous-ensembles de $\mathcal{U}$ qui satisfait $\bigcup_{X \in \mathcal{F}^*} X = \mathcal{U}$. Une *partition* de $\mathcal{U}$ est une couverture $\mathcal{F}^*$ de $\mathcal{U}$ avec la propriété que les sous-ensembles qui appartiennent à $\mathcal{F}^*$ soient tous deux à deux disjoints. Formellement, une partition $\mathcal{F}^*$ est une couverture qui satisfait en plus $\forall\ X \neq Y \in \mathcal{F}^* \ : \ X \cap Y = \emptyset$. On dit respectivement de plus que $\mathcal{F}^* \subseteq 2^{\mathcal{U}}$ est une couverture ou une partition de l'univers $\mathcal{U}$ par une famille $\mathcal{F} \subseteq 2^{\mathcal{U}}$ si $\mathcal{F}^*$ est une couverture ou une partition de $\mathcal{U}$ et si de plus $\mathcal{F}^*$ est une sous-famille de $\mathcal{F}$.
Soient $\mathcal{U}$ un univers et $\mathcal{F}^*$ une partition de $\mathcal{U}$. Si $\mathcal{F}^* = \{X_1, \ldots, X_k\}$ partage $\mathcal{U}$ en $k \in \mathbb{N}^*$ sous-ensembles, on dira que $\mathcal{F}^*$ est une k-partition de $\mathcal{U}$. Dans un tel cas, on appelle *coloration* la fonction $\varphi : V \to \{1, \ldots, k\}$ qui attribue à chaque objet de l'univers un sous-ensemble dans $\mathcal{F}^*$, donc qui satisfait $\forall\ 1 \leq i \leq q,\ \forall\ x \in X_i \ : \ \varphi(x) = i$. Les sous-ensembles $X_{i,\ 1 \leq i \leq k}$ de l'univers sont appelés les *classes* de couleurs de la *k-coloration* φ. En pratique, nous appellerons sans distinction k-coloration ou k-partition tant $\mathcal{F}^*$ que φ.
Deux k-couvertures ou k-partitions $\{X_1, \ldots, X_k\}$ et $\{Y_1, \ldots, Y_k\}$ de $\mathcal{U}$ sont dites *distinctes*, si pour toute permutation $\pi \in \mathfrak{S}_k$, $\{X_{\pi(1)}, \ldots, X_{\pi(k)}\} \neq \{Y_1, \ldots, Y_k\}$.

1.2 Problèmes et Complexité

1.2.1 Problèmes et Algorithmes

En Informatique Théorique, un *problème informatique* prend la forme de deux composantes. La première composante, appelée l'*entrée* du problème, concerne le jeu de données sur lequel on souhaite résoudre le problème. L'entrée représente donc l'ensemble des paramètres et des variables déjà initialisées avant le début de résolution du problème. La deuxième composante, que l'on appelle *sortie*, concerne la formulation du problème. C'est une question posée en rapport avec les paramètres de l'entrée, où seule la représentation des données est importante, il doit être fait abstraction de leurs valeurs.
Il peut exister différentes versions d'un même problème informatique. Une *version* d'un problème se distingue dans sa formulation par l'objectif à atteindre dans la résolution. Les versions les plus classiques que nous pouvons rencontrer sont les versions de décision, d'existence, d'optimisation, de dénombrement ou d'énumération. Pour la version de *décision*, l'objectif du problème est toujours de répondre par *oui* ou par *non*, à savoir décider pour l'entrée considérée si le problème admet une solution. Pour la version d'*existence*, l'objectif est de déterminer une solution au problème, en particulier parmi toutes celles qu'admet l'entrée du problème. Pour la version d'*optimisation*, l'objectif de résolution est de déterminer une des meilleures solutions possibles qu'admet l'entrée. Pour la version de *dénombrement*, l'objectif est de déterminer le nombre de solutions au problème qu'admet l'entrée du problème. Finalement, la version d'*énumération* précise que l'objectif est de fournir une liste de toutes les solutions du problème qu'admet l'entrée.
Dans cette Thèse, nous représenterons un problème sous la forme suivante :

> Nom du problème
> *Version* : *Version du problème*
>
> **Entrée** : Jeu de données, paramètres, variables
> **Sortie** : Formulation du problème

On appelle *résoudre* un problème le fait de répondre à la question posée par le problème en tenant compte des paramètres de l'entrée. Sous cette forme et dans la pratique, nous ne présenterons qu'une seule version du problème si la description des autres versions ne génère par d'ambiguïté. Il est intéressant à ce stade d'observer les relations qui régissent les versions d'un même problème. En particulier, résoudre une version d'un problème permet de résoudre, dans l'ordre présenté, les versions précédentes de ce problème.
Une notion importante pour la suite est la notion de taille de l'entrée. L'entrée d'un problème informatique représente un jeu de données avec une structure particulière, qui typiquement peut être représenté par un ensemble d'objets informatiques très élémentaires. La *taille de l'entrée* est alors le nombre d'objets élémentaires nécessaires, ou à considérer, pour représenter le jeu de données. Dans cette Thèse, nous considérons comme objets élémentaires uniquement les objets informatiques qui peuvent être représentés sous la forme d'un entier relatif. Par exemple, lorsque les données sont organisées en graphes, l'entrée est constituée de n sommets et m arêtes, la taille de l'entrée est ici $n + m$, mais on peut considérer uniquement n sommets comme taille de l'entrée.

Pour un problème informatique, un *algorithme* est une méthode de résolution, essentiellement destinée à être utilisée par un ordinateur. Un algorithme est composé d'opérations très élémentaires, compréhensibles et exécutables en temps constant par un ordinateur, que l'on appelle simplement

opérations ou encore *instructions*. A l'instar de la formulation du problème, c'est la représentation des données du problème qui est importante. Un algorithme est une méthode qui doit pouvoir être appliquée pour résoudre le problème quelque soit la valeur du jeu de données en entrée.
Il existe différentes manières d'évaluer la *qualité* d'un algorithme, comme l'analyse en temps moyen ou amorti, et il existe différentes manières d'exprimer son temps d'exécution, en fonction par exemple de la sortie, mais que nous n'aborderons pas. En Algorithmique Exacte Exponentielle et Algorithmique Paramétrée, nous effectuons une *analyse au pire des cas*. Nous mesurons la qualité d'un algorithme par le nombre d'opérations au pire des cas que doit effectuer un ordinateur pour résoudre le problème, étant considéré toutes les valeurs que peut prendre le jeu de données de l'entrée. Ce nombre d'opérations est appelé *temps d'exécution* de l'algorithme, et afin d'être comparé, se doit d'être exprimé en fonction des paramètres du problème. Dans cette Thèse, nous nous attacherons donc uniquement à exprimer le temps d'exécution d'un algorithme en fonction de la taille de l'entrée du problème, selon certaines contraintes d'espace mémoire disponible.

Soient un jeu de données de taille $n \in \mathbb{N}$, un problème informatique sur cette entrée, et $f : \mathbb{N} \to \mathbb{R}_+$ une fonction sur les entiers naturels. Nous utilisons les notations habitutelles o, O, Ω, Θ, et O^* pour exprimer le temps d'exécution $T(n) : \mathbb{N} \to \mathbb{R}_+$ d'un algorithme pour résoudre le problème. Ces notations définissent des bornes asymptotiques des temps d'exécution, et sont satisfaites pour tout $n \geq n_0$, pour un certain rang $n_0 \in \mathbb{N}^*$.
$\diamond$ $T(n) = o(f(n))$ si $\forall\ c \in \mathbb{R}_+^*$, il existe un rang $n_c \geq n_0$ tel que $\forall\ n > n_c$, on a $T(n) < c \cdot f(n)$
$\diamond$ $T(n) = O(f(n))$ s'il existe une constante $c \in \mathbb{R}_+^*$ telle que $T(n) \leq c \cdot f(n)$
$\diamond$ $T(n) = \Omega(f(n))$ s'il existe une constante $c \in \mathbb{R}_+^*$ telle que $T(n) \geq c \cdot f(n)$
$\diamond$ $T(n) = \Theta(f(n))$ s'il existe deux constantes $c_1, c_2 \in \mathbb{R}_+^*$ telle que $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$
$\diamond$ $T(n) = O^*(f(n))$ s'il existe une constante $c \in \mathbb{R}_+^*$ telle que $T(n) = O(n^c \cdot f(n))$
L'efficacité d'un algorithme regroupe plusieurs notions en informatique. Nous dirons dans cette Thèse qu'un algorithme est plus *efficace* qu'un autre si, pour résoudre le même problème et dans les mêmes contraintes d'espace mémoire, son temps d'exécution $T_1(n)$ est asymptotiquement inférieur au temps d'exécution $T_2(n)$ de l'autre, c'est-à-dire si $T_1(n) = o(T_2(n))$.

1.2.2 Complexité Algorithmique

La Théorie de la Complexité est un champ disciplinaire de l'Informatique Théorique qui évalue la difficulté des problèmes informatiques, le niveau de difficulté pour un ordinateur d'obtenir une solution à un problème, et permet d'obtenir une classification des problèmes selon leur *complexité*. Cette classification est effectuée sous le modèle de calcul des machines de ***Turing***, dont la capacité de calcul est assez similaire au modèle ***RAM*** que nous utilisons pour analyser le temps d'exécution de nos algorithmes. Chacun de ces modèles dispose à peu de choses près de la même capacité de calcul que nos ordinateurs modernes. Pour cette Sous-Section et sans autre précision, l'ensemble des entrées possibles pour un problème $\mathcal{P}$ est noté $\mathcal{I}_\mathcal{P}$, et une entrée $x \in \mathcal{I}_\mathcal{P}$ d'un problème $\mathcal{P}$ considéré est toujours de taille n.

Classe P. Un problème de décision $\mathcal{P}$ appartient à la classe **P**, si ce problème peut être résolu en temps polynomial $O(n^c)$, avec $c \in \mathbb{R}_{\geq 1}$, par une machine de ***Turing*** déterministe. ♣
Classe NP. Un problème de décision $\mathcal{P}$ appartient à la classe **NP**, si ce problème peut être résolu en temps polynomial par une machine de ***Turing*** non-déterministe. Une définition équivalente est qu'un problème décisionnel est dans **NP**, si vérifier la validité d'une solution de $\mathcal{P}$ est un problème de la classe **P**. ♣

Classe coNP. Un problème de décision est dans la classe **coNP**, si vérifier qu'un contre-exemple au problème $\mathcal{P}$ n'est effectivement pas une solution de $\mathcal{P}$ est un problème de la classe **P**. ♣

Par ces définitions, les relations $\mathbf{P} \subseteq \mathbf{NP}$ et $\mathbf{P} \subseteq \mathbf{coNP}$ sont immédiates, tandis que la relation $\mathbf{NP} \subseteq \mathbf{P}$ constitue une question ouverte, un des problèmes du prix du millénaire de l'Institut de mathématiques Clay. Les scientifiques s'accordent à dire que $\mathbf{P} \neq \mathbf{NP}$. La relation $\mathbf{NP} = \mathbf{coNP}$ est une question ouverte, et si comme les scientifiques le pensent $\mathbf{NP} \neq \mathbf{coNP}$, alors on aurait $\mathbf{P} \neq \mathbf{NP}$.

Soient $\mathcal{A}$ et $\mathcal{B}$ deux problèmes de décision, et soient $\mathcal{I}_\mathcal{A}$ et $\mathcal{I}_\mathcal{B}$ l'ensemble des entrées respectives de ces problèmes. On dit que $\mathcal{A}$ est *réductible* en temps polynomial à $\mathcal{B}$, noté $\mathcal{A} \leq_p \mathcal{B}$, s'il existe une *Réduction polynomiale* $f : \mathcal{I}_\mathcal{A} \to \mathcal{I}_\mathcal{B}$ calculable en temps polynomial, telle que l'entrée de $x \in \mathcal{I}_\mathcal{A}$ admet une solution pour le problème $\mathcal{A}$ si et seulement si $f(x) \in \mathcal{I}_\mathcal{B}$ admet une solution pour le problème $\mathcal{B}$. Dans un cas pratique, on peut dire que $\mathcal{A}$ est réductible en $\mathcal{B}$ si l'on peut résoudre le problème $\mathcal{A}$ sur une entrée $a \in \mathcal{I}_\mathcal{A}$ de taille n_a, par un nombre polynomial de résolutions du problème $\mathcal{B}$ sur des entrées $b \in \mathcal{I}_\mathcal{B}$ de taille $n_b = O(n_a^c)$, pour une certaine constante $c \in \mathbb{R}_{\geq 1}$.
Soit **C** une classe de complexité. On dit qu'un problème $\mathcal{P}$ est C-*difficile* si pour tout problème $\mathcal{C} \in \mathbf{C}$, on a $\mathcal{C} \leq_p \mathcal{P}$. Si de plus $\mathcal{P} \in \mathbf{C}$, on dit que $\mathcal{P}$ est C-*complet*. La classe des problèmes C-complets constitue une relation d'équivalence par la relation de réduction polynomiale. Ainsi pour montrer qu'un problème $\mathcal{P}$ est C-difficile dans un cas pratique, il suffira grâce à la transitivité de la réduction polynomiale, de montrer qu'il existe un problème $\mathcal{C}$ qui soit C-complet et qui satisfasse $\mathcal{C} \leq_p \mathcal{P}$. Pour montrer que $\mathcal{P}$ est C-complet, il suffira de montrer en plus que $\mathcal{P} \in \mathbf{C}$.
Pour un problème $\mathcal{P} \in \mathbf{C}$, on définit le problème *opposé* $\overline{\mathcal{P}}$ de $\mathcal{P}$, qui consiste à retourner pour toute entrée $x \in \mathcal{I}_\mathcal{P}$ le *booléen* opposé à ce que retourne le problème $\mathcal{P}$ pour cette entrée. Nous pouvons de cette façon généraliser la classe **coNP** à d'autres classes de complexité **coC**, où **coC** est la classe composée des problèmes opposés de tous les problèmes de la classe **C**.

Les notions de C-complétude et C-difficulté construisent de nombreuses classes de complexité. En particulier, si la classe **C** peut ne concerner que des problèmes de décision, la C-difficulté est une notion qui s'étend à toutes les versions des problèmes. Pour le cas de la classe **NP**, Cook a montré que quelque soit le problème $\mathcal{P} \in \mathbf{NP}$, $\mathcal{P} \leq_p$ SAT, ainsi que SAT $\leq_p$ 3-SAT [46]. La classe **NPC** des problèmes NP-complets n'est par conséquent pas vide, puisque SAT $\in$ **NPC** et 3-SAT $\in$ **NPC**. Plus tard, Karp publia en 1972 la célèbre liste des 21 problèmes NP-complets de Karp [121]. Aujourd'hui, il existe une très longue liste de problèmes NP-complets et NP-difficiles [94]. Au-delà des classes ci-dessus, il existe d'autres classes de complexité des problèmes informatiques.

Classe FP. Cette classe est l'extension naturelle de la classe **P** à toutes les versions des problèmes informatiques. En effet, un problème $\mathcal{P}$ est dans la classe **FP** s'il peut être résolu en temps polynomial par une machine de ***Turing*** déterministe. ♣
Classe #P. Un problème de dénombrement $\mathcal{P}$ appartient à la classe **#P**, s'il existe pour une machine de ***Turing*** non-déterministe, un algorithme qui détermine pour toute entrée $x \in \mathcal{I}_\mathcal{P}$ le nombre de solutions distinctes du problème $\mathcal{P}$ qu'admet x, et dont le temps d'exécution est polynomial en la taille de l'entrée. ♣

En Théorie de la Complexité, la notion d'oracle est une notion importante. Un *oracle* de ***Turing*** représente une machine virtuelle, qui est capable de résoudre des problèmes décisionnels d'une certaine complexité en temps constant. Pour deux classes de complexité **A** et **B**, on définit la classe de complexité $\mathbf{A}^\mathbf{B}$ par l'ensemble des problèmes décisionnels qui peuvent être résolus par

une machine de ***Turing*** en temps polynomial, à l'aide d'un oracle résolvant en temps constant des problèmes B-complets. Cette notion permet de construire par récurrence des classes de problèmes que nous pourrons rencontrer dans cette Thèse.

Classes Δ. On définit la classe de complexité $\Delta_0^p = \mathbf{P}$, puis on définit pour tout $i \in \mathbb{N}^*$, la classe $\Delta_{i+1}^p = \mathbf{P}^{\sum_i^p}$. Nous pouvons constater que $\Delta_1^p = \mathbf{P}$, et que $\Delta_2^p = \mathbf{P}^{\mathbf{NP}}$. ♣
Classes $\sum$. On définit la classe de complexité $\sum_0^p = \mathbf{P}$, puis on définit pour tout $i \in \mathbb{N}^*$, la classe $\sum_{i+1}^p = \mathbf{NP}^{\sum_i^p}$. Nous pouvons constater que $\sum_1^p = \mathbf{NP}$, et que $\sum_2^p = \mathbf{NP}^{\mathbf{NP}}$. ♣
Classes Π. On définit la classe de complexité $\Pi_0^p = \mathbf{P}$, puis on définit pour tout $i \in \mathbb{N}^*$, la classe $\Pi_{i+1}^p = \mathbf{co}\sum_{i+1}^p$. Nous pouvons constater que $\Pi_1^p = \mathbf{coNP}$. ♣

Les classes suivantes sont reliées au temps d'exécution espéré pour l'Algorithmique Exacte Exponentielle, en particulier pour la résolution de problèmes NP-difficiles.
Classe EXP. Cette classe regroupe l'ensemble des problèmes décisionnels qui peuvent être résolus par une machine de ***Turing*** déterministe en temps $O(2^{n^c})$, pour une certaine constante $c \in \mathbb{R}_+$. ♣
Classe NEXP. Cette classe est la version non-déterministe de la classe **EXP**. A savoir un problème décisionnel $\mathcal{P}$ appartient à la classe **NEXP** s'il peut être résolu par une machine de ***Turing*** non-déterministe en temps $O(2^{n^c})$, pour une certaine constante $c \in \mathbb{R}_+$. ♣
Classe SUBEXP. La classe **SUBEXP** regroupe les problèmes qui peuvent être résolus par une machine de ***Turing*** déterministe en temps $O(2^{o(n)})$. ♣
Par construction, ces classes satisfont les relations $\mathbf{P} \subseteq \mathbf{NP} \subseteq \mathbf{EXP} \subseteq \mathbf{NEXP}$.

Les classes suivantes sont reliées à l'Algorithmique Paramétrée.
Classe FPT. La classe **FPT** rassemble les problèmes décisionnels $\mathcal{P}$, pour un certain paramètre t, sont solubles en temps $O^*(f(t))$, pour une certaine fonction f arbitraire. ♣
Classe XP. La classe **XP** rassemble les problèmes décisionnels $\mathcal{P}$ qui, pour un certain paramètre t, sont solubles en temps $O(n^{f(t)})$, pour une certaine fonction f arbitraire. ♣
Les classes de complexité $\mathbf{W}[i]$ regroupent des problèmes que nous ne décrivons pas ici mais qui ont un intérêt pour l'Algorithmique Paramétrée. En particulier, ces classes de complexité vérifient les relations $\mathbf{FPT} \subseteq \mathbf{W}[1] \subseteq \ldots \subseteq \mathbf{W}[i] \subseteq \mathbf{W}[i+1]$. De plus, si $\mathbf{FPT} = \mathbf{W}[1]$, alors $\mathbf{NP} \subseteq \mathbf{SUBEXP}$. De fait, il est très probable que les inclusions de ces relations soient strictes, et qu'aucun problème W[i]-complet ne soit **FPT**.

La Théorie de la Complexité est par nature reliée à la Théorie des Langages.
La *Logique Monadique du Second Ordre*, notée **MSOL**, est un système logique qui permet d'exprimer des propriétés de graphes. Nous référons au livre de Courcelle et Engelfriet pour une introduction à ce système [50]. Un des résultats les plus puissants en Algorithmique Paramétrée concernant **MSOL** est que tout problème de graphe qui peut être exprimé dans cette logique est **FPT** lorsque le paramètre est la largeur arborescente du graphe [48]. La logique **MSOL** est une restriction de la logique du second ordre. En 1974, Fagin a montré que les problèmes de la classe **NP** sont les problèmes décisionnels qui correspondent à une propriété de graphe qui peut être exprimée en *Logique Existentielle du Second Ordre*, notée **ESO** [74].
Classe SNP. En Théorie de la Complexité, la classe $\mathbf{SNP} \subseteq \mathbf{NP}$ regroupe les problèmes qui correspondent à une propriété de graphe qui peut être exprimée uniquement avec des quantificateurs universels ($\forall$) sur les sommets. En particulier, pour tout $k \in \mathbb{N}_{\geq 3}$, k-SAT $\in$ **SNP**. ♣

Cette Thèse s'inscrit dans les spécialités Algorithmique Exacte Exponentielle et Algorithmique Paramétrée de l'Algorithmique. Nous nous attachons à fournir des algorithmes de résolution pour des problèmes dits *difficiles*, c'est-à-dire des problèmes qui ne sont pas dans la classe **FP**. Plus particulièrement, les problèmes que nous étudions sont des problèmes de graphes ou d'hypergraphes. Ces objets théoriques, introduits respectivement dans les Sections 1.3 et 1.4, caractérisent la structure de l'entrée d'un problème étudié, et permettent de modéliser un très grand nombre d'objets relationnels tant réels que virtuels que nous connaissons. A l'opposé des problèmes difficiles, on appelle problèmes *faciles* ou *polynomiaux* les problèmes de la classe **FP**. Au regard de la complexité d'un problème difficile, et sous conditions d'espace mémoire disponible, nous nous intéressons à fournir les meilleurs algorithmes exacts exponentiels pour résoudre ce problème. Nous nous intéressons de plus, pour les problèmes décisionnels difficiles, à décider pour un paramètre t fixé, si le problème est soluble à paramètre fixe, c'est-à-dire appartient à la classe **FPT**.

Certains résultats de la Théorie de la Complexité sont prouvés, alors que d'autres sont à l'heure actuelle des hypothèses de travail pour l'Informatique Théorique. Tout comme la majorité des travaux de l'Algorithmique Exacte Exponentielle et Algorithmique Paramétrée, cette Thèse n'échappe pas à ce principe. Certains résultats de cette Thèse ne sont valables que tant que certaines hypothèses de la Théorie de la Complexité ne sont pas invalidées. Si dans le futur ces hypothèses venaient à trouver une preuve, ou à être transformées en axiome, les résultats de cette Thèse seraient confortés. A l'inverse, si ces hypothèses de travail se révélaient être fausses, beaucoup des algorithmes dans cette Thèse deviendraient obsolètes. Bien entendu, ce sont des hypothèses fondamentales et historiques de l'Informatique Théorique. Nous résumons ici certaines hypothèses sur lesquelles nous travaillons dans cette Thèse.

- ◇ **P** $\neq$ **NP** : Tout problème NP-difficile ne peut être résolu en temps polynomial.
- ◇ ***ETH***, **E***xponential* **T***ime* **H***ypothesis* : 3-SAT $\notin$ **SUBEXP**.
- ◇ **SNP** $\neq$ **SUBEXP** : Conséquence de **ETH**, car 3-SAT $\in$ **SNP**.
- ◇ ***SETH***, **S***trong* **E***xponential* **T***ime* **H***ypothesis* : $\forall\ \epsilon < 1$, SAT ne peut être résolu en temps $O^*(2^{\epsilon n})$.
- ◇ $\forall\ i \in \mathbb{N}^*$: **FPT** $\neq$ **W**$[i]$ $\neq$ **W**$[i+1]$.

1.3 Notions de Graphes

1.3.1 Fondamentaux de Graphes

Soit G un graphe. Nous dénotons par V_G son *ensemble de sommets* et par $E_G \subseteq V_G \times V_G$ son *ensemble d'arêtes*. On note donc $G = (V_G, E_G)$, ou plus simplement $G = (V, E)$ [1]. En règle générale, on note respectivement $n(G)$ et $m(G)$, ou plus simplement n et m [1], les tailles respectives des ensembles V_G et E_G. Pour une arête $e \in E_G$, on note $e = \{x, y\}$ lorsque cette arête relie deux sommets $x, y \in V_G$. On dit que x et y sont les *extrémités* de l'arête e, et on pourra noter $x \in e$ et $y \in e$. De plus, si l'arête e relie x à y, on dit que x et y sont *adjacents*, ou encore que x et y sont *voisins*. Dans ce cas encore, on dit que l'arête e est *incidente* au sommet x, et par symétrie que e est incidente au sommet y. De plus, on dit que deux arêtes $e_1, e_2 \in E_G$ sont *incidentes* si elles partagent une extrémité.
Pour un sommet $x \in V_G$, on note $N_G(x)$, ou $N(x)$ [1], son *voisinage ouvert* constitué de l'ensemble de ses voisins. On note $N_G[x]$, ou $N[x]$ [1], son voisinage *voisinage fermé* défini par $N_G[x] = N_G(x) \cup \{x\}$. De manière similaire, pour un sous-ensemble de sommets $X \subseteq V_G$, on note respectivement $N_G(X)$

1. Cette notation pourra être utilisée lorsqu'il n'y a pas d'ambiguïté sur le graphe G.

et $N_G[X]$, ou plus simplement $N(X)$ et $N[X]$ [1], les voisinages ouverts et fermés de X, où $N_G(X) = \cup_{x \in X} N_G(x) - X$, et $N_G[X] = N_G(X) \cup X$.
On appelle *degré* d'un sommet $x \in V_G$, noté $deg_G(x)$ ou plus simplement $deg(x)$ [1], le nombre de ses voisins, soit donc $deg_G(x) = |N_G(x)|$. On note respectivement $\delta(G)$ et $\Delta(G)$ les degrés minimum et maximum du graphe G, qui correspondent au plus petit et plus grand degré de ses sommets.

Un graphe $H = (V_H, E_H)$ est un *sous-graphe* de G si $V_H \subseteq V_G$ et $E_H \subseteq E_G$. Pour un sous-ensemble de sommets $U \subseteq V_G$, on note $G[U]$ le *sous-graphe induit* de G par U, composé des sommets de U et de toutes les arêtes qui relient les sommets de U. On dit aussi que U induit le sous-graphe $G[U]$ dans G. Le sous-graphe induit $G[U]$ est donc défini formellement par $G[U] = (U, \{\{x, y\} \in E_G \ : \ x, y \in U\})$. On dit en conséquence que le graphe H est un sous-graphe induit de G si H est un sous-graphe de G et si $H = G[V_H]$. De manière similaire, pour un sous-ensemble de sommets $U \subseteq V_G$, on construit $G - U$ le graphe obtenu à partir de G en supprimant tous les sommets dans U. Si $U = \{u\}$, nous pourrons écrire plus simplement $G - u$. Le graphe $G - U$ est donc défini formellement par $G - U = (V_G - U, \{\{x, y\} \in E_G \ : \ x, y \in \overline{U}\})$.
Une séquence $P = \{x_1, \ldots, x_p\}$, $p \geq 1$, est un *chemin* dans G si tous les éléments de la séquence sont des sommets distincts de G, et si tous les sommets sont reliés deux à deux dans l'ordre de la séquence. Soit donc formellement P est un chemin dans G si pour tout $1 \leq i < j \leq p \ : \ x_i \neq x_j$, si pour tout $1 \leq i \leq p \ : \ x_i \in V_G$ et si pour tout $1 \leq i \leq p-1 \ : \ \{x_i, x_{i+1}\} \in E_G$. On dit alors que P est un chemin qui *relie* le sommet x_1 au sommet x_p dans G. Pour deux sommets $x, y \in V_G$, on pourra noter $x \rightsquigarrow y$ un chemin $P_{x,y}$ qui relie x à y dans G, et si $z \in V_G$ est sur ce chemin, alors on pourra écrire $z \in P_{x,y}$. La longueur d'un chemin est le nombre d'arêtes qui le composent, soit donc P est ici un chemin de longueur $p-1$. On dit que P est un *chemin induit* si les seules arêtes que composent le sous-graphe induit $G[P]$ sont les arêtes du chemin, soit donc si $G[P] = (P, \cup_{i=1}^{i=p-1} \{x_i, x_{i+1}\})$. Un chemin P_1 est *plus court* qu'un chemin P_2 dans un graphe G si la longueur de P_1 est inférieure ou égale à la longueur de P_2. De manière similaire, P_1 est *plus long* que P_2 si sa longueur est supérieure ou égale à celle de P_2.
Une séquence $P = \{x_1, \ldots, x_p\}$, $p \geq 3$, est un *cycle* dans G si pour tout P est un chemin, et si x_1 et x_p sont voisins soit donc si $\{x_1, x_p\} \in E_G$. La *longueur* du cycle P est également définie par le nombre d'arêtes qui le composent, soit donc P est ici un cycle de longueur p. De même, P est un *cycle induit* si les seules arêtes que composent le sous-graphe induit $G[P]$ sont les arêtes du cycle.

Soient $x, y \in V_G$ deux sommets du graphe G. Un chemin $P_{x,y}$ est le plus court chemin entre x et y si tous les chemins qui relient x à y sont plus longs. On appelle *distance* entre x et y, ou de x à y dans G, la longueur du plus court chemin entre x et y s'il existe. Si aucun chemin ne relie x à y dans G, la distance de x à y vaut $+\infty$. La distance de x à y dans G est notée $d_G(x, y)$, ou $d(x, y)$ [1]. Notons que par définition de la longueur du chemin $P_{x,x} = \{x\}$ dans G, on a $d_G(x, x) = 0$. Soient $X_1, X_2 \subseteq V_G$ deux sous-ensembles de sommets, nous étendons la notion de distance dans G à ces sous-ensembles. Nous définissons donc $d_G(X_1, X_2)$, ou $d(X_1, X_2)$ [1], la distance de X_1 à X_2 dans G, comme la plus petite distance entre un sommet de X_1 et un sommet de X_2.
Le *diamètre* $diam(G)$ du graphe G est la plus longue distance entre deux sommets du graphe. Un sous-ensemble de sommets $X \subseteq V_G$ est *connexe* dans G si quelque soient $x, y \in X$, il existe un chemin qui relie x à y dans G. Autrement dit puisque G est fini, X est connexe si et seulement si le diamètre de $G[X]$ est fini. On appelle *composante connexe* d'un graphe G un sous-ensemble connexe $X \subseteq V_G$ de sommets et qui soit maximal au sens de la Section 1.1.

Un sous-ensemble $X \subseteq V_G$ de sommets est un *ensemble stable* de G si aucune arête de E_G ne relie deux sommets de X dans G. Autrement dit, X est un ensemble stable de G si et seulement si

$\forall\ x, y \in V_G\ :\ \{x, y\} \notin E_G$. Un sous-ensemble $X \subseteq V_G$ de sommets est une *clique* de G si tous les sommets de X sont reliés deux à deux. Formellement, X est une clique de G si et seulement si $\forall\ x, y \in V_G\ :\ \{x, y\} \in E_G$. Un sommet est dit *simplicial* si son voisinage induit une clique. Un sous-ensemble $X \subseteq V_G$ de sommets est un *ensemble dominant* de G si tous les autres sommets de V_G ont un voisin parmi X. Autrement dit, X est un ensemble dominant de G si et seulement si $\forall\ y \in \overline{X}\ :\ \exists\ x \in X$ tel que $\{x, y\} \in E_G$. Un sous-ensemble $X \subseteq V_G$ de sommets est une *couverture de sommets* de G si toutes les arêtes sont incidentes à un sommet de X. Autrement dit, X est une couverture de sommets si et seulement si $\forall\ e \in E_G\ :\ e \cap X \neq \emptyset$. Pour ces notions fondamentales de graphes, nous référons au sens de la Section 1.1 pour définir les stables, cliques, dominants ou couvertures de sommets maximaux, minimaux, maximums et minimums. Conformément aux standards de la Théorie des Graphes, on notera $\alpha(G)$ la taille d'un ensemble stable maximum et $\omega(G)$ la taille d'une clique maximum de G.
On appelle *coloration propre* d'un graphe $G = (V, E)$ une partition des sommets de V par la famille des ensembles stables de G, ou plus simplement une coloration de V en ensembles stables. Le *nombre chromatique* $\chi(G)$ de G est le plus petit nombre de couleurs d'une coloration propre de G.

Nous décrivons certaines opérations classiques sur les graphes. Soient $G = (V_G, E_G)$ et $H = (V_H, E_H)$ deux graphes. Le *graphe complémentaire* $\overline{G}$ de G est le graphe obtenu à partir de G en supprimant les arêtes existantes et en créant les arêtes qui n'existent pas. Formellement, on a donc $\overline{G} = (V_G, \{\{x, y\}\ :\ x, y \in V_G \text{ et } \{x, y\} \notin E_G\})$. L'*union disjointe* de G et H est le graphe $G \cup H$ défini par $G \cup H = (V_G \cup V_H, E_G \cup E_H)$. L'*union complète* de G et H est le graphe $G \star H$ défini par $G \star H = (V_G \cup V_H, E_G \cup E_H \cup \{\{x, y\}\ :\ x \in V_G, y \in V_H\})$. Dans la littérature, l'opération union complète peut également être appelé opération *join*.
La *suppression* d'un arête $e \in E_G$ dans le graphe G consiste à retirer e de E_G, autrement dit cette opération résulte en un graphe $G' = (V_G, E_G - e)$. La *suppression* d'un sommet dans G revient à supprimer de E_G ses arêtes incidentes et à retirer le sommet de V_G. Une *subdivision* d'arêtes est la création d'un sommet au milieu d'une arête, autrement dit le remplacement d'une arête par un chemin de longueur deux. Une *contraction* d'arêtes consiste à supprimer l'arête et fusionner ses extrémités, autrement dit pour une arête $e = \{x, y\}$ le voisinage de x s'ajoute au voisinage de y puis on supprime le sommet x.

L'isomorphisme de graphes est un outil de comparaison pour observer la ressemblance entre deux graphes. On dit que deux graphes $G = (V_G, E_G)$ et $H = (V_H, E_H)$ sont *isomorphes*, et on note $G \sim H$, s'il existe une bijection des sommets de V_G dans les sommets de V_H qui préserve les arêtes. Formellement, G et H sont isomorphes s'il existe une bijection $f : V_G \to V_H$ tel que $\forall\ x, y \in V_G\ :\ \{x, y\} \in E_G \Leftrightarrow \{f(x), f(y)\} \in E_H$.

1.3.2 Décompositions de Graphes

En Algorithmique, la méthode *diviser pour régner* est une méthode récursive qui résoud un problème de grande taille en résolvant deux ou plus sous-problèmes indépendants de plus petite taille, puis combine les solutions pour obtenir la solution du problème originel. La *programmation dynamique* est une autre méthode pour résoudre des problèmes informatiques en décomposant le problème originel en sous-problèmes. Une idée intuitive pour appliquer ces méthodes est de décomposer le graphe d'entrée du problème en sous-graphes, de résoudre le problème sur ces instances de plus petite taille, et de combiner les différentes solutions pour résoudre le problème sur le graphe originel. Dans un cas pratique, on souhaite de plus que les sous-graphes partagent les mêmes propriétés que le graphe originel.

Il y a différentes manières de décomposer un graphe. L'une des premières méthodes les plus employées consistait à séparer le graphe en plusieurs composantes connexes. Un sous-ensemble $X \subseteq V$ de sommets d'un graphe connexe $G = (V, E)$ est un *séparateur* de G si $G[V - X]$ n'est pas connexe. Pour appliquer la méthode *diviser pour régner*, l'objectif pour un graphe donné est de déterminer en temps linéaire ou polynomial un séparateur de plus petite taille possible, et dont les composantes connexes soient les plus équilibrées possibles. Un graphe $G = (V, E)$ est dit $s(n)$-*séparable* si G admet un séparateur de taille au plus $s(n)$ qui sépare le graphe en deux composantes connexes de taille au plus $2n/3$. Nous référons à l'article de Bodlaender pour une introduction plus détaillée sur les séparateurs [24].

Les décompositions arborescentes, en chemins, ou en branches d'un graphe constituent des décompositions désormais bien connues en Théorie des Graphes et en Algorithmique. Ces notions ont été introduites notamment par Robertson et Seymour en 1983 [170]. Les intérêts pratiques de ces notions sont importantes [9]. Depuis leur introduction, ces décompositions suscitent un grand intérêt en Théorie des Graphes et ont été largement étudiées [22, 24]. D'un point de vue Algorithmique Exacte Exponentielle ou Algorithmique Paramétrée, il existe de nombreux algorithmes qui utilisent ces décompositions [10, 166, 51]. En particulier, de nombreux problèmes NP-difficiles sont solubles à paramètre largeur arborescente fixée [31].

Décomposition arborescente. Une *décomposition arborescente* d'un graphe $G = (V, E)$ est un couple (X, T), où $T = (I, F)$ est un graphe arbre non orienté, et $X = \{X_i \subseteq V, i \in I\}$ est une famille de sous-ensembles de sommets de G. A chaque sommet $i \in I$ de l'arbre T est donc associé un unique sous-ensemble $X_i \subseteq V$ de sommets du graphe, sous-ensemble que l'on appelle *sac* de sommets. Dans la pratique, on pourra ne pas faire de distinction entre un sommet $i \in I$ de l'arbre T et le sac $X_i \in X$ qui lui est associé. Une décomposition arborescente (X, T) d'un graphe $G = (V, E)$ satisfait les propriétés suivantes :
◇ $\bigcup_{i \in I} X_i = V$
◇ $\forall\, e \in E \ : \ \exists\, i \in I$ tel que $e \subseteq X_i$
◇ $\forall\, i, j, k \in I$ tel que $j \in i \rightsquigarrow k$ dans $T \ : \ X_i \cap X_k \subseteq X_j$
Pour tout graphe $G = (V, E)$, un arbre T constitué d'un unique sommet isolé r associé au sac $X_r = V$ constitue une décomposition arborescente triviale. Tout graphe admet donc une décomposition arborescente. L'intérêt de ces décompositions pour l'Algorithmique est d'obtenir pour un graphe donné une décomposition dont les sacs associés sont de plus petite taille possible. On définit ainsi la *largeur* d'une décomposition arborescente (X, T) d'un graphe G par $l = \max_{i \in I}(X_i) - 1$. La *largeur arborescente* du graphe G, notée $\mathbf{tw}(G)$, est la plus petite largeur parmi les décompositions arborescentes possibles du graphe. ♣

Décomposition arborescente jolie. Une décomposition arborescente enracinée (X, T), avec $T = (I, F)$, d'un graphe $G = (V, E)$ est dite *jolie* si les sacs des feuilles de T ne possèdent qu'un sommet et si T ne possède que trois types de nœuds internes :
◇ $i \in I$ est un nœud *join* si i possède deux fils i_1 et i_2 tel que $X_i = X_{i_1} = X_{i_2}$
◇ $i \in I$ est un nœud *introduce* si i possède un unique fils j tel que $X_i = X_j + \{v\}$, avec $v \in V - X_j$
◇ $i \in I$ est un nœud *forget* si i possède un unique fils j tel que $X_i = X_j - \{v\}$, avec $v \in X_j - X_i$
Pour tout entier $k \geq 1$, il y a un algorithme en temps $O(n)$ pour rendre jolie une décomposition arborescente de largeur k qui préserve la largeur et composée d'au plus $4n$ nœuds [125]. ♣

Décomposition en chemins. Une *décomposition en chemins* d'un graphe $G = (V, E)$ est une

décomposition arborescente (X, T) de G pour laquelle T est un chemin. De manière similaire, puisqu'une décomposition arborescente triviale est une décomposition en chemin triviale, tout graphe admet une décomposition en chemin. On définit de la même façon la *largeur en chemin* d'un graphe G, encore appelée *largeur linéaire* de G, notée $\mathbf{pw}(G)$, comme la plus petite largeur parmi les décompositions en chemin possibles du graphe. Naturellement, tout graphe G vérifie $\mathbf{tw}(G) \leq \mathbf{pw}(G)$. On pourra noter $(X_1, \ldots, X_r)$ une décomposition en chemins. Une telle décomposition est dite *jolie* si c'est une décomposition arborescente jolie avec $|X_1| = |X_r| = 1$. Une décomposition en chemins jolie possède au plus $2n$ sommets. ♣

Décomposition en branches. Une *décomposition en branches* d'un graphe $G = (V, E)$ est un couple (σ, T), où $T = (I, F)$ est un arbre dont les sommets sont de degré $d = 1$ ou $d = 3$, et où σ est une bijection entre les sommets de V dans les feuilles de I. Pour une décomposition en branches (σ, T) d'un graphe $G = (V, E)$, retirer une arête $f \in F$ de l'arbre T le sépare en deux sous-abres T_1 et T_2. Ainsi chaque arête $f \in F$ de T sépare le graphe G en deux sous-graphes G_1 et G_2, chacun induits par les arêtes de E dont les feuilles associées par σ se situent respectivement dans T_1 et T_2. L'*ordre* d'une arête $f \in F$ de T est alors défini par le nombre de sommets de V partagés en commun par les deux sous-graphes G_1 et G_2, autrement dit le cardinal de l'intersection entre l'ensemble des extrémités des arêtes de G_1 et l'ensemble des extrémités des arêtes de G_2. La *largeur* d'une décomposition en branches (σ, T) d'un graphe G est le plus grand ordre des arêtes $f \in F$ de T. La *largeur de branche* de G, notée $\mathbf{bw}(G)$, est la plus petite largeur parmi les décompositions en branches possibles du graphe. ♣

Il existe d'autres décompositions de graphes bien connues, chacune associée à un paramètre telle que la *largeur de coupe* ou la *largeur de clique* que nous ne définissons pas ici. Déterminer la largeur arborescente, la largeur en chemin ou la largeur en branche d'un graphe sont des problèmes NP-difficiles [9]. Ces décompositions sont d'un intérêt capital en Algorithmique, et sont très étudiées en Théorie des Graphes. Nous listons ici certaines relations connues entre ces paramètres.

◇ Pour tout graphe G : $\mathbf{bw}(G) \leq \mathbf{tw}(G) + 1 \leq 3/2 \cdot \mathbf{bw}(G)$ [171]
◇ Si G est une forêt : $\mathbf{pw}(G) \leq 2/\log 3 \cdot \log n \leq 1.2619 \cdot \log n$ [127, Thm. 5]
◇ Pour tout graphe G : $\mathbf{pw}(G) \leq 2/\log 3 \cdot \log n \cdot \mathbf{tw}(G)$ [127, Thm. 6]
◇ Tout graphe G est $\mathbf{tw}(G)$-séparable [24, Thm. 20]
◇ Si G est $s(n)$-séparable, alors $\mathbf{pw}(G) = O(s(n) \cdot \log n)$ [24, Thm. 20]

1.3.3 Classes de Graphes

Une classe de graphes est une famille qui définit des graphes partageant les mêmes propriétés. En règle générale, une classe de graphes représente un ensemble infini de graphes, et pour les classes les plus simples, il existe des méthodes de construction de tous les graphes composant la famille. La catégorisation des graphes en classes est le fruit d'un immense travail de la Théorie des Graphes. L'intérêt de ces travaux pour notre spécialité est dans un premier temps d'identifier si un problème difficile dans le cas général l'est toujours si l'on restreint les graphes d'entrée à cette classe. Si le problème demeure difficile, alors ils nous est possible de construire des méthodes spécifiques pour les graphes qui constituent la classe étudiée, en tirant partie des propriétés structurelles que partagent ces graphes, toujours dans le but d'accélerer le plus possible les méthodes de résolution de nos problèmes. Ce travail de classification, de l'étude des propriétés et des relations entre les classes de graphes est toujours d'actualité et a certainement été initié avant l'invention de nos ordinateurs modernes que l'on peut situer vers 1938. Par exemple, en 1931 déjà, Dénes König prouvait son

théorème montrant l'égalité entre la taille maximum d'un couplage et la taille minimum d'une couverture de sommets dans un graphe biparti [126]. Cette section n'a pour objectif que d'illustrer quelques classes de graphes infinies très classiques et sur lesquelles nous serons amenés à travailler. Nous référons au livre *Graph Classes - A survey* de Brandstädt, Le et Spinrad pour une étude exhaustive des propriétés et des classes de graphes [27]. Nous référons également au livre *Algorithmic Graph Theory and Perfect Graphs* de Golumbic pour l'étude d'aspects algorithmiques de certaines classes de graphes [101].

Graphes Remarquables. Il existe des graphes bien connus et remarquables, au rang desquels on trouve le *triangle*, le complément du chemin à $n = 5$ sommets appelé graphe *house*, ou le graphe de *Pettersen*. Ces petits graphes qui portent un nom peuvent constituer, avec tous les graphes non isomorphes à peu de sommets, les premiers contre-exemples à certaines propriétés recherchées sur une classe de graphes. ♣

Chemins. Un graphe *chemin* est un graphe qui induit lui-même un chemin. Les chemins forment une des familles infinies les plus simples. Pour tout entier $n \in \mathbb{N}^*$, à un isomorphisme près, il existe un unique graphe chemin à n sommets, noté P_n. La reconnaissance de ces graphes est linéaire, et il y a très peu de problèmes NP-difficiles sur cette classe de graphes. Hormis les classes de propriété bornée, la plus petite classe de cette sous-section qui contienne les chemins est celle des arbres. ♣

Cycles. Un graphe *cycle* est un graphe qui induit lui-même un cycle. Cette famille est infinie, et de même que les chemins, pour tout entier $n \in \mathbb{N}_{\geq 3}$, et à un isomorphisme près, il existe un unique graphe cycle à n sommets note C_n. Hormis les classes de propriété bornée, la seule classe de cette sous-section qui contienne les cycles est celle des graphes planaires. ♣

Complets. Un graphe *complet* est un graphe qui induit lui-même une clique. En particulier, le complémentaire d'un graphe complet est une union disjointe de sommets isolés. Comme précédemment, cette famille est infinie et pour tout entier $n \in \mathbb{N}^*$, et à un isomorphisme près, il existe un unique graphe complet à n sommets note K_n. ♣

Bipartis. Un graphe *biparti* est un graphe dont les sommets peuvent être partitionnés en deux ensembles stables. La reconnaissance de tels graphes est linéaire, et certains problèmes deumeurent NP-difficiles sur cette classe de graphe, comme le problème du nombre b-chromatique. Une généralisation de cette classe de graphe est la famille des classes de graphes dont les sommets peuvent être partitionnés en k ensembles stables, pour $k \in \mathbb{N}_{\geq 2}$. Ces classes sont directement reliées au problème du nombre chromatique, puisque ce sont exactement les classes de graphes k-colorables. Décider si un graphe est k-colorable pour $k \geq 3$ est un problème NP-complet [94]. Le complémentaire d'un graphe biparti est appelé un graphe co-biparti. Un graphe *biparti complet* est un graphe résultant de l'union complète de deux stables. Les bipartis complets sont notés $K_{l,r}$ et sont uniques à un isomorphisme près pour tout $l \leq r \in \mathbb{N}^*$. De la même façon que précédemment, les bipartis complets peuvent se généraliser aux classes des graphes k-colorables complets. ♣

Etoiles. Bien qu'infinie, la classe des graphes *étoiles* représente une sous-classe de la classe des bipartis beaucoup plus petite, puisque les étoiles sont les bipartis complets de la forme $K_{1,r}$, avec $r \in N^*$. Pour tout $r \in N^*$, ces graphes sont par conséquent uniques à un isomorphisme près. Les bipartis complets, dont les étoiles sont une sous-classe, forment une sous-classe de la classe des cographes. Les étoiles sont en plus une sous-famille des arbres, et une sous-famille des splits. ♣

Splits. Un graphe *split* est un graphe dont les sommets peuvent être partitionnés en un ensemble stable et une clique. Le complémentaire d'un split est un split, et un graphe complet est un split particulier. La formule de Clark et Royle permet de déterminer le nombre de graphes splits non isomorphes à $n \in N^*$ sommets [176]. La classe des splits, ainsi que la classe de graphes dont le complémentaire est un split, constituent à la fois des sous-classes des cographes, et des sous-classes des graphes cordaux. ♣

Roues. Les graphes *roues* sont l'union complète d'un sommet isolé avec un cycle, ils sont donc de la forme $K_1 \star C_n,\ n \in \mathbb{N}_{\geq 3}$. Par conséquent, pour tout $n \in \mathbb{N}_{\geq 3}$, ces graphes sont également uniques à un isomorphisme près. De même que les cycles, hormis les classes de propriété bornée, la seule classe de cette sous-section qui contienne les roues est celle des graphes planaires. ♣

Arbres. Les graphes *arbres* sont les graphes qui ne contiennent pas de cycle. Ce sont exactement les graphes connexes satisfaisant $m = n - 1$. Les sommets d'un arbre sont également appelés ses *nœuds*. Un arbre non orienté peut se présenter sous une forme enracinée en choisissant arbitrairement un sommet du graphe pour *racine* de l'arbre. Ces arbres enracinés et l'arbre lui-même sont tous isomorphes. Pour un arbre enraciné satisfaisant $n \geq 2$, on distingue parmi les sommets du graphe qui ne sont pas la racine, les *feuilles* qui sont de degré $d = 1$, des autres sommets appelés *nœuds internes*. Dans un arbre, il existe toujours un unique chemin qui relie deux sommets. Dans un arbre enraciné, pour deux sommets x, y adjacents, on dira que x est le *père* de y si x appartient au chemin qui relie y à la racine. Dans ce cas, on dira de plus que y est un *fils* de x.
Il y a des algorithmes à délai constant pour construire tous les arbres non isomorphes à n sommets [162]. Cette classe contient les chemins et les étoiles. Les arbres ont des propriétés structurelles très intéressantes, et constituent en général une classe pour lesquelles les problèmes deviennent faciles. Cependant, comme nous le verrons dans cette thèse, il y a des problèmes pour lesquels les problèmes demeurent difficiles, comme le problème du nombre a-chromatique ou le problème d'ensemble tropical connexe. ♣

Forêts. Les *forêts* sont également des graphes sans cycle, et constituent une des plus simples classes définies par récurrence. Tout arbre est une forêt, et toute union disjointe de forêts est une forêt. Du fait que chaque composante connexe d'une forêt est un arbre, les problèmes sur cette classe sont en général de même difficulté que les problèmes sur les arbres. Les forêts constituent une sous-classe des graphes bipartis, une sous-classe des graphes planaires, et une sous-classe des graphes cordaux. ♣

Planaires. Les graphes *planaires* sont les graphes que l'on peut dessiner dans un plan sans qu'aucune arête ne se croise. Cette classe de graphes présente des propriétés structurelles très fortes pour l'Algorithmique. En particulier les graphes planaires ont un nombre d'arêtes linéaire, par équivalence degré moyen borné, puisqu'un graphe planaire à $n \geq 3$ sommets possède au plus $3n - 6$ arêtes [107]. En 1930 Kuratowski énonce le théorème selon lequel un graphe est planaire si et seulement s'il ne peut être obtenu à partir des graphes K_5 ou $K_{3,3}$ par une série de subdivisions d'arêtes [132]. Une conséquence directe de ce théorème est que tout graphe planaire G satisfait $\omega(G) \leq 4$. La reconnaissance de ces graphes est linéaire [115], et il existe au plus $2^{4.92n}$ graphes planaires non isomorphes à n sommets [25].
Bien que beaucoup de problèmes demeurent NP-difficiles sur cette classe de graphes, de nombreux algorithmes sous-exponentiels tirent partie de leurs propriétés structurelles [87, 64]. En 1979, Lipton et Tarjan énoncent le célèbre *Planar Separator Theorem* qui stipule que tout graphe *planaire* à n sommets est $\sqrt{8n}$-séparable [146], et Djidjev améliora ce résultat en 1982 pour montrer que

tout graphe planaire est $\sqrt{6n}$-séparable [60]. Ces théorèmes sont la pierre angulaire de la méthode *diviser pour régner* et a permis de construire de nombreux algorithmes sous-exponentiels pour ces graphes [5, 4].
Il existe beaucoup d'algorithmes pour résoudre des problèmes difficiles dont le temps d'exécution dépend fortement de la largeur arborescente du graphe. Fomin et Thilikos ont montré que la largeur arborescente d'un graphe planaire est bornée par $3.182\sqrt{n}$ [91]. Cependant, la complexité du problème consistant à calculer une décomposition arborescente optimale d'un graphe planaire demeure un problème ouvert, et il est en pratique difficile de déterminer une décomposition optimale [22]. Fomin et Thilikos ont montré malgré tout qu'un algorithme qui s'appuie une décomposition arborescente peut être transposé en un algorithme qui s'appuie sur décomposition en branche en respectant le temps d'exécution [87]. Ils ont montré que la largeur de branche d'un graphe planaire est bornée par $2.122\sqrt{n}$ [91], et que l'algorithme de Seymour et Thomas permet de calculer en temps $O(n^4)$ une décomposition en branche optimale d'un graphe planaire [177]. De ce fait, ils proposent une longue liste d'algorithmes sous-exponentiels plus efficaces pour résoudre des problèmes NP-difficiles sur les graphes planaires.
Dans cette Thèse, nous étudierons dans le chapitre 2 des problèmes de coloration sur les graphes. Les graphes planaires sont quant à eux très célèbres pour le fameux théorème des quatres couleurs [169]. Bien qu'un graphe planaire soit 4-colorable, déterminer le nombre chromatique d'un graphe planaire demeure NP-difficile [94]. Beigel et Eppstein proposent un algorithme exact exponentiel en temps $O^*(1.3289^n)$ pour décider si un graphe quelconque est 3-colorable [16]. Pour illustrer la technique de Fomin et Thilikos sur les graphes planaires, nous construirons dans le chapitre 2 un algorithme sous-exponentiel en temps $O^*(2^{5.044\sqrt{n}})$ pour calculer le nombre chromatique d'un tel graphe. ♣

Cordaux. Un graphe est *cordal* si tout cycle de plus de quatre sommets possède une *corde*, c'est-à-dire un graphe est cordal s'il ne possède aucun cycle induit de longueur au moins quatre. Cette propriété est héréditaire pour tous les sous-graphes induits. Ces graphes sont aussi appelés les graphes *triangulés*. Cette classe de graphes est une sous-famille des graphes d'intersection, et contient tous les graphes d'intervalles, que nous ne présentons pas ici. La classe des graphes *cordaux* présente certaines propriétés structurelles intéressantes pour l'ALGORITHMIQUE. En particulier, les graphes cordaux admettent au plus n cliques maximales. De plus, tout graphe cordal admet un sommet simplicial. Tout sous-graphe induit d'un cordal étant lui-même cordal, cette propriété est donc stable par récurrence sur les sous-graphes successifs obtenus par suppression d'un sommet simplicial. L'ordre dans lequel les sommets sont retirés s'appelle un *ordre d'élimination parfaite*. En 1965, Fulkerson et Gross ont montré que les graphes cordaux sont exactement les graphes qui admettent un *ordre d'élimination parfaite* [92]. Rose, Tarjan et Luecker ont proposé en 1976 un algorithme linéaire pour déterminer s'il existe un *ordre d'élimination parfaite* dans un graphe, et par conséquence la reconnaissance des graphes cordaux est linéaire [174]. Le fait que deux graphes soient cordaux ne rend pas plus facile le problème de décider s'ils sont isomorphes [196]. ♣

Cographes. Un *cographe* est un graphe qui ne possède pas de P_4 induit, et donc à fortiori, aucun chemin induit de longueur au moins quatre. Les cographes peuvent être définis par récurrence. K_1 est un cographe, et si G et H sont des cographes, alors $\overline{G}$ et $G \cup H$ sont aussi des cographes. On peut ainsi décomposer un cographe pour représenter dans un arbre ces opérations qui l'ont construit, où chaque nœud interne est l'opération complémentaire ou union disjointe, et les feuilles sont des sommets isolés. On peut de la même façon construire les cographes uniquement à partir des opérations union disjointe et union complète. K_1 est un cographe, et si G et H sont des cographes, alors $G \cup H$ et $G \star H$ sont aussi des cographes. Cette deuxième façon de procéder permet de représenter un cographe dans un arbre binaire que l'on appelle un *cotree*. Les cographes peuvent être reconnus en temps

linéaire [47]. La puissance algorithmique de leur décomposition est telle qu'il existe peu de problèmes NP-difficiles sur cette classe. Cependant, ces graphes peuvent être étudiés par l'Algorithmique Exacte Exponentielle pour des problèmes d'énumération d'objets combinatoires dont le nombre est exponentiel [52]. ♣

Parfaits. Un graphe est *parfait* s'il satisfait $\omega(G) = \chi(G)$. Un graphe est parfait si et seulement si son complémentaire est parfait. Cette propriété s'appelle le *théorème faible* des graphes parfaits [148]. Le *théorème fort* stipule qu'un graphe est parfait si et seulement si ni lui, ni son complémentaire, ne contiennent de cycle induit de longueur impaire plus grande que cinq. Ce théorème était une conjecture de Berge datant de 1961, et n'a été prouvé qu'en 2006 [40]. Les graphes bipartis, cordaux, les cographes, ou donc leur complémentaire, appartiennent tous à cette famille. Tous les graphes planaires ne sont pas parfaits, mais on peut le décider en temps polynomial [116]. ♣

Propriété Bornée. On peut regrouper les graphes selon une certaine propriété qui présente une borne commune. Par exemple, les classes de graphes dont le degré moyen est inférieur à $\overline{d} \in \mathbb{N}_{\geq 6}$, et les classes de graphes dont la taille de clique maximum est bornée par $\overline{w} \in \mathbb{N}_{\geq 5}$, contiennent chacune l'ensemble des graphes planaires.
Le genre d'une surface topologique est une notion mathématique complexe que nous ne décrivons pas ici. En Théorie des Graphes, le *genre* d'un graphe est le plus petit genre d'une surface topologique sur laquelle on puisse dessiner le graphe sans qu'aucune arête ne se croise. Déterminer le genre d'un graphe est un problème NP-difficile [181], mais solvable à paramètre fixe $g \in N$, et déterminer le genre d'un graphe de largeur arborescente bornée peut se faire en temps linéaire [123]. Les classes de graphes dont le genre est borné par $\overline{g} \in \mathbb{N}$ possèdent de bonnes qualités structurelles pour l'Algorithmique. Les graphes planaires sont exactement les graphes qui peuvent être dessinés sur une surface de genre $g = 0$ comme un disque. Toutes les classes de graphes dont le genre est borné par $\overline{g} \in \mathbb{N}$ contiennent donc les graphes planaires, et en constituent une généralisation. Il existe une borne asymptotique pour le nombre de graphes non isomorphes à n sommets et de genre g [34], et on peut décider en temps polynomial si deux graphes de même genre sont isomorphes [155]. Les classes de graphes dont le genre est borné présentent des propriétés structurelles similaires à celles des graphes planaires. Par la formule d'Euler, tout graphe de genre g contient au plus $3n-6+6g$ arêtes [181]. Gilbert, Hutchinson et Tarjan ont montré en 1984 que tout graphe de genre g admet un séparateur de taille au plus $6\sqrt{gn}+2\sqrt{2n}+1$ sommets qui sépare le graphe en composantes connexes de taille au plus $2n/3$ [98]. Dorn, Fomin et Thilikos proposent de la même manière que pour les graphes planaires des algorithmes sous-exponentiels pour résoudre des problèmes NP-difficiles sur les graphes de genre borné [63].
Pour cette thèse, nous aurons besoin de la meilleure borne possible de la largeur arborescente d'un graphe de genre fixé $g \in \mathbb{N}$. Pour établir le temps d'exécution de leurs algorithmes, Fomin et Thilikos [88] montrent le lien entre la largeur de branche et le genre d'un graphe d'après les résultats de Djidjev et Venkatesan [61]. Ainsi donc, la largeur de branche $\mathbf{bw}(G)$ d'un graphe G est borné par $\mathbf{bw}(G) \leq (2.13 + 2.83\sqrt{eg(g)}) \cdot \sqrt{n}$, où $eg(g)$ représente le genre eulérien du graphe. Pour un graphe G de genre g, son genre eulérien $eg(g)$ est borné par $eg(g) \leq 2g$ [61], et Robertson et Seymour ont montré en 1991 que pour tout graphe G, on a $\mathbf{tw}(G) \leq 1.5 \cdot \mathbf{bw}(G)$ [171]. Ainsi donc, pour un graphe de genre g, nous pouvons établir la relation $\mathbf{tw}(G) \leq (3.19 + 6\sqrt{g}) \cdot \sqrt{n}$. ♣

1.3.4 Problèmes de Graphes

Dans cette Sous-Section, nous présentons des résultats algorithmiques sur des problèmes de graphes parmi les plus célèbres et les plus étudiés. Beaucoup de notions issues de la Théorie des Graphes

qui caractérisent des propriétés sur les sous-ensembles de sommets ou d'arêtes sont accompagnées de familles de problèmes. Les notions de maximal, minimal, maximum et minimum au sens de la Section 1.1 constituent chacunes des problèmes différents autour de la même notion.

Ensemble Stable. Tout graphe admet un ensemble stable. L'ensemble vide est l'unique stable minimum et minimal du graphe. En observant qu'un ensemble stable est une clique dans le graphe complémentaire, et que le complémentaire d'un graphe peut être obtenu en temps polynomial, les problèmes et résultats ci-dessous sur les stables peuvent être directement traduits aux problèmes de cliques. Egalement, on observera que le complémentaire d'un stable dans un graphe est une couverture de sommets dans ce même graphe. Par conséquent, hormis les résultats d'Algorithmique Paramétrée, la plupart des résultats sur les stables maximaux et maximums peuvent être directement traduits aux problèmes de couvertures de sommets minimales et minimums d'un graphe.

Ensemble Stable Maximum
Version : *Optimisation*

Entrée : $G = (V, E)$
Sortie : Déterminer un ensemble stable $X \subseteq V$ maximum de G

Couverture de Sommets Minimum
Version : *Paramétrée*

Entrée : $G = (V, E)$
Paramètre : $k \in \mathbb{N}$
Sortie : Décider si G admet une couverture de sommets de taille k

De même que le problème de clique maximum, ces problèmes sont NP-difficiles, et il y a un algorithme exact exponentiel en temps $O^*(1.2109^n)$ pour les résoudre [172]. Pour un paramètre $k \in \mathbb{N}$, il y a un algorithme en temps $O^*(1.2738^k)$ pour les graphes en général [37], et un algorithme en temps sous-exponentiel $O^*(2^{O(\sqrt{k})})$ pour les graphes de genre borné [57], pour décider si un graphe admet une couverture de sommets de taille $k \in \mathbb{N}$. Sous l'hypothèse **ETH**, ces deux algorithmes sont de plus optimaux, dans le sens où il n'y a pas d'algorithme pour ce problème respectivement en temps $O^*(2^{o(k)})$ sur les graphes en général et $O^*(2^{o(\sqrt{k})})$ sur les graphes planaires [28]. Cependant, le fait que le problème couverture de sommets de paramètre taille de solution $k \in \mathbb{N}$ soit FPT n'implique pas que le problème ensemble stable de même paramètre le soit également. En effet, avec ce paramètre le problème ensemble stable n'est pas FPT mais est dans **W**[1] [65].

Ensemble Stable Maximal
Version : *Enumération*

Entrée : $G = (V, E)$
Sortie : Enumérer les ensembles stables maximaux de G

Les stables maximaux sont des objets importants pour des algorithmes de coloration de graphe, il y a un intérêt certain à les énumérer. Déterminer un stable maximal d'un graphe est un problème facile. Cependant, énumérer les stables maximaux, les cliques maximales, ou les couvertures de sommets minimales d'un graphe prend au pire des cas un temps nécessairement exponentiel. En

effet, une union disjointe de triangles admet $3^{n/3}$ stables maximaux. Toutefois, Moon et Moser ont montré en 1965 que cette borne est la plus grande possible [159]. L'algorithme de Tuskiyama et al. est un algorithme à délai polynomial pour énumérer les cliques maximales d'un graphe [182]. Cet algorithme est donc algorithme d'énumération *optimal* en temps $O^*(3^{n/3})$ pour énumérer ces objets combinatoires dans un graphe. Pour une version de dénombrement du problème, il y a un algorithme en temps $O^*(1.3642^n)$ pour dénombrer les stables maximaux d'un graphe [97]. ♣

Ensemble Dominant. Pour tout graphe $G = (V, E)$, V est l'unique ensemble dominant maximum et maximal.

> ENSEMBLE DOMINANT MINIMUM
> *VERSION* : *OPTIMISATION*
>
> **Entrée** : $G = (V, E)$
> **Sortie** : Déterminer un ensemble dominant $X \subseteq V$ minimum de G

Déterminer un dominant minimal d'un graphe est un problème facile, mais le problème ENSEMBLE DOMINANT MINIMUM est un problème NP-difficile. Il y a un algorithme en temps $O^*(1.4969^n)$ pour déterminer un ensemble dominant minimum d'un graphe [184], et un algorithme en temps $O^*(1.5048^n)$ pour compter tous les dominants minimums [185]. D'un point de vue complexité paramétrée, pour un paramètre $k \in \mathbb{N}$, déterminer si un graphe admet un dominant de taille au plus k est un problème W[2]-difficile [65].
Pour l'ALGORITHMIQUE, énumérer les dominants minimaux dans un graphe est également un problème d'une grande importance. Contrairement aux ensembles stables, il n'y a pas de borne supérieure optimale connue pour le nombre de dominants minimaux dans un graphe. On sait qu'il existe un cograph avec au moins 1.5704^n dominants minimaux, et un graphe admet au plus $O^*(1.7159^n)$ dominants minimaux. Cette borne supérieure est celle donnée par le meilleur algorithme actuel pour énumérer les dominants minimaux d'un graphe [89]. Cette question concernant l'ajustement de cette borne a été longuement étudiée, notamment concernant certaines classes de graphes [52]. Cependant, bien que l'on ne dispose pas à l'heure actuelle d'algorithme optimal pour l'énumération des dominants minimaux, il est intéressant d'observer qu'à l'instar du problème ENSEMBLE STABLE, il y a un algorithme d'optimisation qui est plus performant pour déterminer un dominant minimum que celui qui consiste à énumérer, même de façon optimale, tous les dominants minimaux. ♣

Couverture d'Ensembles. Le problème de COUVERTURE D'ENSEMBLES est un problème important pour l'ALGORITHMIQUE EXACTE EXPONENTIELLE et ALGORITHMIQUE PARAMÉTRÉE. On pourra par exemple transformer une instance d'un problème en une instance du problème COUVERTURE D'ENSEMBLES. Il est donc important d'avoir à disposition les meilleurs algorithmes pour résoudre ce problème.

> COUVERTURE D'ENSEMBLES
> *VERSION* : *OPTIMISATION*
>
> **Entrée** : Un univers $\mathcal{U}$, une famille $\mathcal{S}$ de l'univers
> **Sortie** : Déterminer une sous-famille $\mathcal{S}^*$ de $\mathcal{S}$ de taille minimum qui forme une couverture de $\mathcal{U}$

Ce problème est NP-difficile. Le problème ENSEMBLE DOMINANT MINIMUM est un cas particulier du problème COUVERTURE D'ENSEMBLES, où l'univers $\mathcal{U}$ représente les sommets du graphe, et

la famille $\mathcal{S}$ est constituée des voisinages fermés des sommets dans le graphe. Ainsi, le meilleur algorithme pour le problème Ensemble Dominant Minimum était auparavant l'algorithme en temps $O^*(1.2355^{|\mathcal{S}|+|\mathcal{U}|})$, qui est le meilleur algorithme actuel pour résoudre la version optimisation du problème Couverture d'Ensembles [85]. Ce problème peut en plus être directement traduit en un problème d'hypergraphe. Ce problème est FPT de paramètre $k = |\mathcal{S}^*|$ taille de la sous-famille, et peut être résolu en temps $O^*(1.2738^k)$ [37]. ♣

Nombre Chromatique. Tout graphe admet une coloration propre de taille $|V|$. L'objectif du nombre chromatique est de déterminer une coloration propre de taille minimale.

> Nombre Chromatique
> *Version* : *Optimisation*
>
> **Entrée** : $G = (V, E)$
> **Sortie** : Déterminer une coloration propre de G de plus petite taille possible

Ce problème est un problème NP-difficile. Pendant longtemps, le meilleur algorithme connu pour résoudre ce problème était l'algorithme de programmation dynamique de Lawler [136] en temps $O^*(2.4423^n)$ publié en 1976, en s'appuyant sur les résultats sur l'énumération des stables maximaux d'un graphe. Björklund, Husfeldt, et Koivisto ont montré en 2009 que ce problème, parmi d'autres problèmes de coloration comme le problème du Nombre Domatique, peut être résolu en temps $O^*(2^n)$ grâce au principe d'inclusion-exclusion [20]. Une question ouverte est alors de savoir si un tel résultat peut être amélioré. L'algorithme de Björklund, Husfeldt, et Koivisto résout en temps $O^*(2^n)$ la version dénombrement du problème Partition d'Ensembles suivant :

> Partition d'Ensembles
> *Version* : *Dénombrement*
>
> **Entrée** : Un univers $\mathcal{U}$, une famille $\mathcal{F} \subseteq 2^{\mathcal{U}}$ de sous-ensembles de l'univers, et un entier $k \in \mathbb{N}^*$
> **Sortie** : Déterminer le nombre de k-partitions de $\mathcal{U}$ par $\mathcal{F}$

Un résultat négatif pour améliorer l'algorithme de Björklund, Husfeldt, et Koivisto, est qu'il a été montré en 2012 que sous l'hypothèse de complexité **SETH**, les algorithmes connus de nombreux problèmes ne peuvent être améliorés [53]. En particulier, il est conjecturé que sous l'hypothèse **SETH**, on ne peut résoudre la version décisionnelle du problème Partition d'Ensembles en un temps meilleur que $O^*(2^n)$.
Décider la 2-coloration d'un graphe est un problème facile, il est le même que celui de décider si un graphe est biparti. Pour décider la 3-coloration ou la 4-coloration propre d'un graphe, il existe des algorithmes en temps respectifs $O^*(1.3289^n)$ et $O^*(1.7272^n)$ [16, 83]. Il y a de plus des algorithmes en temps $O^*(1.6262^n)$ et $O^*(1.9464^n)$ pour calculer respectivement le nombre de 3-colorations et de 4-colorations propres qu'admet un graphe [83].
Il est difficile d'obtenir des algorithmes paramétrés pour des problèmes de coloration. En effet, pour tout $k \in \mathbb{N}_{\geq 3}$, décider si un graphe admet une coloration propre de taille k est un problème NP-complet [94]. ♣

1.4 Notions d'hypergraphes

Dans cette Section, nous reprenons en majeure partie les notations du livre de Berge [17].

Un *hypergraphe* $\mathcal{H} = \{X_1, \ldots, X_q\}$ est une famille de sous-ensembles d'un ensemble fini d'objets. Dans notre cas, et pour permettre des transitions avec la Théorie des Graphes, on notera $V_{\mathcal{H}}$ cet ensemble d'objets, que nous appellerons *sommets*. S'il n'y a pas d'ambiguité, on pourra noter $V_{\mathcal{H}}$ plus simplement par V. Les éléments $X_i \subseteq V$ de la famille $\mathcal{H}$ sont appelés les *hyperarêtes* de $\mathcal{H}$. Nous avons par conséquent $\mathcal{H} \subseteq 2^V$, et on pourra noter $e \in \mathcal{H}$ une hyperarête de l'hypergraphe $\mathcal{H}$. Les hypergraphes constituent une généralisation naturelle des graphes à des arêtes de plus grande taille, où les arêtes d'un graphe modélisent des hyperarêtes de deux sommets. Tout graphe est donc en particulier un hypergraphe.
Le *rang* $r(\mathcal{H})$ d'un hypergraphe $\mathcal{H}$ est la taille maximum de ses hyperarêtes. On a donc $r(\mathcal{H}) = \max_{e \in \mathcal{H}} |e|$. La taille d'une hyperarête pourra également être appelée le rang de l'hyperarête. Un hypergraphe est dit *uniforme* si toutes ses hyperarêtes ont le même rang, et il est dit *simple* s'il n'a pas d'hyperarêtes multiples. Si aucune hyperarête n'est contenue dans une autre, l'hypergraphe est dit appartenant à la *Famille de Sperner*.
Sans précision particulière on notera pour un hypergraphe $\mathcal{H}$, de même que pour un graphe, n le nombre de sommets et m le nombre d'hyperarêtes de l'hypergraphe.
Soit $\mathcal{H}$ un hypergraphe. On appelle *degré* d'un sommet $x \in V$, noté $d_{\mathcal{H}}(x)$, le nombre d'hyperarêtes qui contiennent x. Dans la littérature, le degré d'un sommet dans un hypergraphe pourra également être appelé sa *fréquence*. Cette notion s'étend à tout sous-ensemble de sommets. Le degré $d_{\mathcal{H}}(X)$ d'un sous-ensemble $X \subseteq V$ est donné par $d_{\mathcal{H}}(X) = |\{e \in \mathcal{H} \ : \ e \cap X \neq \emptyset\}|$. Pour un sommet $x \in V$, on notera $\mathcal{H}_x$ le sous-hypergraphe induit de $\mathcal{H}$, formé par les hyperarêtes de $\mathcal{H}$ qui contiennent x. De même, pour un sous-ensemble $X \subseteq V$ de sommet de $\mathcal{H}$, on notera $\mathcal{H}_X = \{e \in \mathcal{H} \ : \ e \cap X \neq \emptyset\}$.
Un sous-ensemble $X \subseteq V$ est un *ensemble stable* de $\mathcal{H}$ si X ne contient entièrement aucune hyperarête, c'est-à-dire X est stable si $\forall\, e \in \mathcal{H} \ : \ e - X \neq \emptyset$. Un sous-ensemble $X \subseteq V$ est un *transversal* de $\mathcal{H}$ si X possède une intersection non vide avec chaque hyperarête, soit donc X est un transversal si $\forall\, e \in \mathcal{H} \ : \ e \cap X \neq \emptyset$, ou encore $\mathcal{H}_X = \mathcal{H}$. Les notions de transversaux minimums et minimaux sont définies au sens de la Section 1.1. L'ensemble des transversaux minimaux de $\mathcal{H}$, noté $Tr(\mathcal{H})$, constitue un hypergraphe sur le même ensemble de sommets V.

Non seulement les graphes et les hypergraphes sont reliés car les seconds sont une généralisation des premiers, mais aussi parce que des notions ou des problèmes de graphes se traduisent en notions ou en problèmes d'hypergraphe. Par exemple, un stable dans un graphe G est aussi un stable au sens des hypergraphes, lorsque l'on considère G comme un hypergraphe $\mathcal{G}$. De même, une couverture de sommets dans G est un transversal dans $\mathcal{G}$. Nous avons vu que le problème Couverture d'Ensembles recouvre beaucoup de problèmes informatiques, et peut directement être traduit en problème d'hypergraphe. En particulier, nous avons vu que le problème Ensemble Dominant est un de ces problèmes qui peut être traduit par un problème d'hypergraphe.
Les hypergraphes peuvent être classés selon leur rang. Dans cette Thèse, nous nous intéresserons en grande partie au problème d'énumération des transversaux minimaux d'un hypergraphe de rang $k \in \mathbb{N}_{\geq 2}$, formulé de la façon suivante :

> k-Hitting Set
> *Version* : *Enumération*
>
> **Entrée** : $\mathcal{H}$ un hypergraphe de rang $k \in \mathbb{N}_{\geq 2}$
> **Sortie** : Enumérer les transversaux minimaux $Tr(\mathcal{H})$ de $\mathcal{H}$

La version énumération du problème Hitting Set est connue dans la littérature sous de nombreuses formes, comme par exemple celui de *dualisation d'hypergraphe*, traduit de l'anglais *Duali-*

zing Hypergraphs. Nous nous intéresserons également à d'autres versions du problème k-Hitting Set, notamment la version optimisation qui consiste à déterminer un transversal minimum d'un hypergraphe $\mathcal{H}$. Nous étudierons de plus le problème co$k$-Hitting Set suivant :

co$k$-Hitting Set
Version : *Enumération*

Entrée : $\mathcal{H}$ un hypergraphe de rang $k \in \mathbb{N}_{\geq 2}$
Sortie : Enumérer les transversaux minimaux de $\mathcal{H}$ dont le complémentaire dans V est un transversal de $\mathcal{H}$

Le problème d'énumération des transversaux minimaux dans un hypergraphe quelconque est appelé le problème Hitting Set. Cependant, autant nous pouvons espérer obtenir des temps d'exécution meilleurs que $O^*(2^n)$ pour résoudre le problème k-Hitting Set, autant sous l'hypothèse **SETH**, aucun algorithme plus rapide que $O^*(2^n)$ ne peut exister pour résoudre la version optimisation du problème Hitting Set [53].

1.5 Techniques Algorithmiques

Nous présentons dans cette Section des techniques fondamentales en algorithmique exacte exponentielle pour construire des algorithmes efficaces pour la résolution de problèmes difficiles. Nous référons au livre de Fomin et Kratsch pour des détails plus avancés sur ces méthodes [90]. Nous référons également à l'article de Björklund, Husfeldt, et Koivisto pour plus de détails encore sur la mise en œuvre de la technique d'*Inclusion-Exclusion* [20].

1.5.1 Brute-Force

Brute-Force. La technique *Brute-Force*, aussi appelée *recherche exhaustive*, n'est pas une méthode pour obtenir des algorithmes efficaces, mais fournit toujours un algorithme de résolution d'un problème. Comme son nom l'indique, cette méthode résout un problème en testant toutes les possibilités de réponses possibles, jusqu'à trouver une solution pour une version de décision ou d'existence, ou jusqu'à avoir parcouru l'espace de toutes les solutions possibles pour les autres versions. Un algorithme obtenu par cette méthode est appelé un algorithme *trivial*. L'intérêt pratique de cette méthode est d'étudier la possibilité pour une autre méthode d'obtenir un meilleur temps d'exécution. Le temps d'exécution d'un algorithme trivial constitue alors un challenge à relever. Pour un univers à n éléments, le temps d'exécution d'un algorithme trivial varie selon le type de problème étudié. Pour un problème de sous-ensembles, à supposer que vérifier la validité d'une propriété recherchée s'exécute en temps polynomial, le temps d'exécution d'un algorithme *Brute-Force* sera en général $O^*(2^n)$. Pour un problème d'ordonnancement, l'algorithme trivial teste toutes les permutations possibles de $\mathfrak{S}_n$ en temps $O^*(n!) = O^*(2^{n \log n})$. Pour un problème de partitionnement, l'algorithme trivial teste toutes les partitions possibles des éléments en temps $O^*(n^n) = O^*(2^{n \log n})$. Il y a des problèmes pour lesquels on ne connaît pas d'algorithme de résolution qui soit meilleur algorithme que l'algorithme trivial. Par exemple, sous l'hypothèse de complexité **SETH**, il n'existe pas, pour résoudre la version décisionnelle du problème SAT ou pour résoudre la version optimisation du problème Hitting Set, d'algorithme qui soit meilleur que l'algorithme trivial en temps $O^*(2^n)$ [53]. ♣

Au-delà de la méthode *Brute-Force*, il existe deux grandes catégories de techniques, qui ensuite possèdent des ramifications. Ces deux grandes techniques sont la technique de *Branchement* et la technique de *Programmation Dynamique*. Une grande différence entre ces deux techniques est que la première est une programmation par récurrence, tandis que la seconde ne l'est pas. Dans un modèle de calcul pourvu d'une machine ***RAM***, cette différence a deux conséquences. La première conséquence est que la pile d'appels du programme qui exécute l'algorithme peut être de taille constante pour un algorithme de programmation dynamique, tandis que pour un algorithme de branchement la pile d'appels peut avoir une taille linéaire en la taille de l'entrée. La deuxième conséquence se situe au niveau de l'espace mémoire des variables. Pour un algorithme de branchement, le programme n'a besoin de stocker qu'un très faible ensemble de solutions, tandis que pour un algorithme de programmation dynamique, l'algorithme a besoin en général pour résoudre un problème difficile de stocker un nombre exponentiel de solutions. Une deuxième grande différence entre ces deux techniques se situe au niveau du type de problème que peut résoudre efficacement la technique. En général, on utilisera la technique de *Branchement* pour des problèmes de type propriété de sous-ensembles sur un univers, tandis que l'on utilisera la *Programmation Dynamique* pour des problèmes de partionnement ou d'ordonnancement. Cette deuxième différence a un impact sur le temps d'exécution espéré des algorithmes. On attend de chacun d'eux qu'ils soient plus efficaces que l'algorithme trivial. En conséquence, sur une instance de taille n, on attend en général d'un algorithme de branchement qu'il soit en temps $O^*((2-\epsilon)^n)$, pour $0 < \epsilon < 1$, tandis qu'on attend d'un algorithme de programmation dynamique qu'il soit en temps $O^*(c^n)$, pour $c \in \mathbb{R}_{>1}$, avec en général $c \geq 2$.

1.5.2 Branchement

Branchement. La technique de *Branchement* est appelée de différentes manières dans la littérature, comme *branch and bound* traduit *séparation et évaluation*, *branch and reduce* traduit *brancher et réduire*, *pruning the search tree* traduit *élagage de l'arbre de recherche*, *backtracking*, ou simplement comme nous l'appelons *branching* traduit *branchement*. La technique de *Branchement* est une technique de résolution qui décompose un problème en sous-problèmes que l'on résout de manière récurvive, et dont on combine les solutions pour obtenir la solution du problème originel. En Algorithmique Exacte Exponentielle, cette décomposition a lieu en général au niveau de l'instance du problème, que l'on décompose ainsi en instances de plus petite taille.
Pour résoudre un problème sur une instance d'entrée, un algorithme de branchement utilise un ensemble fini de *règles* qui sont appliquées dans l'ordre bien précis dans lequel elles sont édictées, afin d'obtenir des instances de plus petite taille sur lesquelles appliquer l'algorithme récursivement. Chacune des règles de l'algorithme dispose de certaines conditions sur les instances du problème pour lesquelles elles peuvent s'appliquer, conditions qui sont en général d'ordre structurel.
Au *départ*, l'algorithme tente donc d'appliquer la première règle édictée. Si la condition à une règle échoue, l'algorithme tente d'appliquer la règle suivante, jusqu'à appliquer la dernière règle de l'algorithme. Lorsqu'une instance échoue aux conditions d'application d'une règle, elle renseigne en général de certaines de ses propriétés structurelles, dont pourront tirer profit les règles suivantes de l'algorithme. Pour pouvoir être *valide*, c'est en toute logique que la dernière règle d'un algorithme de branchement doit pouvoir être appliquée sur toutes les instances qui ont échoué aux conditions des règles précédentes. Dès qu'une règle réussit, l'algorithme décompose le problème en un nombre fini de sous-problèmes, dont la taille doit être plus petite que celle du problème étudié. Sur chacune des instances obtenues et dans l'ordre dans lequel elles ont été créées, l'algorithme est alors appliqué de manière récursive, c'est-à-dire qu'il reprend depuis le *départ*. Il est finalement à préciser que les conditions des règles doivent en règle général pouvoir être testées en temps polynomial en la taille de l'instance, et la création d'une instance de sous-problème doit prendre un temps polynomial.

Les règles d'un algorithme de branchement sont de deux types, les règles de réduction et les règles de branchement. Une *règle de réduction* permet, soit de stopper la décomposition récursive en sous-problèmes en résolvant en temps polynomial l'instance considérée, ou alors de simplifier l'instance considérée et d'obtenir une unique instance de sous-problème de plus petite taille. Une *règle de branchement* permet de décomposer l'instance considérée et d'obtenir plusieurs instances de sous-problèmes de plus petite taille. ♣

L'exécution d'un algorithme de branchement décrit naturellement un arbre, appelé *arbre de recherche*. À chaque instance de sous-problème créée par l'algorithme correspond un sommet de l'arbre. Le sommet associé dans l'arbre est une feuille si l'algorithme résout l'instance par une règle de réduction sans créer de sous-problème, et correspond à un nœud interne dans le cas contraire. Dans ce dernier cas, les nœuds de l'arbre correspondant aux instances de sous-problèmes, créées par l'algorithme pour résoudre l'instance considérée, sont reliés par une arête pour représenter ce qui constitue un lien de parenté. De cette manière, l'arbre de recherche engendré par l'exécution d'un algorithme de branchement peut être considéré indifféremment orienté ou non orienté. Le temps d'exécution d'un algorithme de branchement correspond alors, à un facteur polynomial près, au plus grand nombre de nœuds d'un arbre de recherche parmi toutes les exécutions sur toutes les entrées possibles de l'algorithme. Un calcul direct du temps d'exécution d'un algorithme de branchement est en général très compliqué. Cependant, les méthodes dont nous disposons nous permettent d'en fournir une borne supérieure.
Soit $\mathcal{A}$ un algorihme de branchement sur une entrée de taille n. Supposons que pour cette entrée l'algorithme applique la j-ème règle de branchement, et créé $m_j \in \mathbb{N}_{\geq 2}$ instances de sous-problèmes, chacune de taille respective $n - a_i$, pour $a_i \in \mathbb{R}_+^*$, $1 \leq i \leq m_j$. Le temps d'exécution $T(n)$ de l'algorithme $\mathcal{A}$ satisfait alors naturellement la récurrence $T(n) \leq \sum_i T(n - a_i)$. Les solutions de cette récurrence linéaire homogène sont de la forme α^n, avec α un nombre complexe. La valeur α_j qui nous intéresse est l'unique réel positif racine du polynôme $P_j(X) = 1 - \sum_i X^{-a_i}$. Le réel α_j et le vecteur $(a_1, \ldots, a_{m_j}) \in (\mathbb{R}_+^*)^{m_j}$ sont alors appelés respectivement le *nombre de branchement* et le *vecteur de branchement* de la j-ème règle de branchement. Une borne supérieure sur le temps d'exécution $T(n)$ de l'algorithme de branchement $\mathcal{A}$ est alors $O^*(\alpha^n)$, et on dit que $\mathcal{A}$ s'exécute en temps $O^*(\alpha^n)$, où $\alpha = \max_j \alpha_j$ est le plus grand nombre de branchement des règles que composent l'algorithme.

Fondamentaux. Dans cette Thèse, nous nous sommes longuement intéressés aux proriétés des vecteurs de branchement, dans le but de pouvoir les manipuler, et afin de pouvoir construire des algorithmes exacts exponentiels toujours plus efficaces. Certaines propriétés de ces outils remarquables sont bien connues, tandis qu'un très grand nombre d'entre elles restent encore à découvrir. La Section 3.2 est entièrement dédiée à cette étude. Nous présentons dans cette sous-section uniquement la Proposition 1.1, qui est fondamentale pour tout algorithme qui utilise la technique de *Branchement*. En particulier, cette proposition montre que tout vecteur de branchement admet bien un nombre de branchement, que ce dernier est unique, et qu'il satisfait certaines propriétés. Nous donnons une preuve de cette proposition en Section 3.2.

Un vecteur de branchement est un vecteur $A \in (\mathbb{R}_+^*)^{m_A}$ de réels strictement positifs, avec $m_A \in \mathbb{N}_{\geq 2}$. Pour un vecteur $A \in (\mathbb{R}_+^*)^{m_A}$, on note $A = (a_1, \ldots, a_{m_A})$ ce vecteur, et on appelle $m_A \in \mathbb{N}^*$ sa *dimension*. Si $m_A \geq 2$, on note $\tau(A) = \alpha_A$ le *nombre de branchement* de A, défini comme l'unique racine réelle positive du polynôme $P_A(X) = 1 - \sum_{i=1}^{i=m_A} X^{-a_i}$. La notation $\tau(A)$ d'un nombre de branchement peut être retrouvée dans le livre de Fomin et Kratsch [90].

Proposition 1.1. *Pour tous vecteurs de branchement $A \in (\mathbb{R}_+^*)^{m_A}$ et $B \in (\mathbb{R}_+^*)^{m_B}$, avec m_A, $m_B \geq 2$, les propriétés suivantes sont satisfaites :*

(i) Le polynôme $P_A(X)$ est croissant sur $\mathbb{R}_+^*$, avec $\lim_{X\to 0} P_A(X) = -\infty$, $P_A(1) \leq -1$, et $\lim_{X\to+\infty} P_A(X) = 1$.

(ii) $P_A(X)$ admet une unique racine réelle α_A qui existe, et tel que $\alpha_A \in \mathbb{R}_{>1}$.

(iii) $\alpha_A < \alpha_B$ si et seulement si $P_A(\alpha_B) > 0$ si et seulement si $P_B(\alpha_A) < 0$.

1.5.3 Programmation Dynamique

La *Programmation Dynamique* est une technique incontournable en Algorithmique Exacte Exponentielle [90]. La technique d'*Inclusion-Exclusion* se présente elle-même comme une variante de *Programmation Dynamique*. Sans considérer les variantes, il en existe différents types. L'algorithme de Lawler pour calculer le nombre chromatique d'un graphe en temps $O^*(2.4423^n)$ [136] est appelé en 2003 par Woeginger une *Programmation Dynamique sur les sous-ensembles* [188]. Nous l'appelerons *Programmation Dynamique Classique*. Nous avons présenté en Sous-Section 1.3.2 quelques décompositions de graphes. Nous appellerons *Programmation Dynamique Arborescente* une *Programmation Dynamique* dont les sous-problèmes sont donnés par un sous-arbre d'une décomposition arborescente d'un graphe.

Programmation Dynamique Classique. La technique de *Programmation Dynamique Classique* est une technique qui donne des résultats tant pour résoudre des problèmes faciles que pour résoudre des problèmes difficiles. La *Programmation Dynamique Classique* se définit comme une méthode de résolution d'un problème sur une instance d'une certaine taille, en combinant les résultats de la résolution du problème sur des instances de plus petite taille. Cette technique tire donc profit de l'hérédité d'un problème. Toutefois, ce n'est pas une programmation par récurrence. Les résultats sur les instances de plus petite taille doivent donc être calculés préalablement et être stockés afin de pouvoir être utilisés ultérieurement au cours du programme. Le fait que cette méthode ne soit pas une programmation par récurrence a un avantage au niveau de la pile d'appels d'un programme, par exemple dans le modèle de calcul ***RAM***. Cependant pour l'Algorithmique Exacte Exponentielle, cette méthode présente souvent le désavantage de nécessiter un espace mémoire exponentiel. ♣

Réaliser un algorithme de *Programmation Dynamique Classique* n'est pas toujours aisé. Il faut définir ce qu'est un sous-problème, et comment résoudre le problème à partir des sous-problèmes. Il faut également définir la dynamique de la résolution, c'est-à-dire dans quel ordre les sous-problèmes doivent être résolus, en résolvant initialement les plus petits sous-problèmes, puis en résolvant des sous-problèmes de taille toujours plus grande. La dynamique pour un problème de graphes peut être par exemple l'augmentation progressive du nombre de sommets dans les sous-graphes du graphe, les sous-graphes représentant alors les sous-problèmes de l'instance de départ.
Aujourd'hui, nous pouvons nous appuyer sur les décompositions de graphes pour obtenir des algorithmes de *Programmation Dynamique* d'un autre type. Par exemple, Fomin et Thilikos ont montré que la plupart des problèmes expressibles en **MSOL** admettent un algorithme de programmation dynamique sur une décomposition arborescente de graphe [87]. Cette méthode permet donc de construire des algorithmes sous-exponentiels pour certains problèmes appliqués à des graphes de genre borné [63]. Pour construire un algorithme de *Programmation Dynamique Arborescente*, il faut définir ce qu'est une caractéristique pour un nœud de la décomposition, et comment les obtenir en combinant les caractéristiques des sous-problèmes calculées préalablement.

Programmation Dynamique Arborescente. Soit $G = (V, E)$ un graphe, et soit (X, T) une décomposition arborescente de G, avec $T = (I, F)$ un arbre. Pour un nœud $i \in I$ de l'arbre T, on note $V_i \subseteq V$ l'ensemble des sommets de G rencontrés dans les sacs associés aux nœuds du sous-arbre de T enraciné à i, c'est-à-dire $V_i = \cup_{j \in I \cap T[i]} X_j$. On suppose que les sommets des sacs X_i respectent l'ordre des sommets de V dans le graphe G, et pour un nœud $i \in I$, on note $l(i) = |X_i|$, et on pose $X_i = \{x_1, \ldots, x_{l(i)}\}$. Pour construire un algorithme de *Programmation Dynamique* sur une décomposition arborescente, nous associons à chaque nœud de la décomposition un ensemble de caractéristiques. Le sous-arbre $T[i]$ constitue un sous-problème du problème originel, il correspond à celui restreint au sous-graphe $G[V_i]$. De manière générale, une *caractéristique* est une solution partielle au sous-problème correspondant au sous-arbre $T[i]$. Une *solution partielle* est une représentation partielle d'une solution au sous-problème restreint à $G[V_i]$. Cette représentation est partielle car en général ne sont stockées dans une caractéristique que les informations restreintes et relatives aux sommets de X_i. Pour un algorithme de *Programmation Dynamique Arborescente*, et pour un nœud $i \in I$, on note usuellement $\Gamma(i)$ l'ensemble de ses caractéristiques. Un algorithme qui utilise cette technique propage les solutions des sous-problèmes des feuilles vers la racine, et nous considérons la décomposition (X, T) jolie et enracinée en $r \in I$. On remarque en particulier que pour tout $i \in I : X_i \subseteq V_i$, et que $V_r = V$. Ainsi, un problème de décision admet une solution sur G si $\Gamma(r) \neq \emptyset$. ♣

Les techniques de *Branchement* et de *Programmation Dynamique* sont des grandes techniques majeures en Algorithmique Exacte Exponentielle. Ces techniques possèdent également des ramifications. Nous présentons ici deux techniques, la technique *Mesurer et Conquérir* qui est une technique de *Branchement*, et la technique d'*Inclusion-Exclusion* qui est une technique de *Programmation Dynamique*. Ces deux techniques constituent aujourd'hui des outils très puissants pour la construction d'algorithmes exacts exponentiels. La technique *Mesurer et Conquérir* consiste à mesurer différemment les entrées du problème, pour mieux analyser le temps d'exécution des algorithmes. Un des faits d'armes les plus célèbres de cette technique est l'amélioration du temps d'exécution d'un même algorithme de branchement pour résoudre le problème Ensemble Dominant Minimum. Avec une mesure classique, le temps d'exéuction de l'algorithme était en $O^*(1.8025^n)$, tandis qu'en choissant une mesure différente, c'est-à-dire en analysant différemment, le temps d'exécution de l'algorithme est descendu à $O^*(1.5137^n)$ [84]. Cette technique, en plus d'être redoutablement efficace, présente également comme nous le verrons, l'avantage d'améliorer notre compréhension des problèmes. La technique d'*Inclusion-Exclusion* quant à elle est une technique de *Programmation Dynamique*, que l'on applique en règle générale spécialement pour des problèmes de partitionnement. Un des faits d'armes les plus célèbres de cette technique, est l'amélioration de l'algorithme pour résoudre la version optimisation du Nombre Chromatique. Jusqu'en 2009 et le désormais célèbre algorithme d'*Inclusion-Exclusion* de Björklund, Husfeldt, et Koivisto en temps $O^*(2^n)$ [20], le meilleur algorithme connu pour résoudre ce problème était l'algorithme en temps $O^*(2.4423^n)$ de Lawler publié en 1976 [136]. Avec cette technique et toujours en temps $O^*(2^n)$, cette découverte de 2009 a considérablement amélioré d'autres algorithmes de partitionnement. Ce fut en particulier le cas du problème du nombre domatique qui consiste à déterminer la plus grande partition des sommets d'un graphe en dominants, et pour lequel le meilleur algorithme connu était en temps $O^*(2.8718^n)$ [89]. Du fait de la puissance de cette technique, on cherche aujourd'hui encore à en cerner toutes les limites et le champ d'application.

1.5.4 Mesurer et Conquérir

Mesurer et Conquérir. *Mesurer et Conquérir* est une technique de *Branchement*. Dans cette technique, on cherche cependant à mesurer l'entrée du problème différemment que de manière classique, afin d'obtenir un temps d'exécution différent et le plus petit possible. Tout l'art de cette méthode est dans le choix de la bonne mesure. Ce choix dépend du problème étudié, et peut dépendre en plus de l'algorithme lui-même. La construction d'un algorithme utilisant la technique mesurer et conquérir se déroule en quatre étapes.

La premiere étape consiste à construire et valider un algorithme de branchement qui permet de résoudre le problème étudié. Cette étape se fait de la même façon que pour la technique de branchement classique, l'analyse du temps d'exécution en moins.

La deuxième étape consiste à définir une mesure μ pour les entrées du problème étudié. Par exemple, pour un graphe d'entrée $G = (V, E)$, une mesure classique du graphe est son nombre de sommets n. Il est possible cependant de définir une autre mesure $\mu(G)$ d'un graphe G, par exemple une mesure qui ne tienne compte que des sommets de degré plus petits que $d \leq 2$. Pour se faire, nous pouvons attribuer différents poids $\omega_0, \omega_1, \omega_2$ à ces sommets en fonction de leur degré. A cette étape, l'introduction de différents poids permet d'affiner notre mesure. Egalement, ce choix peut être justifié car cela peut nous renseigner sur l'origine de la difficulté du problème, au niveau de la structure de l'entrée. Par exemple, il se peut que nous ayons introduit cette mesure car si le graphe ne comportait que des sommets de degré $d \geq 3$, le problème devenait polynomial à résoudre.

La troisième étape consiste à analyser le temps d'exécution de l'algorithme, c'est-à-dire à calculer les vecteurs de branchement de l'algorithme. Toutefois, il s'agit désormais d'exprimer les vecteurs de branchement dans la mesure μ que nous avons défini à la deuxième étape. Dans notre exemple, nous exprimons les vecteurs de branchement non plus en fonction de n le nombre de sommets, mais en fonction des poids ω_i. Les vecteurs de branchement conduisent alors à des récurrences de la forme $T(\mu(G)) \leq \sum_j T(\mu(G) - \psi_j(\omega_i))$, pour des certaines fonctions ψ_j sur les vecteurs de poids dans les réels, et où $T(\mu(G))$ est le temps d'exécution de l'algorithme dans la mesure $\mu(G)$ définie. Pour des valeurs de poids données, le temps d'exécution $T(\mu(G))$ de l'algorithme pourra alors être calculé et sera de la forme $O^*(\alpha^{\mu(G)})$. Dans le cas particulier où le gain de la mesure dans les branches d'une règle de l'algorithme s'exprime comme combinaison linéaire des variables poids, le vecteur de branchement de la règle satisfait alors une récurrence sous la forme $T(\mu(G)) \leq \sum T(\mu(G) - \sum_j a_{i,j} \cdot \omega_i)$, où les coefficients $a_{i,j}$ sont des nombres réels.

Pour comparer la qualité de cet algorithme avec d'autres algorithmes classiques, on montrera alors que pour toute entrée $x \in \mathcal{I}$ de taille n, la mesure $\mu(x)$ satisfait la relation $\mu(x) \leq f(n)$, pour une certaine fonction f. Le temps d'exécution $T(n)$ de notre algorithme sera alors $O^*(\alpha^{f(n)})$.

La dernière étape de la construction de notre algorithme utilisant la technique mesurer et conquérir consiste à optimiser la valeur des poids. Jusqu'à présent, nous n'avions aucune idée de la valeur précise des poids. Nous aurions pu leur en attribuer de manière arbitraire, mais cela n'aurait sans doute pas été un bon choix pour deux raisons. La première raison est que notre algorithme doit respecter des contraintes sur la mesure elle-même. En créant un sous-problème, nous devons garantir que le poids de l'entrée dans le sous-problème doit être inférieur au poids de l'entrée initiale. Dans chaque branche de chaque règle, qu'elle soit de réduction ou de branchement, le gain de la mesure en passant de l'instance initiale à la nouvelle instance doit toujours être positif. La valeur des poids que nous avons défini respectent donc des contraintes dont nous ne pouvons nous passer. La deuxième raison est que, comme montré à la troisième étape, la valeur des poids a un impact direct sur le temps d'exécution de notre algorithme. Il est probable qu'en choisissant tout aussi arbitrairement d'autres valeurs pour les poids, le temps d'exécution de l'algorithme soit meilleur. L'objectif même de cette quatrième étape consiste donc à optimiser la valeur des poids pour obtenir

le temps d'exécution le plus petit possible. Lorsque les récurrences sont exprimées en combinaisons linéaires des variables de la mesure, le système de récurrences obtenu à la troisième étape est dit linéaire, et son optimisation se déroule dans le cadre de la programmation quasi-convexe introduite par Eppstein en 2004 [71]. Il existe des langages mathématiques, comme AMPL, pour modéliser ces systèmes d'optimisation de variables sous contraintes, et il existe des outils pour les optimiser. Pour valider l'analyse d'un algorithme, nous n'avons cependant besoin que des valeurs annoncées des poids dans la mesure choisie, afin d'en valider le temps d'exécution. ♣

1.5.5 Inclusion-Exclusion

Inclusion-Exclusion. La technique d'*Inclusion-Exclusion*, avant d'être une technique de *Programmation Dynamique*, est un outil de Mathématiques Combinatoire. Le père de ce principe est le mathématicien Abraham de Moivre qui vécut de 1667 à 1754. Soient $A_1, \ldots, A_n$ des ensembles finis sur un univers $\mathcal{U}$ d'objets quelconque. Initialement, le principe d'*Inclusion-Exclusion* établit la relation suivante, en utilisant les opérations ensemblistes définies en Section 1.1, et en supposant autorisé des objets multiples :

$$\bigcup_{i=1,\ldots,n} A_i = -\sum_{I\subseteq\{1,\ldots,n\}} -1^{|I|} \bigcap_{i\in I} A_i$$

Ce principe s'applique aussi bien à l'énumération qu'au dénombrement ensembliste, c'est-à-dire au calcul du cardinal des ensembles. Cette formule permet de calculer la réunion des ensembles A_i à partir de leurs intersections, en prenant en compte tous les objets de l'univers mais sans qu'aucun d'eux ne soit pris en compte plus d'une fois. *Inclusion-Exclusion* est un terme qui définit bien ce principe car il se démontre par récurrence à partir de deux ensembles. En effet, pour deux ensembles A et B, on dénombre les objets qui sont dans $A \cup B$ en *incluant* les objets qui sont dans A, puis ceux qui sont dans B, et finalement en *excluant* une fois tous les éléments qui sont à la fois dans A et dans B, c'est-à-dire dans $A \cap B$, car ils ont déjà été pris en compte deux fois. Pour généraliser à n ensembles quelconques, il suffit de poser $A = A_1$ et $B = \bigcup_{i\geq 2} A_i$.
Par la suite, ce principe a également été appliqué en prenant en compte les opposés $\overline{A_i}$ des ensembles dans l'univers, qui doit dans certains cas être supposé fini. Par les lois de De Morgan, nous avons immédiatement la relation suivante par passage au complémentaire des deux membres de l'égalité :

$$\bigcap_{i=1,\ldots,n} \overline{A_i} = \sum_{I\subseteq\{1,\ldots,n\}} -1^{|I|} \bigcap_{i\in I} A_i$$

La formule ci-dessus correspond au Théorème 4.1 du livre de Fomin et Kratsch [90]. Le Théorème 4.2 du même livre peut se retrouver à partir de la formule d'origine et être formulé de la façon suivante, simplement en permutant les A_i avec leur complémentaire $\overline{A_i}$:

$$\bigcap_{i=1,\ldots,n} A_i = \sum_{I\subseteq\{1,\ldots,n\}} -1^{|I|} \bigcap_{i\in I} \overline{A_i}$$

Ce principe a de nombreuses applications en mathématiques, comme par exemple en probabilités. Pour l'Algorithmique Exacte Exponentielle, il y a un certain nombre de problèmes qui peuvent être résolus par ce principe. Dans cette Thèse, nous nous intéressons à l'application de ce principe en particulier pour des problèmes de coloration. ♣

Soient $\mathcal{U}$ un univers fini d'objets avec $n = |\mathcal{U}|$, et $\mathcal{F}$ une famille de sous-ensembles de $\mathcal{U}$. Les possibilités connues de ce principe pour ces types de problèmes sont résumées dans les théorèmes suivants. Nous nous référons au livre de Fomin et Kratsch [90] ainsi qu'à l'article de Björklund, Husfeldt, et Koivisto [20] pour des détails concernant l'implémentation de ces théorèmes.

Théorème 1.2 ([90]). *Pour tout $k \in \mathbb{N}$, il y a un algorithme d'Inclusion-Exclusion qui calcule en temps et espace $O^*(2^n)$ le nombre de k-couvertures distinctes de $\mathcal{U}$ par $\mathcal{F}$.*

Théorème 1.3 ([90]). *Pour tout $k \in \mathbb{N}$, il y a un algorithme d'Inclusion-Exclusion qui calcule en temps et espace $O^*(2^n)$ le nombre de k-partitions distinctes de $\mathcal{U}$ par $\mathcal{F}$.*

Les deux théorèmes précédents concernent le cas où la famille des sous-ensembles $\mathcal{F}$ est explicitement donnée. Le Théorème 1.4 étend ces résultats lorsque $\mathcal{F}$ ne fait pas partie de l'entrée, et donc est implicite. L'algorithme du Théorème 1.4 consiste en la construction préalable de la fonction caractéristique f de $\mathcal{F}$, puis en l'application des Théorèmes 1.2 et 1.3 précédents. L'analyse du temps d'exécution découle de celle des algorithmes de ces théorèmes.

Théorème 1.4. *Etant donnés un graphe $G = (V, E)$ et un entier positif k, il y a un algorithme d'Inclusion-Exclusion pour décider s'il existe une k-couverture ou une k-partition $\{V_1, V_2, \ldots, V_k\}$ de V, ou pour toutes les dénombrer, telle que toutes les classes de couleurs V_i appartiennent à une famille (implicitement donnée) $\mathcal{F}$ de sous-ensembles de V. Le temps d'exécution de l'algorithme est $t(n) + O^*(2^n)$, où $t(n)$ est le temps requis pour énumérer tous les éléments de $\mathcal{F}$. Avec $\mathcal{F} \subseteq 2^V$, l'espace mémoire utilisé est $O^*(|\mathcal{F}| + 2^n) = O^*(2^n)$.*

Considérons le problème bien connu du nombre chromatique. Un graphe $G = (V, E)$ possède un nombre chromatique au plus k si et seulement s'il y a une k-partition de l'ensemble $S = V$ où la famille $\mathcal{F} \subseteq 2^S$ des classes de couleurs possibles dans une telle k-partition de V est la famille des ensembles stables de G. Observons que cette famille particulière $\mathcal{F}$ peut facilement être calculée en temps $O^*(2^n)$. Dans la pratique, nous avons $t(n) = O^*(2^n)$ dans la majorité des cas.
Les algorithmes d'*Inclusion-Exclusion* présentés jusqu'ici s'exécutent en temps $O^*(2^n)$, et requièrent un espace exponentiel. Pour établir des algorithmes qui s'exécutent en espace polynomial, il est nécessaire que la famille $\mathcal{F}$ puisse être énumérée en espace polynomial. Le Théorème 1.5 résume différentes possibilités pour établir des algorithmes d'*Inclusion-Exclusion* pour les problèmes de couverture ou de partitionnement qui s'exécutent en espace polynomial. La preuve du Théorème 1.5 est donnée dans le livre de Fomin et Kratsch [90].

Théorème 1.5 ([90]). *Le nombre de k-couvertures ou de k-partitions distinctes d'une instance $(\mathcal{U}, \mathcal{F})$ peut être calculé par un algorithme d'Inclusion-Exclusion en espace polynomial et en temps :*

(i) $2^n|\mathcal{F}|n^{O(1)}$, *si $\mathcal{F}$ peut être énumérée en délai et espace polynomial*

(ii) $3^n n^{O(1)}$, *si l'appartenance à $\mathcal{F}$ peut être décidée en temps polynomial*

(iii) $\sum_{j=0}^{n} \binom{n}{j} T_{\mathcal{F}}(j)$, *s'il y a un algorithme en temps $T_{\mathcal{F}}(j)$ et espace polynomial pour compter pour chaque sous-ensemble $W \subseteq \mathcal{U}$ de taille j le nombre d'ensembles $F \in \mathcal{F}$ qui satisfont $F \cap W = \emptyset$.*

Partition Pondérée. Il existe une extension du problème d'*Inclusion-Exclusion* aux problèmes de partitionnement pondéré. Une instance du problème de Partition Pondérée est un univers d'objets $\mathcal{U}$, et une fonction de pondération f sur les k-tuples $\{U_1, \ldots, U_k\}$ de sous-ensembles de $\mathcal{U}$. La fonction f est ici un produit tensoriel $\otimes$ de k fonctions sur les sous-ensembles U de $\mathcal{U}$ dans un intervalle $[-M, M]$ pour une certaine valeur M. Formellement on a pour tout $1 \leq i \leq k$:

$$\begin{array}{rccc} f_i & : & 2^{\mathcal{U}} & \to & [-M, M] \\ & & U & \mapsto & f_i(U) \end{array}$$

et la fonction f de pondération des k-tuples est donnée par :

$$\begin{array}{rccc} f & : & 2^{\mathcal{U}} \times 2^{\mathcal{U}} \times \ldots \times 2^{\mathcal{U}} & \to & \left[-M^k, M^k\right] \\ & & \{U_1, U_2, \ldots, U_k\} & \mapsto & f_1(U_1) \otimes f_2(U_2) \otimes \ldots \otimes f_k(U_k) \end{array}$$

Dans la pratique, nous utiliserons l'addition ou la multiplication pour le produit tensoriel. ♣

Les théorèmes suivants agrémentent les résultats sur les algorithmes d'*Inclusion-Exclusion*.

Théorème 1.6 ([20]). *Soient $\mathcal{U}$ un univers d'objets, $k \in \mathbb{N}$ et f une fonction de pondération définie comme précédemment par un produit tensoriel. Alors il y a un algorithme d'Inclusion-Exclusion qui s'exécute en temps $O^*(2^n\ k\ \log M)$ et espace $O^*(2^n)$ pour calculer $p_k(f)$ la somme des partitions pondérées défini par :*

$$p_k(f) = \sum_{X=\{U_1,\ldots,U_k\}} f(X) = \sum_{X=\{U_1,\ldots,U_k\}} \otimes_{i=1}^{k} f_i(U_i)$$

où quelque soit $X = \{U_1, \ldots, U_k\}$, X est une k-partition de $\mathcal{U}$.

Théorème 1.7 ([20]). *Soient $\mathcal{U}$ un univers d'objets, $k \in \mathbb{N}$ et f une fonction de pondération définie comme précédemment par un produit tensoriel. Alors il y a un algorithme d'Inclusion-Exclusion qui s'exécute en temps $O^*(2^n\ k^2\ \log M)$ et espace $O^*(2^n)$ pour déterminer une k-partition $X = \{U_1, \ldots, U_k\}$ de $\mathcal{U}$ qui maximise $f(X) = f_1(U_1) \otimes \ldots \otimes f_k(U_k)$.*

1.5.6 Autres Méthodes Exactes

Dans cette Sous-Section, nous présentons des techniques qui ont une importance moindre en Algorithmique Exacte Exponentielle. Ceci peut se justifier soit par le fait que ces techniques sont en général mises en œuvre en parallèle d'autres techniques plus importantes que nous avons déjà présentées, soit par le fait qu'un faible nombre d'algorithmes exacts exponentiels efficaces empreintent ces techniques. Quoiqu'il en soit, nous utiliserons dans cette Thèse chacune des techniques présentées dans la Section 1.5 et ses Sous-Sections.

Pré-Traitement. La technique de *Pré-Traitement* est par définition même une technique qui s'utilise en parallèle d'une autre technique. D'un point de vue algorithmique, cette technique consiste à transformer préalablement l'entrée d'un problème, en vue d'améliorer l'efficacité de l'exécution du coeur d'un algorithme. Pour mettre en œuvre cette technique, on pourra utiliser de plus une autre technique déjà présentée. L'emploi de la technique de *Pré-Traitement* est en général implicite, et peut par exemple consister en un tri de l'entrée, ou encore en un calcul préalable de tous les dominants d'un graphe. Nous référons à l'étude de Woeginger de 2003 pour des exemples concrets d'application de cette méthode [188]. ♣

Compression Iterative. La technique de *Compression Iterative* est une technique très récente et d'abord issue de l'Algorithmique Paramétrée [104, 56, 38, 113]. L'idée principale d'un algorithme paramétré utilisant la méthode de *Compression Iterative* est, étant donnée une solution de taille $k+1$ pour un certain problème, soit *compresse* celle-ci et produit une solution de taille k, ou alors démontre qu'il n'existe pas de solution de taille k. Pour l'Algorithmique Exacte Exponentielle, la méthode est plutôt inversée. Etant donnés un problème sur une instance de taille n et un certain paramètre $k \in \mathbb{N}$, l'objectif de la méthode de *Compression Iterative* est, à partir d'un algorithme de résolution en temps $O^*(f(k,n))$, de produire un algorithme en temps $O^*(g(k+1,n))$ pour deux fonctions quelconques f et g. Un exemple d'application de cette technique sont les résultats de 2008 de Fomin, Gaspers, Kratsch et al. pour des versions de dénombrement ou d'optimisation du problème d'hypergraphes k-Hitting Set [82]. Pour notre part, nous appliquerons cette méthode à la version d'énumération des transversaux minimaux d'un hypergraphe de rang borné $k \in \mathbb{N}_{\geq 3}$ que nous développerons dans le Chapitre 3. ♣

Balancement d'Algorithmes. Pour mettre en œuvre cette technique, il faut disposer de plusieurs algorithmes pour résoudre le même problème. De plus, ces algorithmes doivent en général utiliser différentes approches de résolution. En particulier, les temps d'exécution de ces algorithmes doivent pouvoir être exprimés en fonction d'un paramètre commun. L'objectif de la technique de *Balancement d'Algorithmes* est alors de produire un algorithme qui choisit et utilise judicieusement ces algorithmes déjà existants, en fonction de la valeur du paramètre pour l'instance considérée. En mettant en commun ces algorithmes par cette méthode de *Balancement d'Algorithmes*, on permet une nouvelle et meilleure analyse du temps d'exécution de résolution du problème. On peut avoir à disposition plusieurs algorithmes différents, mais il est à noter que seulement deux suffisent pour l'analyse au pire des cas. Prendre en compte tous les autres permet cependant de se faire une idée des axes d'amélioration possibles en fonction de la valeur du paramètre.
Nous renvoyons à la Section 5.2 de cette Thèse et l'étude du problème Ensemble Tropical Connexe pour un exemple de mise en œuvre de cette technique. ♣

Chapitre 2

Algorithmes pour des Problèmes de Coloration

Sommaire

2.1 Introduction

Les problèmes de colorations sont des problèmes très étudiés tant d'un point de vue de la Théorie des Graphes que d'un point de vue Algorithmique. Un problème de coloration sur les sommets

d'un graphe se présente sous la forme suivante. Etant donné un graphe $G = (V, E)$, on souhaite décider l'existence, déterminer, dénombrer ou optimiser la taille d'une partition $\{V_1, \ldots, V_k\}$ de V qui satisfait plusieurs propriétés. Chaque classe de couleur V_i doit satisfaire une certaine propriété, qui peut lui être propre, et les classes de couleurs entre elles doivent satisfaire une autre propriété.

Le problème de coloration le plus célèbre est celui du Nombre Chromatique d'un graphe. Pour autant, selon sa *formulation*, le problème apparaît plus ou moins complexe. Sous la forme présentée en Sous-Section 1.3.4, le problème du Nombre Chromatique revient à déterminer une partition $\{V_1, \ldots, V_k\}$ des sommets, telle que toute classe de couleur doit satisfaire la propriété d'ensemble stable. On peut donc penser que ce problème n'est qu'un cas particulier du problème plus général de Partition d'Ensembles, donc au sein duquel les relations entre les classes de couleurs n'ont pas d'importance. Cependant, on sait qu'il existe une coloration optimale d'un graphe si et seulement si les classes de couleurs peuvent être contractées pour obtenir un graphe complet. Ainsi, la formulation équivalente suivante exhibe cette relation entre les classes de couleurs, et fait apparaître ainsi une plus grande complexité du problème.

k-Coloration Propre
Version : *Décision*

Entrée : $G = (V, E)$, $k \in \mathbb{N}$
Sortie : Existe-t-il une partition $\{V_1, \ldots, V_k\}$ de V en k stables telle que le graphe complet à k sommets peut être obtenu par contraction des classes de couleurs ?

La formulation pour un problème de coloration a donc son importance. Nous savons que la version décisionnelle du problème du Nombre Chromatique est un problème NP-complet [94]. C'est un des célèbres 21 problèmes NP-complets de Karp [121]. Beaucoup d'efforts ont été engagés pour obtenir des algorithmes toujours plus efficaces pour résoudre ce problème. Les plus grands progrès ont été obtenus sous l'angle de la première formulation, puisqu'il rend le problème en apparence plus simple. La seconde formulation est toute aussi importante, car il a permis d'étudier tout un tas de variantes du problème. En particulier, les problèmes de type Vertex Grouping se formulent de la même façon, les variantes du problème se concentrant sur le graphe à obtenir par contraction des classes de couleurs [36].

Un algorithme trivial pour résoudre un problème de coloration s'exécute en temps $\Omega(n^n)$. Rapidement, un algorithme plus efficace pour résoudre le problème du Nombre Chromatique a été trouvé. Le meilleur algorithme pour résoudre ce problème de manière exacte a longtemps été un algorithme qui s'appuie sur le nombre d'ensembles stables maximaux d'un graphe [159], et sur un algorithme à *délai polynomial* pour les énumérer [182]. L'algorithme obtenu par *Programmation Dynamique* permet donc d'établir un temps d'exécution de $O^*(2.4423^n)$ [136]. Depuis 1976 et pendant une trentaine d'années, cet algorithme n'a pas été sensiblement amélioré, excepté sur certaines classes de graphes d'entrée particulières, ou pour décider la k-colorabilité d'un graphe pour $k \leq 4$ [16, 83]. L'avancée la plus importante a sans aucun doute été celle de Bjorklünd, Husfeldt, et Koivisto [20], qui ont proposé en 2009 un algorithme basé sur le principe d'*Inclusion-Exclusion* pour résoudre la version dénombrement du problème de Partition d'Ensembles. Cet algorithme s'exécute en temps $O^*(2^n)$, où n est le nombre d'objets de l'univers, et peut s'appliquer à d'autres problèmes de colorations, sitôt que l'on puisse énumérer trivialement les classes de couleurs, et que l'on puisse formuler le problème sans faire apparaitre de contraintes entre les classes

de couleurs [90]. Aujourd'hui, la technique d'*Inclusion-Exclusion* est une technique majeure pour la résolution de problèmes de coloration.

2.1.1 Nombre Chromatique

Par le Théorème des quatre couleurs, tout graphe planaire est 4-colorable [169]. De plus, décider si un graphe est biparti est un problème facile, qui peut être résolu par un simple parcours en profondeur du graphe. Pour calculer le nombre chromatique d'un graphe planaire non biparti, il suffit donc de décider s'il est 3-colorable. Courcelle a montré que tous les problèmes de graphe expressibles en logique **MSOL** sont solubles en temps $O^*(f(\mathbf{tw}(G)))$ [48]. Le problème de 3-coloration est expressible en logique **MSOL** [22], nous allons montrer qu'il appartient à la classe **SUBEXP** lorsque l'entrée est un graphe planaire.
Les algorithmes de résolution de problèmes qui utilisent la *Programmation Dynamique* sur une décomposition en branches de Fomin et Thilikos sont célèbres pour s'exécuter en temps sous-exponentiel sur les graphes planaires. Ces auteurs ont montré qu'un algorithme qui s'exécute en temps $O^*(f(\mathbf{tw}(G)))$ pour une certaine fonction f peut être transposé en un algorithme en temps $O^*(f(1.5\cdot\mathbf{bw}(G)))$ [87]. Pour un graphe planaire, on a $\mathbf{bw}(G) \leq 2.122\sqrt{n}$ [91], et une décomposition en branche optimale peut être calculée en temps polynomial [177].
Il y a des algorithmes en temps $O^*(c^{f(k)\cdot\mathbf{tw}(G)})$ pour décider la k-coloration d'un graphe, pour une certaine fonction f et une certaine constante $c \in \mathbb{R}_{>1}$. Par exemple, Telle et Proskurowski ont construit en 1997 des algorithmes pour résoudre des problèmes sur des graphes de largeur arborescente bornée, par le biais d'une formulation très générique de ces problèmes [180]. Comme résultat, ils proposent un algorithme en temps $O^*(k^{2\cdot\mathbf{tw}(G)})$ pour décider la k-coloration d'un graphe G.
Afin d'illustrer ces résultats, nous construisons un algorithme en temps $O^*(k^{\mathbf{tw}(G)})$ pour décider la k-coloration d'un graphe. Cet algorithme de *Programmation Dynamique Arborescente* s'exécute en temps sous-exponentiel $O^*(2^{5.044\sqrt{n}})$ pour calculer le nombre chromatique d'un graphe planaire. Comparativement, ce temps d'exécution est sensiblement le même que celui de l'algorithme de résolution du problème Ensemble Dominant pour les graphes planaires proposé par Fomin et Thilikos en temps $O^*(2^{5.043\sqrt{n}})$ [87].

Afin de construire notre algorithme, nous montrons les lemmes suivants.

Lemme 2.1. *Soit (X, T) une décomposition arborescente jolie d'un graphe $G = (V, E)$, avec $T = (I, F)$, et soit $i \in I$. Si i est de type* introduce*, avec pour fils $j \in I$ et pour sommet introduit $v \in V$, alors les seuls voisins de v parmi les sommets de V_i sont tous dans X_i, c'est-à-dire $N_G(v) \cap V_i \subseteq X_i$.*

Démonstration. Supposons par contradiction que v soit adjacent à un sommet $y \in V_i - X_i$. Par la deuxième propriété d'une décomposition arborescente, l'arête $\{v, y\}$ est contenue dans un certain sac $X_l \in X$. En particulier, $v \in X_l$ et $y \in X_l$. Dans un premier cas, si $l = i$, alors $y \in X_i$, ce qui est une contradiction, puisque $y \in V_i - X_i$. Dans un deuxième cas, si $l \in T[i]$, alors j l'unique fils de i dans T est sur le chemin qui relie l à i. La troisième propriété d'une décomposition arborescente implique alors que $v \in X_j$, ce qui est une contradiction, puisque i est de type INTRODUCE de sommet introduit v. Dans le troisième et dernier cas, si $l \notin T[i]$, nous observons que $y \in V_i - X_i$ implique qu'il existe $p \in T[i]$, $p \neq i$ tel que $y \in X_p$. Or puisque $y \in X_l$, et puisque i est sur l'unique chemin dans T qui relie p à l, on en déduit que $y \in X_i$, ce qui contredit que $y \in V_i - X_i$. Nous avons donc montré que $N_G(v) \cap V_i \subseteq X_i$. □

Lemme 2.2. *Soit (X, T) une décomposition arborescente d'un graphe $G = (V, E)$, avec $T = (I, F)$, et soit $i \in I$ un nœud de la décomposition. Alors les seuls voisins dans G des sommets de $V_i - X_i$ sont dans V_i, c'est-à-dire $N_G(V_i - X_i) \subseteq V_i$.*

Démonstration. . Soit $x \in V_i - X_i$. En particulier, par définition de V_i, x appartient à un sac $X_j \in X$ pour un certain $j \in T[i]$. Supposons qu'il existe $y \notin V_i$ tel que $\{x, y\} \in E$. Par la deuxième propriété d'une décomposition arborescente, l'arête $\{x, y\}$ est contenue dans un certain sac $X_l \in X$. En particulier, $y \in X_l$ et $x \in X_l$ pour un certain $l \in I$. Dans un premier cas, si $l = i$ ou si $l \in T[i]$, alors par définition $X_l \subseteq V_i$, et on a $y \in V_i$, ce qui contredit que $y \notin V_i$. Dans un deuxième et dernier cas, si $l \notin T[i]$, alors i est sur l'unique chemin dans T qui relie j à l. $x \in X_j$ et $x \in X_l$ implique par la troisième propriété d'une décomposition arborescente que $x \in X_i$, ce qui contredit que $x \in V_i - X_i$. □

Corollaire 2.3. *Soit (X, T) une décomposition arborescente jolie d'un graphe $G = (V, E)$, avec $T = (I, F)$, et soit $i \in I$. Si i est de type* join*, avec pour fils $i_1, i_2 \in I$, alors les seuls voisins dans G des sommets de $V_{i_1} - X_i$ sont dans V_{i_1}, et symétriquement les seuls voisins dans G des sommets de $V_{i_2} - X_i$ sont dans V_{i_2}.*

Démonstration. La preuve de ce corollaire se déduit du Lemma 2.2 en remarquant que $X_{i_1} = X_{i_2} = X_i$. Ainsi, si i est de type JOIN, alors $N_G(V_{i_1} - X_i) \subseteq V_{i_1}$, de même que $N_G(V_{i_2} - X_i) \subseteq V_{i_2}$. Autrement dit, il n'y a pas d'arête dans G qui relie un sommet de $V_{i_1} - X_i$ à un sommet de $V_{i_2} - X_i$. □

Corollaire 2.4. *Soit (X, T) une décomposition arborescente jolie d'un graphe $G = (V, E)$, avec $T = (I, F)$, et soit $i \in I$ de type* join*. Soit un sous-ensemble $S \subseteq V_i$ tel que $S = S_{i_1} \cup S_{i_2}$ pour deux ensembles $S_{i_1} \subseteq V_{i_1}$ et $S_{i_2} \subseteq V_{i_2}$. Alors S est stable dans $G[V_i]$ si et seulement si S_{i_1} et S_{i_2} sont stables respectivement dans $G[V_{i_1}]$ et $G[V_{i_2}]$.*

Théorème 2.5. *Il y a un algorithme de Programmation Dynamique qui s'exécute en temps sous-exponentiel $O^*(2^{5.044\sqrt{n}})$ pour déterminer le nombre chromatique d'un graphe planaire.*

Démonstration. Nous présentons un algorithme de *Programmation Dynamique Arborescente* pour décider la k-colorabilité d'un graphe. Soit $G = (V, E)$ un graphe, et soit (X, T) une décomposition arborescente jolie de G, avec $T = (I, F)$ et $r \in I$ la racine de l'arbre T. Un vecteur $(k_1, \ldots, k_{l(i)}) \in \{1, \ldots, k\}^{l(i)}$ est une caractéristique d'un nœud $i \in I$, s'il existe une k-coloration propre $\varphi : V_i \to \{1, \ldots, k\}$ des sommets de V_i dans $G[V_i]$ telle que $\forall\ 1 \leq p \leq l(i)\ :\ \varphi(x_p) = k_p$. Dans notre contexte, une caractéristique est donc la représentation partielle d'une k-coloration propre de $G[V_i]$. Etant donnée une caractéristique $(k_1, \ldots, k_{l(i)}) \in \Gamma(i)$, on note $S_i(c) = \{x_p \in X_i, 1 \leq p \leq l(i)\ :\ k_p = c\}$ l'ensemble des sommets de X_i de couleur $c \in \{1, \ldots, k\}$ selon la caractéristique. L'Algorithme 2.1, pour décider l'existence d'une k-coloration, associe l'ensemble des caractéristiques $\Gamma(i)$ d'un nœud $i \in I$ en fonction des caractéristiques de ses sous-problèmes déjà calculées.
La validité d'un algorithme de *Programmation Dynamique* sur une décomposition arborescente se montre par récurrence sur les sommets de la décomposition, en partant des feuilles vers la racine. Montrons que quel que soit le nœud $i \in I$, on a $G[V_i]$ admet une coloration propre φ si et seulement s'il existe dans $\Gamma(i)$ une caractéristique $(k_1, \ldots, k_{l(i)})$ telle que $\forall\ 1 \leq p \leq l(i)\ :\ \varphi(x_p) = k_p$. Supposons que l'Algorithme 2.1 construise correctement les caractéristiques de tous les sous-problèmes d'un nœud $i \in I$, montrons alors que cet algorithme calcule correctement les caractéristiques $\Gamma(i)$.

- ◇ Si i est une FEUILLE : Un sommet isolé, pour constituer une coloration propre de lui-même, peut être coloré de toutes les manières possibles. Par l'Algorithme 2.1, toutes les représentations partielles restreintes à X_i de ces colorations sont dans $\Gamma(i)$.

Algorithme 2.1: Algorithme de *Programmation Dynamique* du Théorème 2.5 pour la version décisionnelle du problème de k-COLORATION qui associe à chaque nœud $i \in I$ d'une décomposition arborescente jolie $T = (X, (I, F))$ d'un graphe $G = (V, E)$, l'ensemble de ses caractéristiques $\Gamma(i)$

input : $G = (V, E)$, $T = (X, (I, F))$, et $i \in I$
output: $\Gamma(i)$
// Étape d'initialisation
$\Gamma(i) \leftarrow \emptyset$;
// Différents cas selon le type du nœud i
if i *est de type* join **then**
  $\Gamma(i) \leftarrow \Gamma(i_1) \cap \Gamma(i_2)$;
if i *est de type* introduce **then**
  Soit $X_i = \{x_1, \ldots, x_{p-1}, v, x_p, \ldots, x_{l(j)}\}$, pour un certain $1 \leq p \leq l(j) + 1$;
  foreach $(k_1, \ldots, k_{l(j)}) \in \Gamma(j)$ **do**
    for $k(v) = 1$ **to** k **do**
      if $S_i(k(v)) = S_j(k(v)) \cup \{v\}$ *est un stable de* G **then**
        $\Gamma(i) \leftarrow \Gamma(i) \cup \{(k_1, \ldots, k_{p-1}, k(v), k_p, \ldots, k_{l(j)})\}$;
if i *est de type* forget **then**
  Soit $X_j = \{x_1, \ldots, x_{p-1}, v, x_p, \ldots, x_{l(i)}\}$, pour un certain $1 \leq p \leq l(i) + 1$;
  foreach $(k_1, \ldots, k_{p-1}, k(v), k_p, \ldots, k_{l(i)}) \in \Gamma(j)$ **do**
    $\Gamma(i) \leftarrow \Gamma(i) \cup \{(k_1, \ldots, k_{p-1}, k_p, \ldots, k_{l(j)})\}$;
else
  // Ici i est une feuille
  for $j = 1$ **to** k **do**
    $\Gamma(i) \leftarrow \Gamma(i) \cup \{(j)\}$;
return $\Gamma(i)$;

◇ Si i est de type FORGET : En observant que $G[V_i] = G[V_j]$, les sous-problèmes correspondant aux sous-arbres $T[i]$ et $T[j]$ sont les mêmes. Ainsi donc, il existe une k-coloration propre φ de $G[V_i]$ représentée partiellement par une caractéristique du nœud i si et seulement si φ est une k-coloration propre de $G[V_j]$. Par hypothèse, $\Gamma(j)$ est correctement calculé, on en déduit la validité de la construction de $\Gamma(i)$.

◇ Si i est de type INTRODUCE : Il existe une k-coloration propre φ de $G[V_i]$ représentée partiellement par une caractéristique du nœud i si et seulement si φ est une k-coloration propre de $G[V_j]$ représentée partiellement par une caractéristique du nœud j et si la classe de couleur de v est stable. Or par le Lemme 2.1, les seuls voisins de v sont dans X_i. On en déduit donc que pour une k-coloration φ de $G[V_i]$, les classes de couleur de φ sont des stables si et seulement si φ est une k-coloration propre de V_j et si $S_i(k(v))$ est stable dans G. Puisque par hypothèse, $\Gamma(j)$ est correctement calculé, on en déduit la validité de la construction de $\Gamma(i)$ par l'algorithme 2.1.

◇ Si i est de type JOIN : Par le Corollaire 2.4, on déduit qu'il existe une k-coloration propre φ de $G[V_i]$ si et seulement s'il existe des trois colorations propres φ_1, φ_2 respectivement de $G[V_{i_1}]$ et de $G[V_{i_2}]$ qui satisfont $\forall\ x \in X_i\ :\ \varphi_1(x) = \varphi_2(x)$. Ces trois colorations φ, φ_1, φ_2 ont la même représentation partielle respectivement dans X_i, X_{i_1} et X_{i_2}. Autrement dit, une caractéristique

du nœud i est dans $\Gamma(i)$ si et seulement si elle est à la fois dans $\Gamma(i_1)$ et dans $\Gamma(i_2)$, ce qui valide la construction de $\Gamma(i)$ par l'algorithme lorsque i est de type JOIN.

Pour analyser le temps d'exécution, on munit l'Algorithme 2.1 d'un codage c en base k qui à toute k-coloration de $l(i) \in \mathbb{N}$ sommets $X = (k_1, \ldots, k_{l(i)})$ associe un entier $c(X) = \sum_{i=1}^{l(i)} k_i \cdot k^{i-1}$. Ce code transforme l'ensemble des caractéristiques de chaque nœud $i \in I$ en ensemble ordonné. La fonction de codage c et sa fonction de décodage s'exécutent en temps $O(l(i))$. On associe ensuite à chaque nœud $i \in I$ de la décomposition une structure de données pour stocker ses caractéristiques. Cette structure est choisie pour permettre la recherche ou l'insertion dans un ensemble ordonné à n éléments en temps $O(\log n)$, et le parcours de tous les éléments non ordonnés en temps $O(n)$. Ceci peut être réalisé par un Arbre Rouge et Noir doublé d'une Liste Doublement Chaînée. Nous renvoyons au livre de *Introduction To Algorithms* pour une introduction à ces structures de données [141]. Le temps d'exécution des caractéristiques $\Gamma(i)$ d'un nœud $i \in I$ de la décomposition dépend alors du type de i :

- ◇ Si i est une FEUILLE : L'algorithme effectue k insertions distinctes dans $\Gamma(i)$, et s'exécute donc en temps $O(k \log k)$
- ◇ Si i est de type FORGET : L'algorithme effectue un parcours des caractéristiques de $\Gamma(j)$, et au plus $k^{l(j)}$ recherches et $k^{l(i)}$ insertions dans $\Gamma(i)$.
  L'algorithme s'exécute donc en temps $O(k^{l(j)} + (k^{l(j)} + k^{l(i)}) \log k^{l(i)}) = O(\mathbf{tw}(G) \cdot \log k \cdot k^{\mathbf{tw}(G)})$.
- ◇ Si i est de type INTRODUCE : L'algorithme effectue un parcours des caractéristiques de $\Gamma(j)$, et au plus $k \cdot k^{l(j)}$ insertions dans $\Gamma(i)$.
  L'algorithme s'exécute donc en temps $O^*(k^{l(j)} + k^{l(j)+1} \cdot \log k^{l(i)}) = O^*(\mathbf{tw}(G) \cdot \log k \cdot k^{\mathbf{tw}(G)})$.
- ◇ Si i est de type JOIN : L'algorithme effectue un parcours des caractéristiques de $\Gamma(i_1)$, au plus $k^{l(i_1)}$ recherches dans les caractéristiques de $\Gamma(i_2)$, et au plus $k^{l(i_1)}$ insertions dans $\Gamma(i)$.
  Avec $l(i_1) = l(i_2) = l(i)$, l'algorithme s'exécute donc en temps $O(k^{l(i)} + k^{l(i)} \cdot \log k^{l(i)}) = O(\mathbf{tw}(G) \cdot k^{\mathbf{tw}(G)} \cdot \log k)$.

Pour $k \in \mathbb{N}_{\geq 3}$ fixé, et avec $\mathbf{tw}(G) \leq n$ pour tout graphe G, l'Algorithme 2.1 s'exécute donc en temps $O^*(k^{\mathbf{tw}(G)})$. Pour $k = 3$, et en reprenant les bornes affinées de Fomin et Thilikos [91], l'Algorithme 2.1 s'exécute donc en temps sous-exponentiel $O^*(3^{3.182\sqrt{n}}) = O^*(2^{5.044\sqrt{n}})$. □

2.1.2 Nos Résultats

Ce chapitre s'attache à étudier des problèmes de colorations de sommets de graphes, qui apparaissent comme des variantes plus compliquées que le NOMBRE CHROMATIQUE, soit en introduisant une propriété non triviale de classe de couleurs, comme le problème de CLIQUE COLORATION, soit en introduisant des relations entre les classes de couleurs, comme le problème de b-COLORATION. Dans la majorité des cas, chacun de ces problèmes étudiés possède certaines relations avec les autres. L'objectif est alors de développer des algorithmes en temps $O^*(c^n)$ pour résoudre ces problèmes, pour une certaine constante $c \in \mathbb{R}_{>1}$ la plus petite possible. Pour ce faire, soit nous utiliserons la *Programmation Dynamique*, ou alors nous étendrons le principe d'*Inclusion-Exclusion* à des types de problèmes qui n'avaient pas été appliqués jusqu'alors.

Dans ce Chapitre, nous établissons entre autres les résultats principaux suivants :

- ◇ Nous montrons dans la Section 2.2 que le nombre grundy d'un graphe peut être obtenu en temps $O^*(2.4423^n)$ et espace $O^*(2^n)$, en utilisant une variante de l'algorithme de Lawler en *Programmation Dynamique Classique* pour calculer le nombre chromatique d'un graphe.
- ◇ Nous construisons dans la Section 2.3 un algorithme de *Programmation Dynamique Classique* en temps $O^*(4^n)$ et espace $O^*(3^n)$ pour calculer le nombre grundy partiel d'un graphe introduit par Erdős en 2003.

- ◇ Nous construisons dans la Section 2.4 un algorithme de *Programmation Dynamique Arborescente* pour le nombre a-chromatique d'un graphe. En particulier, à un facteur sous-exponentiel près, cet algorithme s'exécute en temps et espace *single exponentiel* $O^*(2^n)$ et $O^*(8^n)$ lorsqu'il est appliqué respectivement sur les arbres et sur les graphes planaires.
- ◇ Dans la Section 2.5, nous construisons le premier algorithme d'*Inclusion-Exclusion* pour un problème de cette liste, celui de calculer le nombre b-chromatique d'un graphe. Ce problème de coloration est un problème qui fait apparaître de fortes relations entre les classes de couleurs.
- ◇ Dans la Section 2.6, nous présentons un algorithme d'*Inclusion-Exclusion* pour déterminer le nombre clique-chromatique d'un graphe. Ce problème apparaît comme le problème le plus compliqué de cette liste, puisqu'il est Σ_2^p-difficile. En particulier, décider l'appartenance d'un sous-ensemble de sommets à la famille des classes de couleur est un problème coNP-complet. Le temps d'exécution et l'espace en $O^*(2^n)$ de notre algorithme est obtenu au prix d'une étude approfondie des transversaux de l'hypergraphe des cliques maximales d'un graphe, et grâce à l'introduction de la notion d'obliques. Ce résultat a fait l'objet en 2014 d'une publication [42].
- ◇ Finalement, nous étudions dans la Section 2.7 d'autres variantes de problèmes de colorations fortement reliées au problème du nombre a-chromatique. Plus particulièrement, nous montrons comment adapter l'algorithme de *Programmation Dynamique Arborescente* de la Section 2.4 et obtenir des résultats similaires à ceux obtenus pour le nombre a-chromatique, afin de calculer le nombre pseudo-a-chromatique, chromatique harmonieux, ou chromatique pseudo-harmonieux d'un graphe.

2.2 Nombre Grundy

2.2.1 Introduction

Déterminer une coloration propre optimale d'un graphe est un problème NP-difficile [94]. L'algorithme *greedy coloring* est un algorithme *glouton* qui construit une coloration propre φ d'un graphe $G = (V, E)$ en évaluant dans un ordre donné $v_1, \ldots, v_n$ les sommets du graphe. Il stocke l'ensemble des couleurs déjà utilisées $\{1, \ldots, k\}$ pour colorier les sommets $v_1, \ldots, v_{i-1}$. Puis l'algorithme évalue le sommet v_i. S'il y a une couleur $1 \leq c \leq k$ telle que $\{v_i\} \cup \{v_j,\ 1 \leq j \leq i-1\ :\ \varphi(v_j) = c\}$ est un stable dans G, alors l'algorithme attribue $\varphi(v_i) = c$. Sinon, l'algorithme attribue $\varphi(v_i) = k + 1$.

La coloration propre φ des sommets fournie par cet algorithme est appelée *parsimonious proper coloring* par Erdős et al. en 1987 [72]. Il est clair que lors de son évaluation, une nouvelle couleur est attribuée au sommet v_i à la seule condition que v_i soit adjacent à un sommet de chaque classe de couleur déjà existante. La pire valeur que puisse retourner cet algorithme, c'est-à-dire le plus grand nombre de couleur utilisées par l'algorithme parmi tous les ordres possibles des sommets, est appelée le *nombre ochromatique*. Cette notion a été introduite et définie par Simmons en 1982.

A l'instar de l'Algorithmique d'Approximation où on cherche à évaluer et améliorer l'écart à la valeur optimale de la valeur trouvée par un algorithme, le nombre grundy d'un graphe mesure donc le pire écart possible entre le nombre de couleurs utilisées par l'algorithme *greedy coloring* et le nombre chromatique.

Nombre Grundy. La notion de nombre grundy d'un graphe a été introduite en 1979 par Christen et Selkow [39]. Nous reprendrons dans cette Section la notation $\Gamma(G)$ pour désigner le nombre grundy d'un graphe $G = (V, E)$. A l'origine, le problème du Nombre Grundy avait été formulé de la façon suivante :

k-Coloration Grundy
Version : *Décision*

Entrée : $G = (V, E)$, $k \in \mathbb{N}^*$
Sortie : Existe-t-il une partition $\varphi = \{V_1, \ldots, V_k\}$ de V en k stables telle que pour tout $1 \leq i \leq k$, pour tout sommet $x \in V_i$, et pour tout $1 \leq j < i$, il existe un sommet $y \in V_j$ vérifiant $\{x, y\} \in E$?

Nous avons vu que la formulation d'un problème a son importance. Nous reformulons le problème du Nombre Grundy de la manière équivalente suivante :

k-Coloration Grundy
Version : *Décision*

Entrée : $G = (V, E)$, $k \in \mathbb{N}^*$
Sortie : Existe-t-il une partition $\varphi = \{V_1, \ldots, V_k\}$ de V en k stables telle que $\forall\, x \in V$, $\forall\, 1 \leq i < \varphi(x) : x$ est adjacent à un sommet $y \in V_i$?

Pour une k-coloration $\varphi = \{V_1, \ldots, V_k\}$ des sommets d'un graphe $G = (V, E)$, un sommet $x \in V$ est appelé *sommet grundy* si pour tout $1 \leq i < \varphi(x) : x$ est adjacent à au moins un sommet de couleur i. Une *k-coloration grundy* peut donc être définie de manière équivalente comme une k-coloration propre des sommets telle que tous les sommets de V sont des sommets grundys. Le *Nombre Grundy* d'un graphe $G = (V, E)$, noté $\Gamma(G)$, est alors défini comme le plus grand entier $k \in \mathbb{N}$ tel que G admette une k-coloration grundy.
Comme nous l'avons présenté, ce paramètre de graphe est directement relié au problème du Nombre Chromatique. Il peut être vu, par l'intermédiaire de l'algorithme *greedy coloring*, à la fois comme un problème d'ordonnancement des sommets, tout comme il peut être vu comme un problème de coloration des sommets avec des fortes relations entre les classes de couleur. Notons toutefois qu'en 1987, Erdős et al. ont montré que pour tout graphe, ces deux valeurs nombre ochromatique et nombre grundy sont exactement les mêmes [72]. ♣

Déterminer le nombre grundy d'un graphe est un problème NP-difficile [195]. Zaker a montré en 2006 que le problème de décider, pour un graphe d'entrée $G = (V, E)$, si $\chi(G) = \Gamma(G)$, est un problème coNP-complet [195]. Telle et Proskurowski proposent un algorithme en temps $O^*(n^{3 \cdot \mathbf{tw}(G)^2})$ pour déterminer le nombre grundy d'un graphe de largeur arborescente bornée [180].
Concernant l'Algorithmique Paramétrée, Zaker a montré que décider si $\Gamma(G) \geq k$ est un problème de la classe **XP** [195]. Havet et Sampaio ont montré en 2013 que le problème dual de décider si $\Gamma(G) \geq n - k$ est un problème FPT [110]. Cependant, connaître si le problème du Nombre Grundy est un problème soluble à paramètre fixé demeure une question ouverte.

2.2.2 Algorithme Exact

Nous construisons un algorithme de *Programmation Dynamique Classique* pour déterminer le nombre grundy d'un graphe. Cet algorithme s'exécute en temps $O^*(2.4423^n)$ et espace $O^*(2^n)$. Pour construire cet algorithme, nous aurons besoin des Lemmes 2.6 et 2.7 ainsi que du Théorème 2.8 suivants.

Lemme 2.6. *Soit $G = (V, E)$ un graphe, et soit $\varphi = \{V_1, \ldots, V_k\}$ une k-coloration de V.*
Si φ est une coloration grundy, alors pour tout $1 \leq i \leq k$: V_i est un stable maximal de $G[V - \cup_{j<i} V_j]$.

Démonstration. Soit $G = (V, E)$ un graphe, et soit $\varphi = \{V_1, \ldots, V_k\}$ une k-coloration grundy de V. Par définition, φ est une k-coloration propre. Les classes de couleurs sont donc stables dans G, et donc dans tout sous-graphe de G, en particulier pour tout $1 \leq i \leq k$: V_i est un stable dans $G[V - \cup_{j<i} V_j]$. En observant pour $i = k$ que $V - \cup_{j<i} V_j = V_k$, il vient que V_k est maximal dans $G[V - \cup_{j<i} V_j] = G[V_k]$. Montrons que la propriété est respectée pour toutes les autres classes de couleurs. Supposons par contradiction qu'il existe $1 \leq i < k$ tel que V_i ne soit pas un stable maximal de $G[V - \cup_{j<i} V_j]$. Alors il existe $y \in V - \cup_{j \leq i} V_j$ tel que $\{y\} \cup V_i$ est un stable de $G[V - \cup_{j<i} V_j]$. Puisque par hypothèse $\varphi(y) > i$ et n'est adjacent à aucun sommet de V_i, alors $y \in V$ n'est pas grundy, ce qui contredit que φ est une coloration grundy de V. □

Lemme 2.7. *Soit $G = (V, E)$ un graphe, et soit $\varphi = \{V_1, \ldots, V_k\}$ une k-coloration de V.*
Si $\forall\ 1 \leq i \leq k$: V_i est un stable maximal dans $G[V - \cup_{j<i} V_j]$, alors φ est une coloration grundy.

Démonstration. Soit $G = (V, E)$ un graphe, et soit $\varphi = \{V_1, \ldots, V_k\}$ une k-coloration de V qui satisfait $\forall\ 1 \leq i \leq k$: V_i est un stable maximal de $G[V - \cup_{j<i} V_j]$. Observons que φ est en particulier une coloration propre, car les classes de couleurs sont des ensembles stables dans des sous-graphes induits de G. Montrons alors que tous les sommets de V sont des sommets grundy de la k-coloration. Supposons par contradiction qu'il existe un sommet $x \in V$ qui ne soit pas grundy. Il existe alors un entier $1 \leq i \leq \varphi(x) - 1$ tel que x ne soit adjacent à aucun sommet de couleur i. Alors $x \in V - \cup_{j \leq i} V_j$, et $V_i \cup \{x\}$ est un ensemble stable dans G. Par conséquent, $V_i \cup \{x\}$ est également un ensemble stable dans $G[V - \cup_{j<i} V_j]$. Ceci contredit que V_i est maximal dans $G[V - \cup_{j<i} V_j]$. Nous avons donc montré que tout sommet $x \in V$ est grundy, ce qui achève notre preuve. □

Théorème 2.8. *Pour tout graphe $G = (V, E)$:*

$$\Gamma(G) = 1 + \max_{\{X \subseteq V \text{ stable maximal de } G\}} \Gamma(G[V - X])$$

Démonstration. Ce théorème s'appuie sur l'objectif de maximisation du nombre grundy d'un graphe, ainsi que sur les Lemmes 2.6 et 2.7 précédents, pour lesquels nous avons montré qu'une k-coloration $\{V_1, \ldots, V_k\}$ est grundy si et seulement si toutes les classes de couleur V_i sont des stables maximaux dans $G[V - \cup_{j<i} Vj]$. □

Algorithme 2.2: Algorithme de *Programmation Dynamique Classique* du Théorème 2.9 pour calculer le nombre grundy $\Gamma(G)$ d'un graphe $G = (V, E)$

input : $G = (V, E)$
output: $\Gamma(G)$
`// Étape d'initialisation`
1 $OPT[\emptyset] \leftarrow 0$;
`// Programmation dynamique sur les sous-ensembles de sommets dans un ordre croissant de taille`
2 **for** $i = 1$ **to** n **do**
3 **foreach** $X \subseteq V$, $|X| = i$ **do**
4 $OPT[X] \leftarrow 1 + \max_{\{Y \subseteq X \text{ stable maximal de } G[X]\}} OPT[X - Y]$;
5 **return** $\Gamma(G) = OPT[V]$;

Théorème 2.9. *Il y a un algorithme de Programmation Dynamique pour déterminer le nombre grundy d'un graphe $G = (V, E)$ en temps $O^*(2.4423^n)$ et espace $O^*(2^n)$.*

Démonstration. L'Algorithme 2.2 que nous présentons est très similaire à celui de Lawler pour déterminer le nombre chromatique d'un graphe. Comme c'est le cas de la plupart des algorithmes de *Programmation Dynamique Classique*, nous stockons les valeurs $\Gamma(G[X])$ de tous les sous-ensembles $X \subseteq V$ dans une table OPT. Pour stocker la valeur $OPT[X] = \Gamma(G[X])$ pour tous les sous-ensembles $X \subseteq V$, la table OPT dispose de 2^n valeurs, soit donc utilise un espace mémoire $O^*(2^n)$. La validité de l'Algorithme 2.2 pour calculer le nombre grundy d'un graphe s'appuie la récurrence obtenue dans le Théorème 2.8. Le temps d'exécution $T(n)$ de l'algorithme vérifie, pour un facteur polynomial $p : \mathbb{N} \to \mathbb{R}$, et en s'appuyant sur l'énumération des stables maximaux d'un graphe :

$$T(n) = \sum_{i=0}^{n} \binom{n}{i} p(i) \cdot 3^{i/3} \leq p(n) \cdot (1 + 3^{1/3})^n = O^*(2.4423^n)$$

□

2.3 Nombre Grundy Partiel

2.3.1 Introduction

Nombre Grundy Partiel. Le problème du Nombre Grundy Partiel est une relaxation du problème du Nombre Grundy. Soit $G = (V, E)$ un graphe. Nous rappelons que pour une k-coloration $\varphi = \{V_1, \ldots, V_k\}$ de V, un sommet $x \in V$ est un sommet grundy si pour tout $1 \leq i < \varphi(x) : N_G(x) \cap V_i = \emptyset$. Une k-coloration propre de V au sein de laquelle chaque classe de couleur contient au moins un sommet grundy est appelée une *k-coloration grundy partielle*, ou plus simplement une *k-∂coloration*. Le *Nombre Grundy Partiel* de G, noté $\partial\Gamma(G)$, est le plus grand entier $k \in \mathbb{N}^*$ pour lequel G admet une k-∂coloration.

> Nombre Grundy Partiel
> *Version* : *Optimisation*
>
> **Entrée** : $G = (V, E)$
> **Sortie** : Déterminer le plus grand entier $k = \partial\Gamma(G) \in \mathbb{N}^*$ pour lequel G admet une k-coloration propre de V où chaque classe de couleur contient au moins un sommet grundy.

Le nombre grundy partiel est défini pour tout graphe G. ♣

La notion de nombre grundy partiel d'un graphe a été introduite par Erdős, Hedetniemi et al. en 2003 [73]. Shi, Goddard, Hedetniemi et al. ont montré en 2005 que le problème du Nombre Grundy Partiel est NP-difficile, même lorsque l'on se restreint aux graphes cordaux ou aux graphes bipartis [178]. Ils ont montré de plus que pour tout $k \in \mathbb{N}^*$ fixé, décider si $\Gamma(G) \geq k$ est un problème NP-complet même si l'on se restreint aux graphes cordaux. Finalement, ils proposent un algorithme en temps linéaire pour déterminer le nombre grundy partiel d'un arbre.
Toute ∂coloration est en particulier une coloration grundy, la relation $\Gamma(G) \leq \partial\Gamma(G)$ est naturelle pour tout graphe G. En parallèle à son introduction, Erdős, Hedetniemi et al. ont également établi d'autres relations du nombre grundy partiel avec d'autres paramètres de graphe liés aux problèmes de coloration [73]. A notre connaissance, il n'existe pas d'algorithme exact exponentiel pour calculer le nombre grundy partiel d'un graphe.

2.3.2 Algorithme Exact

Nous construisons dans cette Sous-Section un algorithme de *Programmation Dynamique Classique* en temps $O^*(4^n)$ pour calculer le nombre grundy partiel d'un graphe. Nous commençons par établir le Lemme 2.10 suivant.

Lemme 2.10. *Soit $G = (V, E)$ et soit $\varphi = \{V_1, \dots, V_k\}$ une k-coloration de V. Alors φ est une k-∂coloration de G si et seulement si pour tout $0 \leq \ell \leq k-1$: $\varphi(\ell) = \{V_{\ell+1}, \dots, V_k\}$ est une $(k-\ell)$-∂coloration de $G[V - \cup_{i \leq \ell} V_i]$.*

Démonstration. Pour $\ell = 0$, il est clair que si $\varphi(0)$ est une k-∂coloration de $G[V - \cup_{i \leq \ell} V_i]$, alors $\varphi = \varphi(0)$ est une k-∂coloration de G. Soit $\varphi = \{V_1, \dots, V_k\}$ une k-∂coloration de G, et soit $0 \leq \ell \leq k-1$. Par définition d'une k-∂coloration et des sommets grundys, $\forall\ 1 \leq i \leq k : \exists\ x_i \in V_i$ tel que $\forall\ 1 \leq j < i$, $\exists\ y_j \in V_j$ satisfaisant $\{x_i, y_j\} \in E$. En particulier, cette propriété reste valable pour tout $i > \ell$ et pour tout $\ell < j < i$. Il est clair alors que $\varphi(\ell)$ constitue une $(k-\ell)$-coloration propre avec les mêmes sommets grundys que φ dans $V - \cup_{i \leq \ell} V_i$. Ainsi donc, si φ une k-∂coloration de G, alors pour tout $0 \leq \ell \leq k-1$: $\varphi(\ell)$ est une $(k-\ell)$-∂coloration de $G[V - \cup_{i \leq \ell} V_i]$. □

Afin de construire notre algorithme, nous avons besoin d'introduire différentes notions. Soit un sous-ensemble de sommets de $W \subseteq V$. Nous appelons *k-∂base* du sous-graphe induit $G[W]$ un sous-ensemble $\mathcal{G}_W = \{g_1, \dots, g_k\}$ de $k \in \mathbb{N}^*$ sommets de W. Une k-∂base $\mathcal{G}_W$ est dite *∂extensible* dans $G[W]$ s'il existe une k-∂coloration $\varphi = \{W_1, \dots, W_k\}$ de $G[W]$ et une permutation $\pi \in \mathfrak{S}_k$ qui vérifient pour tout $1 \leq i \leq k$: $g_i \in W_{\pi(i)}$ et g_i est un sommet grundy de φ.
Nous introduisons finalement une *relation $\mathcal{R}$* dont la satisfaction peut être décidée en temps polynomial $O(n^{O(1)})$. Pour $\mathcal{G} \subseteq V$, $g^* \in \mathcal{G}$ et $Y \subseteq V - \mathcal{G}$, on dit que le couple (g^*, Y) satisfait la relation $\mathcal{R}(\mathcal{G})$, et on note $(g^*, Y) \in \mathcal{R}(\mathcal{G})$, si les propriétés suivantes sont satisfaites :
◇ $Y^* = Y \cup \{g^*\}$ est un ensemble stable de G
◇ $\forall\ g \in \mathcal{G},\ g \neq g^*,\ N_G(g) \cap Y^* \neq \emptyset$

Soit $X \subseteq V$ un sous-ensemble de sommets de V, et soit $\mathcal{G} \subseteq V - X$ un sous-ensemble de sommets dans le complémentaire de X, avec $|\mathcal{G}| = k \in \mathbb{N}^*$. Pour construire notre algorithme de *Programmation Dynamique Classique*, nous associons à chaque couple $(X, \mathcal{G})$ un booléen $OPT[X, \mathcal{G}]$, qui vaut VRAI si et seulement si $X = \mathcal{G} = \emptyset$ ou si $\mathcal{G}$ est une k-∂base ∂extensible dans $G[X \cup \mathcal{G}]$.

Théorème 2.11. *Pour tous $X \subseteq V$ et $\mathcal{G} \subseteq V - X$, avec $|\mathcal{G}| \geq 1$:*

$$OPT[X, \mathcal{G}] = \text{FAUX} \vee \bigvee_{\substack{g^* \in \mathcal{G}, Y \subseteq X \\ (g^*, Y) \in \mathcal{R}(\mathcal{G})}} OPT[X - Y, \mathcal{G} - \{g^*\}]$$

De plus, le nombre grundy partiel $\partial\Gamma(G)$ de G est donné par :

$$\partial\Gamma(G) = n - \min_{\substack{X \subseteq V \\ OPT[X, V-X] = \text{VRAI}}} |X|$$

Démonstration. La récurrence définie dans ce théorème découle du Lemme 2.10, que l'on applique ici pour l'instance $G = G[X \cup \mathcal{G}]$, $k = |\mathcal{G}|$ et $\ell = 1$. Ainsi donc, avec $W = X \cup \mathcal{G}$, $G[W]$ admet une k-∂base $\mathcal{G}$, ∂extensible dans $G[W]$ en une k-∂coloration $\varphi = (W_1, \dots, W_k)$, avec $g^* \in W_1$ donc avec $(g^*, W_1) \in \mathcal{R}(\mathcal{G})$, si et seulement si $G[W - W_1]$ admet une $(k-1)$-∂base $\mathcal{G}' = \mathcal{G} - \{g^*\}$ qui soit ∂extensible dans $G[W - W_1]$. Cette récurrence permet de résoudre une instance du problème, pour décider si une ∂base est ∂extensible, à partir des solutions de ses sous-problèmes. □

Algorithme 2.3: Algorithme de *Programmation Dynamique Classique* du Théorème 2.12 pour calculer le nombre grundy partiel $\partial\Gamma(G)$ d'un graphe $G = (V, E)$

```
   input  : G = (V, E)
   output: ∂Γ(G)
   // Initialisation de la table OPT
1  OPT[∅, ∅] ← VRAI;
   // Parcours par ordre croissant de taille des sous-ensembles X ⊆ V
2  for i = 0 to n − 1 do
3      foreach X ⊆ V − G, |X| = i do
           // Parcours par ordre croissant de taille des sous-ensembles G ⊆ V − X
4          for j = 1 to k do
5              foreach G ⊆ V − X, |G| = j do
                   // Initialisation de la valeur de OPT[X, G]
6                  OPT[X, G] ← FAUX;
                   // Recherche des candidats pour construire W_1 (Thm. 2.11)
7                  foreach g* ∈ G do
8                      foreach Y ⊆ X do
9                          if (g*, Y) ∈ R(G) then
                               // Pour un candidat valable (g*, Y),
                               // Récupération de OPT[X − Y, G − {g*}] déjà calculé
10                             OPT[X, G] ← OPT[X, G] ∨ OPT[X − Y, G − {g*}];
   // Calcul de la valeur ∂Γ(G)
   // Parcours par ordre croissant de taille des sous-ensembles X ⊆ V
11 for i = 0 to n − 1 do
12     foreach X ⊆ V − G, |X| = i do
13         if OPT[X, V − X] = VRAI then
14             return ∂Γ(G) = n − I;
```

Théorème 2.12. *L'Algorithme 2.3 de Programmation Dynamique pour calculer le nombre grundy partiel $\partial\Gamma(G)$ d'un graphe $G = (V, E)$ s'exécute en temps $O^*(4^n)$ et espace $O^*(3^n)$.*

Démonstration. Observons que l'Algorithme 2.3 calcule non seulement le nombre grundy partiel $\partial\Gamma(G)$ d'un graphe G, mais décide également de la ∂extensibilité de toutes ∂bases possibles de tous les sous-graphes induits de G. Il est essentiel d'observer que lors de l'évaluation de la ligne 10, toutes les valeurs $OPT[X-Y, \mathcal{G}-\{g^*\}]$ des candidats (g^*, Y) valables sont déjà calculées, du fait de l'ordre dans lequel les sous-problèmes sont évalués au cours de l'exécution de l'algorithme.
Observons également que l'algorithme initialise correctement $OPT[\emptyset, \{v\}] = \text{VRAI}$ pour tout $v \in V$, de même que pour tout $X \subseteq V$, et pour tout $v \in V - X$: $OPT[X, \{v\}] = \text{VRAI}$ si et seulement si $X \cup \{v\}$ est un stable de G. La validité de l'Algorithme 2.3 est établie dans le Théorème 2.11.
L'algorithme stocke autant de valeurs dans la table OPT qu'il existe de couple $(X, \mathcal{G})$ satisfaisant $X \subseteq V$ et $\mathcal{G} \subseteq V - X$. L'espace mémoire $S(n)$ utilisé par l'Algorithme 2.3 vérifie alors :

$$S(n) = \sum_{i=0}^{n-1} \left\{ \binom{n}{i} \cdot \sum_{j=1}^{n-i} \binom{n-i}{j} \right\} \leq \sum_{i=0}^{n} \left\{ \binom{n}{i} \cdot \sum_{j=0}^{n-i} \binom{n-i}{j} \right\} \leq \sum_{i=0}^{n} \left\{ \binom{n}{i} \cdot 2^{n-i} \right\} \leq 3^n$$

Le temps d'exécution $T(n)$ de l'Algorithme 2.3 satisfait, à un facteur polynomial $n^{O(1)}$ près que nous ne détaillons pas :

$$\begin{aligned} T(n) &= n^{O(1)} \cdot \sum_{i=0}^{n-1} \left\{ \binom{n}{i} \cdot \sum_{j=1}^{n-i} \left[\binom{n-i}{j} \cdot j \cdot 2^i \right] \right\} + O^*(2^n) \\ &\leq n^{O(1)} \cdot \sum_{i=0}^{n} \left\{ \binom{n}{i} \cdot \sum_{j=0}^{n-i} \left[\binom{n-i}{j} \cdot j \cdot 2^i \right] \right\} + O^*(2^n) \\ &\leq n^{O(1)} \cdot \sum_{i=0}^{n} \left\{ \binom{n}{i} \cdot (n-i) \cdot 2^i \cdot \sum_{j=0}^{n-i} \left[\binom{n-i}{j} \right] \right\} + O^*(2^n) \\ &\leq n^{O(1)} \cdot n \cdot \sum_{i=0}^{n} \left\{ \binom{n}{i} \cdot 2^i \cdot 2^{n-i} \right\} + O^*(2^n) \\ &\leq n^{O(1)} \cdot 4^n + O^*(2^n) \\ &= O^*(4^n) \end{aligned}$$

□

2.4 Nombre a-Chromatique

2.4.1 Introduction

Nombre a-Chromatique. Le nombre a-chromatique a été introduit en 1967, et peut être vu comme une variante du nombre chromatique [108]. En effet, si le nombre chromatique d'un graphe $G = (V, E)$ est la plus petite taille d'une partition des sommets en stables telle qu'un graphe complet peut être obtenu par contraction des classes de couleurs, le nombre a-chromatique de G, noté $\chi_a(G)$, est la plus grande taille de ces partitions. Une k-coloration propre de V, pour laquelle le graphe complet K_k à k sommets peut être obtenu par contraction des classes de couleur, est appelé une *k-a-coloration*.

> *k-a-Coloration*
> *Version* : *Décision*
>
> **Entrée** : $G = (V, E)$, $k \in \mathbb{N}^*$
> **Sortie** : Existe-t-il une k-coloration propre de V telle que le graphe complet à k sommets peut être obtenu par contraction des classes de couleurs ?

Le *Nombre a-Chromatique* d'un graphe $G = (V, E)$, noté $\chi_a(G)$, est alors défini comme le plus grand entier $k \in \mathbb{N}$ tel que G admette une k-a-coloration. Ce paramètre est défini pour tout graphe, puisqu'une coloration propre optimale est une a-coloration. Simmons et al. ont montré en 1982 que tout graphe G vérifie la relation $\chi(G) \leq \Gamma(G) \leq \chi_a(G)$ [72], avec $\chi(G)$ et $\Gamma(G)$ respectivement le nombre chromatique et grundy de G. ♣

Le problème du Nombre a-Chromatique est NP-difficile [192]. Le nombre a-chromatique d'une union disjointe de chemins ou une union disjointe de cycles est connu [149, 139]. Il existe un algorithme en temps polynomial pour déterminer le nombre a-chromatique d'une forêt de degré borné [30]. Cependant, déterminer le nombre a-chromatique d'un cograph, d'un graphe d'intervalles ou même d'un arbre demeure un problème NP-difficile [21, 29]. Farber et al. ont montré que le problème du Nombre a-Chromatique est dans la classe **FPT** [75]. Décider si le nombre a-chromatique d'un graphe est au moins k peut être résolu en temps $O^*(k^{3 \cdot 2^{k-3}+k-3})$. Ce résultat s'appuie sur le Théorème de Máté sur le nombre a-chromatique de graphes irréductibles [151].

2.4.2 Programmation Dynamique

A notre connaissance, il n'existe pas à l'heure actuelle d'algorithme en temps *single-exponentiel* pour déterminer le nombre a-chromatique d'un graphe, même lorsqu'il s'agit d'un arbre. Dans cette Sous-Section, nous construisons un algorithme de *Programmation Dynamique Arborescente* sur une décomposition en chemin jolie d'un graphe. Lorsqu'on l'applique à un graphe de genre borné, le temps d'exécution de l'algorithme est, à un facteur sous-exponentiel près, en temps *single-exponentiel*. En particulier, l'algorithme s'exécute, $\forall\ \epsilon > 0$, respectivement en temps $O^*((2+\epsilon)^n)$ et $O^*((8+\epsilon)^n)$ lorsque le graphe d'entrée est un arbre ou un graphe planaire.

Théorème 2.13. *Il y a un algorithme de Programmation Dynamique qui s'exécute en temps et espace $O^*(k^{\mathbf{pw}(G)} \cdot 2^{k^2/2})$ pour décider si un graphe $G = (V, E)$ admet une k-a-coloration.*

Démonstration. Soit $G = (V, E)$ un graphe, et soit $(X_1, \ldots, X_r)$ une décomposition en chemin jolie de G. Un couple $((k_1, \ldots, k_{l(i)}), E_k)$, formé d'un vecteur $(k_1, \ldots, k_{l(i)}) \in \{1, \ldots, k\}^l(i)$ et d'un sous-ensemble d'arêtes $E_k \subseteq \{1, \ldots, k\}^2$ d'arêtes du graphe complet K_k à k sommets, est une caractéristique d'un nœud $1 \leq i \leq r$ de la décomposition, s'il existe une k-coloration propre de V_i dans $G[V_i]$ telle que $\forall\ 1 \leq p \leq l(i)\ :\ \varphi(x_p) = k_p$, et telle que $\forall\ u \neq v \in \{1, \ldots, k\}\ :\ \{u, v\} \in E_k \Leftrightarrow \exists\ x_u, x_v \in V_i,\ \varphi(x_u) = u,\ \varphi(x_v) = v$, et $\{x_u, x_v\} \in E$. Contrairement aux caractéristiques utilisées pour l'algorithme de *Programmation Dynamique* pour le problème de k-coloration présentées en Sous-Section 2.1.1, les caractéristiques utilisées ici sont augmentées d'une trace sur les relations d'adjacence qui régissent les classes de couleur. Nous pouvons observer que G admet une k-a-coloration si et seulement si le nœud r de la décomposition admet une caractéristique $((k(v)), E_k)$, avec $(\{1, \ldots, k\}, E_k) = K_k$. De plus, pour tout nœud $1 \leq i \leq r$ de la décomposition en chemin jolie, les caractéristiques $\Gamma(i)$ de i vérifient $|\Gamma(i)| \leq k^{\mathbf{pw}(G)} \cdot 2^{k(k-1)/2}$.
De la même manière que pour l'Algorithme 2.1, on note $S_i(c) = \{x_p \in X_i, 1 \leq p \leq l(i)\ :\ k_p = c\}$ l'ensemble des sommets de X_i de couleur $c \in \{1, \ldots, k\}$ selon la caractéristique. L'Algorithme 2.4, pour décider l'existence d'une k-a-coloration, associe l'ensemble des caractéristiques $\Gamma(i)$ d'un nœud $1 \leq i \leq r$ en fonction des caractéristiques de ses sous-problèmes déjà calculées.
La validité de l'Algorithme 2.4 se montre de la même manière que la validité de l'Algorithme 2.1, en s'appuyant essentiellement sur le Lemme 2.1. Pour analyser le temps d'exécution, on munit l'Algorithme 2.4 d'un codage d'un vecteur $(k_1, \ldots, k_{l(i)})$ en base k décalé d'un codage d'un sous-ensemble d'arêtes E_k de K_k en base 2, qui à toute caractéristique $X = ((k_1, \ldots, k_{l(i)}), E_k) \in \Gamma(i)$ associe un entier $c(X) = 2^{k(k-1)/2} \cdot \sum_{i=1}^{l(i)} k_i \cdot k^{i-1} + \sum_{\{(i,j) \in E_k,\ i<j\}} 2^{j-2+\sum_{u=1}^{i-1}(k-1-u)}$. Ce code transforme l'ensemble des caractéristiques $\Gamma(i)$ de chaque nœud $1 \leq i \leq r$ en ensemble ordonné. La fonction de codage c et sa fonction de décodage s'exécutent en temps $O(l(i) + |E_k|)$. On associe ensuite à chaque nœud $1 \leq i \leq r$ de la décomposition une structure de données similaire à celle utilisée pour l'Algorithme 2.1 du Théorème 2.5 afin de stocker ses caractéristiques. Ceci permet, pour un nœud $1 \leq i \leq r$, la recherche et l'insertion d'une caractéristique en temps $O(\log |\Gamma(i)|)$, et le parcours de toutes les caractéristiques en temps $O(|\Gamma(i)|)$. Il nous reste à analyser le temps de construction des caractéristiques $\Gamma(i)$ d'un nœud $1 \leq i \leq r$ en fonction de son type dans la décomposition :

- ◇ Si i est une FEUILLE, donc si $i = 1$: L'algorithme effectue k insertions distinctes dans $\Gamma(i)$, et s'exécute donc en temps $O(k \log k)$.
- ◇ Si i est de type FORGET : L'algorithme effectue un parcours des caractéristiques de $\Gamma(j)$, ainsi que $|\Gamma(j)|$ recherches et $|\Gamma(i)|$ insertions dans $\Gamma(i)$. L'algorithme s'exécute donc en temps $O(k^{l(j)} \cdot 2^{k(k-1)/2} + (k^{l(j)} + k^{l(i)}) \cdot 2^{k(k-1)/2} \log (k^{l(i)} \cdot 2^{k(k-1)/2})) = O^*(k^{\mathbf{pw}(G)} \cdot 2^{k^2/2})$.
- ◇ Si i est de type INTRODUCE : L'algorithme effectue un parcours des caractéristiques de $\Gamma(j)$, ainsi que $k \cdot |\Gamma(j)|$ recherches et $|\Gamma(i)|$ insertions dans $\Gamma(i)$. En observant que $l(j) + 1 = l(i) \leq \mathbf{pw}(G)$,

Algorithme 2.4: Algorithme de *Programmation Dynamique* du Théorème 2.13 pour la version décisionnelle du problème de k-a-Coloration qui associe à chaque nœud $1 \leq i \leq r$ d'une décomposition en chemin jolie $(X_1, \ldots, X_r)$ d'un graphe $G = (V, E)$, l'ensemble de ses caractéristiques $\Gamma(i)$

input : $G = (V, E)$, $(X_1, \ldots, X_r)$, et $1 \leq i \leq r$
output: $\Gamma(i)$
// Étape d'initialisation
$\Gamma(i) \leftarrow \emptyset$;
// Différents cas selon le type du nœud i
if *i est de type* introduce **then**
  Soit $X_i = \{x_1, \ldots, x_{p-1}, v, x_p, \ldots, x_{l(j)}\}$, pour un certain $1 \leq p \leq l(j) + 1$;
  foreach $(k_1, \ldots, k_{l(j)}) \in \Gamma(j)$ **do**
    for $k(v) = 1$ **to** k **do**
      if $S_i(k(v)) = S_j(k(v)) \cup \{v\}$ *est un stable de* G **then**
        // ξ représente les relations d'adjacence de classes de couleur générées grâce à v
        $\xi \leftarrow \cup_{1 \leq u \leq k} \{\{u, k(v)\} \ : \ \exists\ 1 \leq q \leq l(j),\ k_q = u,\ \{x_q, v\} \in E\}$;
        $\Gamma(i) \leftarrow \Gamma(i) \cup \{((k_1, \ldots, k_{p-1}, k(v), k_p, \ldots, k_{l(j)}), E_k \cup \xi)\}$;
if *i est de type* forget **then**
  Soit $X_j = \{x_1, \ldots, x_{p-1}, v, x_p, \ldots, x_{l(i)}\}$, pour un certain $1 \leq p \leq l(i) + 1$;
  foreach $((k_1, \ldots, k_{p-1}, k(v), k_p, \ldots, k_{l(i)}), E_k) \in \Gamma(j)$ **do**
    $\Gamma(i) \leftarrow \Gamma(i) \cup \{((k_1, \ldots, k_{p-1}, k_p, \ldots, k_{l(j)}), E_k)\}$;
else
  // Ici $i = 1$
  for $j = 1$ **to** k **do**
    $\Gamma(i) \leftarrow \Gamma(i) \cup \{((j), \emptyset)\}$;
return $\Gamma(i)$;

l'algorithme s'exécute donc en temps $O^*(2^{k(k-1)/2} \cdot (k^{l(j)} + (k \cdot k^{l(j)} + k^{l(i)}) \cdot \log(k^{l(i)} \cdot 2^{k(k-1)/2}))) = O^*(k^{\mathbf{pw}(G)} \cdot 2^{k^2/2})$.

En résumé, nous avons donc montré que l'Algorithme 2.4 s'exécute pour tout nœud $1 \leq i \leq r$ de la décomposition en chemin jolie en temps $O^*(k^{\mathbf{pw}(G)} \cdot 2^{k^2/2})$. Avec $r \leq 2n$, l'Algorithme 2.4 décide la k-acolorabilité d'un graphe $G = (V, E)$ en temps et espace $O^*(k^{\mathbf{pw}(G)} \cdot 2^{k^2/2})$. □

Théorème 2.14. *L'Algorithme 2.4 pour le Théorème 2.13 calcule le nombre a-chromatique d'un graphe $G = (V, E)$ quelconque en temps et espace $O^*(2^{m + \mathbf{pw}(G) \cdot \log m})$*

Démonstration. L'Algorithme 2.4 pour le Théorème 2.13 décide, pour $k \in \mathbb{N}$, si un graphe $G = (V, E)$ est k-a-colorable en temps et espace $O^*(k^{\mathbf{pw}(G)} \cdot 2^{k^2/2})$. Observons que pour tout graphe G, si $k(k-1)/2 > m$, alors G ne peut pas être k-a-colorable car il ne contient pas suffisement d'arêtes pour satisfaire les relations d'adjacence entre les classes de couleurs requises par une k-a-coloration. Tout graphe G vérifie donc l'inégalité $\chi_a(G)^2 - \chi_a(G) - 2m \leq 0$, qui conduit à la relation $\chi_a(G) \leq \sqrt{2m}$. Pour déterminer la valeur $\chi_a(G)$, nous pouvons appliquer l'Algorithme 2.4 du Théorème 2.13 sur toutes les valeurs possibles de k, afin de décider le plus grand entier k pour lequel G est k-a-colorable. Cette approche permet alors de déterminer le nombre a-chromatique de

G en temps et espace $O^*(\chi_a(G)^{\mathbf{pw}(G)} \cdot 2^{\chi_a(G)^2/2})$. En observant que pour tout $x \in \mathbb{R}_+^*$, $x = 2^{\log x}$, et avec la relation $\chi_a(G) \leq \sqrt{2m}$, l'Algorithme 2.4 permet alors de calculer le nombre a-chromatique de G en temps et espace $O^*(2^{\mathbf{pw}(G)\cdot\log(2m)} \cdot 2^m) = O^*(2^{m+\mathbf{pw}(G)\cdot\log m})$. □

Le Théorème 2.13, conjugué avec le Théorème 2.14, conduisent alors au Corollaire 2.15 suivant.

Corollaire 2.15. *Il y a un algorithme single-exponentiel de Programmation Dynamique qui, à un facteur sous-exponentiel près, calcule le nombre a-chromatique :*
- ◇ *d'un arbre en temps et espace* $O^*(2^n)$
- ◇ *d'une forêt en temps et espace* $O^*(2^n)$
- ◇ *d'un graphe planaire en temps et espace* $O^*(8^n)$
- ◇ *d'un graphe de genre borné par* $\overline{g} \in \mathbb{N}$ *en temps et espace* $O^*(8^n)$

Démonstration. Soit $G = (V, E)$ un graphe. Nous appliquons le Théorème 2.14.
- ◇ Si G est un arbre ou une forêt, on a $m \leq n-1$ et $\mathbf{pw}(G) \leq 2/\log 3 \cdot \log n$. L'algorithme 2.4 s'exécute alors en temps et espace $O^*(2^n \cdot 2^{\log^2 n})$ qui est, avec $\log^2 n = o(n)$, à un facteur sous-exponentiel près, $O^*(2^n)$.
- ◇ Si G est un graphe planaire ou de genre borné par $\overline{g} \in \mathbb{N}$, on a $m \leq 3n - 6 + 6\overline{g}$, et $\mathbf{tw}(G) \leq (3.19 + 6\sqrt{\overline{g}}) \cdot \sqrt{n}$. Pour tout graphe G, on a $\mathbf{pw}(G) \leq 2/\log 3 \cdot \log n \cdot \mathbf{tw}(G)$. Le temps d'exécution et l'espace mémoire utilisé par l'Algorithme 2.4 étant $O^*(\chi_a(G)^{\mathbf{pw}(G)} \cdot 2^{\chi_a(G)^2/2})$, avec $\chi_a(G) \leq \sqrt{2m}$, il peut alors être réécrit $O^*(2^m \cdot 2^{\mathbf{pw}(G)\cdot\log m/2})$. Puisque G est de genre borné, on pose la constante réelle $\varepsilon(\overline{g}) = 2^{(3.19+6\sqrt{\overline{g}})/\log 3} > 1$. L'algorithme 2.4 s'exécute alors en temps et espace $O^*(2^n \cdot \varepsilon(\overline{g})^{\sqrt{n}\cdot\log^2 n})$ qui est, au facteur $\varepsilon(\overline{g})^{o(n)}$ sous-exponentiel près, $O^*(8^n)$.

□

2.5 Nombre b-Chromatique

2.5.1 Introduction

Nombre *b*-Chromatique. Une *b-coloration* $\varphi = \{V_1, \ldots, V_k\}$ d'un graphe $G = (V, E)$ est une k-coloration propre de V, telle que chaque classe de couleur contienne un sommet b_i voisin avec toutes les autres classes de couleur. Formellement, une k-coloration propre $\{V_1, \ldots, V_k\}$ de V est donc une b-coloration si $\forall\ 1 \leq i \leq k$, $\exists\ b_i \in V_i$ tel que $\forall\ 1 \leq j \neq i \leq k : N_G(b_i) \cap V_j \neq \emptyset$. Un sommet $x \in V$ voisin avec toutes les autres classes de couleur, c'est-à-dire satisfaisant $\forall\ 1 \leq i \neq \varphi(x) \leq k : N_G(x) \cap V_i \neq \emptyset$, est appelé un *b-sommet*.

NOMBRE *b*-CHROMATIQUE
VERSION : OPTIMISATION

Entrée : $G = (V, E)$
Sortie : Déterminer le plus grand entier $k = \chi_b(G) \in \mathbb{N}^*$ tel que G admette une k-coloration propre où toutes les classes de couleurs contiennent un b-sommet

Le *Nombre b-Chromatique* de G, denoté $\chi_b(G)$, est le plus grand entier $k \in \mathbb{N}^*$ tel que G admette une b-coloration de taille k. Ce paramètre est défini pour tout graphe, puisqu'une coloration propre optimale est une b-coloration. Cependant, un graphe G n'admet pas toujours une b-coloration de taille k pour toutes les valeurs de k, $\chi(G) \leq k \leq \chi_b(G)$. ♣

Le problème du Nombre b-Chromatique est relié à d'autres problèmes de coloration que nous avons étudié jusqu'ici. Par exemple, les b-sommets se présentent comme une extension des sommets grundys à toutes les classes de couleur. Les b-sommets sont des sommets grundy, mais contrairement à la coloration grundy, l'ordre des classes de couleur dans une b-coloration n'a pas d'importance. Egalement, ce problème peut être vu comme une variante du problème du Nombre a-Chromatique. En particulier, toute b-coloration est une a-coloration. En effet, le problème du Nombre a-Chromatique demande de déterminer le plus grand entier $k \in \mathbb{N}^*$ tel que G admette une k-coloration propre de V au sein de laquelle il y ait toujours au moins une arête entre deux classes de couleurs. Il vient donc aisément la relation $\chi_b(G) \leq \chi_a(G)$ satisfaite pour tout graphe G.

Le problème du Nombre b-Chromatique a été introduit par Irving et Manlove en 1999, qui ont montré que le problème est NP-difficile [118]. Le problème a été longuement étudié en Théorie des Graphes, par exemple sur certaines classes de graphes [129, 26, 120], ou pour déterminer des bornes [131], en particulier pour ses applications concernant le clustering dans les bases de données [70], ou la reconnaissance automatique de documents [93].

D'un point de vue algorithmique, on sait que décider si un graphe admet une b-coloration de taille k est NP-complet pour tout entier $k \geq 3$ fixé, même pour les bipartis connexes [131] et les cordaux connexes [109]. Au passage, observons que $\chi_b(G) = 1$ si et seulement si $E = \emptyset$, et que le problème de décider si $\chi_b(G) = 2$ est soluble en temps polynomial [131]. Autour du nombre b-chromatique, d'autres propriétés des b-colorations ont été examinées. En particulier, le *b-spectre* d'un graphe est l'ensemble des entiers k telle qu'une b-coloration de taille k existe, et un graphe est dit *b-continu* si son b-spectre est formé par un ensemble d'entiers consécutifs [15].

Pour l'Algorithmique Paramétrée, une question ouverte est de savoir si le problème de décider si $\chi_b(G) \geq k$ est un problème de la classe **XP**. La version duale de ce problème, décider si $\chi_b(G) \leq |V| - k$, est lui dans la classe **FPT** [110].

2.5.2 Partition Etiquetée

Dans cette Sous-Section, nous introduisons le problème généralisé de Partition Étiquetée, et nous montrons comment il peut être résolu en utilisant le principe d'*Inclusion-Exclusion*. Nous utiliserons les résultats de cette Sous-Section pour construire en Sous-Section 2.5.3 un algorithme exact pour le problème du Nombre b-Chromatique.

Partition Étiquetée. Le problème de Partition Étiquetée est un problème de décision plus général que la version décisionnelle du problème de Partition d'Ensembles. Pour le problème de Partition Étiquetée, l'entrée est donc constituée d'un univers fini $\mathcal{U}$ d'objets quelconques, d'une famille $\mathcal{F} \subseteq 2^{\mathcal{U}}$ de sous-ensembles de $\mathcal{U}$, et de $r \in \mathbb{N}$ étiquettes. Ces étiquettes sont attribuées à chaque sous-ensemble X de $\mathcal{F}$, si bien que chacun dispose de son propre ensemble d'étiquettes $\ell(X)$ parmi les r disponibles, c'est-à-dire $\forall\ X \in \mathcal{F} : \ell(X) \subseteq \{1, \ldots, r\}$. Une *k-partition étiquetée* de $\mathcal{U}$ par $\mathcal{F}$ est définie comme une k-partition $\{X_1, \ldots, X_k\}$ telle que pour tout $1 \leq i \leq k : X_i \in \mathcal{F}$ et $i \in \ell(X_i)$ si $i \leq r$. Autrement dit, une telle k-partition étiquetée est une k-partition de $\mathcal{U}$ par $\mathcal{F}$ au sein de laquelle l'étiquette i doit avoir été attribuée au sous-ensemble $X_i \in \mathcal{F}$, ceci pour tout $1 \leq i \leq r$. Le problème Partition Étiquetée consiste alors, pour une entrée considérée, à décider l'existence d'une k-partition étiquetée.

PARTITION ÉTIQUETÉE
VERSION : *DÉCISION*

Entrée : $\mathcal{U}$, $\mathcal{F} \subseteq 2^{\mathcal{U}}$, $k \in \mathbb{N}^*$, $r \in \mathbb{N}_{\leq k}$ étiquettes, et $\forall\ X \in \mathcal{F} : \ell(X) \subseteq \{1, \ldots, r\}$
Sortie : Existe-t-il une k-partition étiquetée de $\mathcal{U}$ par $\mathcal{F}$ sur les r étiquettes ?

Pour ce problème, nous pourrons considérer que pour un entier $1 \leq i \leq r$, les sous-ensembles $X \in \mathcal{F}$ satisfaisant $i \in \ell(X)$ constituent ensemble une sous-famille $\mathcal{F}_i$ de $\mathcal{F}$. Aussi nous proposons une formulation alternative au problème de PARTITION ÉTIQUETÉE, en considérant non plus des étiquettes mais r sous-familles $\mathcal{F}_i$ de $\mathcal{F}$. Etant donnés l'univers $\mathcal{U}$, la famille $\mathcal{F} \subseteq 2^{\mathcal{U}}$, et $r \leq k$ sous-familles $\mathcal{F}_i$ de $\mathcal{F}$, l'objectif du problème PARTITION ÉTIQUETÉE consiste alors à décider l'existence d'une k-partition $\{X_1, \ldots, X_k\}$ de $\mathcal{U}$ par $\mathcal{F}$ telle que pour tout $1 \leq i \leq r : X_i \in \mathcal{F}_i$.

PARTITION ÉTIQUETÉE
VERSION : *DÉCISION*

Entrée : $\mathcal{U}$, $\mathcal{F} \subseteq 2^{\mathcal{U}}$, $k \in \mathbb{N}^*$, et $r \in \mathbb{N}_{\leq k}$ sous-familles $\mathcal{F}_i \subseteq \mathcal{F}$, $1 \leq i \leq r$
Sortie : Existe-t-il une k-*partition étiquetée* de $\mathcal{U}$ par $\mathcal{F}$ sur les sous-familles $\mathcal{F}_i$, $1 \leq i \leq r$, c'est-à-dire une k-partition $\{F_1, \ldots, F_k\}$ de $\mathcal{U}$ par $\mathcal{F}$ telle que $\forall\ 1 \leq i \leq r : F_i \in \mathcal{F}_i$?

Les deux formulations sont équivalentes, et nous pourrons alterner arbitrairement notre choix de la formulation du problème afin d'établir nos résultats. ♣

Observons que nous pouvons considérer une généralisation de ce problème où certaines étiquettes doivent apparaître plus d'une fois dans la partition. Cette généralisation peut facilement se réduire au problème initial de partition étiquetée en dupliquant les étiquettes multiples dans l'ensemble des étiquettes autant de fois que nécessaire.

Proposition 2.16. *Le problème de Partition Étiquetée est* NP*-complet.*

Démonstration. Le problème de PARTITION ÉTIQUETÉE est une généralisation du problème de graphe k-COLORABILITÉ [94]. Considérons le problème de k-COLORATION PROPRE, qui est un problème connu pour être NP-complet même dans les graphes sans ensembles stables de taille 4 [94]. Etant donné un tel graphe $G = (V, E)$ et un entier k, nous définissons une instance du problème de PARTITION ÉTIQUETÉE où $\mathcal{U} = V$, $\mathcal{F}$ est l'ensemble des stables de G, $r = k$ et chaque F dans $\mathcal{F}$ a toutes les étiquettes possibles ($\ell(F) = \{1, 2, \ldots, r\}$ ou, alternativement, $\mathcal{F}_i = \mathcal{F}$). La réduction est polynomiale puisque G n'a pas de stable de taille 4, donc l'ensemble $\mathcal{F}$ est de taille polynomiale et peut être calculé en temps polynomial. Il est facile de voir que l'instance du problème de COLORATION PROPRE et l'instance du problème de PARTITION ÉTIQUETÉE sont équivalentes. L'appartenance à **NP** de ce dernier problème est directe. □

Nous montrons maintenant comment résoudre le problème de PARTITION ÉTIQUETÉE grâce au principe d'*Inclusion-Exclusion*. Nous allons réduire une instance de ce problème à une instance du problème de SOMME DES PARTITIONS PONDÉRÉES, que nous résoudrons par l'intermédiaire du Théorème 1.6.

Théorème 2.17. *Si l'appartenance dans $\mathcal{F}$ et dans chaque $\mathcal{F}_i$ peut être calculée en temps $O(|\mathcal{U}|^{O(1)})$, alors le problème de partition étiquetée peut être résolu en temps et espace $O^*(2^{|\mathcal{U}|})$.*

Démonstration. Soit $\mathcal{P}$ une instance du problème de Partition Étiquetée telle que l'appartenance dans $\mathcal{F}$ et dans chaque $\mathcal{F}_i$ peut être calculée en temps polynomial $O(|\mathcal{U}|^{O(1)})$. Réduisons $\mathcal{P}$ à une instance de problème de Somme de Partition Pondérée.
Pour commencer, nous créons des étiquettes artificielles i pour chaque i compris entre $r+1$ et k, définies par leur famille $\mathcal{F}_i = \mathcal{F}$. Cela transforme l'instance $\mathcal{P}$ en une nouvelle instance $\mathcal{P}'$ du même problème de Partition Étiquetée. Trivialement, $\mathcal{P}$ admet une solution si et seulement si $\mathcal{P}'$ admet une solution.
Nous introduisons à présent pour chaque $1 \leq i \leq k$, une fonction $f_i : 2^{\mathcal{U}} \to \{0,1\}$, où pour tout $X \subseteq \mathcal{U}$: $f_i(X) = 1 \Leftrightarrow X \in \mathcal{F}_i$. Les fonctions f_i peuvent donc être vues comme les symboles de Kronecker des sous-familles $\mathcal{F}_i$ de $\mathcal{F}$. Par hypothèse, f_i est donc calculable en temps $O(|\mathcal{U}|^{O(1)})$.
Nous définissons ensuite une fonction de pondération f de k-tuples de $\mathcal{F}$ comme produit tensoriel des fonctions f_i.

$$\begin{array}{rccl} f & : & 2^{\mathcal{U}} \times \ldots \times 2^{\mathcal{U}} & \to & \{0,1\} \\ & & S = \{X_1, \ldots, X_k\} & \mapsto & f(S) = \prod_{i=1}^{k} f_i(X_i) \end{array}$$

Nous pouvons vérifier qu'une partition $S = \{X_1, \ldots, X_k\}$ de V est une solution de $\mathcal{P}'$ si et seulement si $f(S) = 1$. Ainsi donc, $\mathcal{P}$ possède une solution si et seulement si $\sum_S f(S) \geq 1$, où $\sum_S f(S)$ représente la somme de la fonction de pondération f sur toutes les k-partitions S de $\mathcal{U}$.
L'univers $\mathcal{U}$, les fonctions f_i et de la fonction de pondération f, constituent ensemble une instance $\mathcal{P}''$ du problème de Somme de Partition Pondérée, et nous avons montré que $\mathcal{P}$ admet une solution si et seulement si la somme $\sum_S f(S) = p_k(f)$ pour $\mathcal{P}''$ vérifie $p_k(f) > 1$. En conséquence, nous pouvons appliquer le Théorème 1.6 pour calculer $p_k(f)$ pour l'instance $\mathcal{P}''$ du problème de Somme de Partition Pondérée, et donc pour résoudre l'instance $\mathcal{P}$ du problème de Partition Étiquetée, en temps $O^*(k \cdot 2^{|\mathcal{U}|} \cdot |\mathcal{U}|^{O(1)}) = O^*(2^{|\mathcal{U}|})$ et espace $O^*(2^{|\mathcal{U}|})$. □

2.5.3 Algorithme Exact

Dans le but de déterminer si un graphe $G = (V, E)$ admet b-coloration de taille $k \in \mathbb{N}^*$, nous construisons un algorithme qui de manière informelle répète essentiellement deux étapes. La première étape consiste à deviner un ensemble $\mathcal{B}$ de b-sommets, un pour chaque classe de couleur de la b-coloration. La deuxième étape consiste ensuite à déterminer si G admet une k-coloration propre φ telle que $\mathcal{B}$ soit inclus dans l'ensemble des b-sommets de φ.
Nous appelons *b-base* un ensemble $\mathcal{B} = \{b_1, \cdots, b_k\}$ de $k \in \mathbb{N}^*$ sommets de V. Une b-base est dite *b-extensible* s'il existe une b-coloration φ de G de taille k, qui satisfait pour tout $1 \leq i \leq k : \varphi(b_i) = i$ et b_i est un b-sommet.

Lemme 2.18. *Il y a un algorithme qui décide si une b-base $\mathcal{B} \subseteq V$ d'un graphe $G = (V, E)$ est b-extensible en temps et espace $O^*(2^{n-|\mathcal{B}|})$.*

Démonstration. Soit une b-base $\mathcal{B} = \{b_1, ..., b_k\} \subseteq V$. Nous souhaitons décider si $\mathcal{B}$ est b-extensible. Nous exprimons ce problème comme un problème de Partition Étiquetée, avec l'instance suivante :
◇ $\mathcal{U} = V - \mathcal{B}$;
◇ $\mathcal{F} \subseteq \mathcal{U}$ est la famille des ensembles stables de $G[V - \mathcal{B}]$.
◇ Il y a $r = k$ étiquettes. Un ensemble $X \in \mathcal{F}$ possède l'étiquette $i \in \{1, \cdots, k\}$, ou de manière équivalente $X \in \mathcal{F}_i$, si X satisfait les propriétés suivantes :

1. $Y_i = X \cup \{b_i\}$ est un ensemble stable de G
2. Tous les autres b-sommets $\mathcal{B}$ sont adjacents à au moins un sommet de Y_i.
   Soit donc $\forall\ b \in \mathcal{B},\ b \neq b_i : \exists\ x \in Y_i\ /\ \{x, b\} \in E$.

Par définition et par cette construction, $\mathcal{B}$ est b-extensible si et seulement s'il existe une k-partition étiquetée de $\mathcal{U}$ par $\mathcal{F}$. De plus, avec $|\mathcal{B}| = k$ fixé, pour tout $X \in \mathcal{U}$, l'appartenance de X à $\mathcal{F}$ ou à une sous-famille $\mathcal{F}_i$ peut être décidée en temps polynomial $O(|\mathcal{U}|^{O(1)})$. Ainsi, pour décider si $\mathcal{B}$ est b-extensible, nous pouvons appliquer le Théorème 2.17 sur l'instance du problème de Partition Étiquetée que nous avons construite. En observant que $|\mathcal{U}| = |V \backslash \mathcal{B}| = |V| - |\mathcal{B}| = n - |\mathcal{B}|$, décider si $\mathcal{B}$ est b-extensible s'exécute donc en temps et espace $O^*(2^{n-|\mathcal{B}|})$. □

Théorème 2.19. *Il y a un algorithme qui decide si un graphe admet une b-coloration de taille $k \in \mathbb{N}_{\geq 3}$ en temps et espace $O^*(2^n)$.*

Démonstration. L'algorithme énumère tous les sous-ensembles $\mathcal{B} \subseteq V$ de sommets de taille k du graphe, et décide si $\mathcal{B}$ est une b-base b-extensible. En utilisant le Lemme 2.18, l'algorithme s'exécute en temps $O^*(\binom{n}{k} \cdot 2^{n-k}) = O^*(n^k \cdot 2^{n-k}) = O^*(2^n)$ et espace $O^*(2^{n-k}) = O^*(2^n)$. □

Théorème 2.20. *Il y a un algorithme pour déterminer le b-spectre, décider la b-continuité, et calculer le nombre b-chromatique $\chi_b(G)$ d'un graphe $G = (V, E)$ en temps $O^*(3^n)$ et espace $O^*(2^n)$.*

Démonstration. L'algorithme énumère les sous-ensembles de V comme b-base, et vérifie si elles peuvent être étendues, en utilisant le Lemme 2.18. Le temps d'exécution $T(n)$ de l'algorithme vérifie, pour un facteur polynomial $p : \mathbb{N} \to \mathbb{R}$:

$$T(n) = \sum_{\mathcal{B} \subseteq V} p(n - |\mathcal{B}|) \cdot 2^{n-|\mathcal{B}|} = \sum_{i=1}^{n} \binom{n}{i} p(n-i) \cdot 2^{n-i} = O^*(3^n)$$

□

Notons qu'en plus de déterminer le nombre b-chromatique d'un graphe, cet algorithme nous permet de construire en temps $O^*(3^n)$ le b-spectre du graphe, et de déterminer s'il est b-continu ou non.
Nous listons à présent quelques classes de graphes avec une meilleure borne du temps d'exécution de l'algorithme.

Corollaire 2.21. *Le nombre b-chromatique d'un graphe avec degré maximum borné peut être calculé en temps et espace $O^*(2^n)$.*

Démonstration. Soit $G = (V, E)$ un graphe avec degré maximum borné $\Delta \in \mathbb{N}$. Alors un b-sommet ne peut être voisin à plus de Δ autres classes de couleurs, ce qui signifie que toute b-coloration de G a une taille au plus $\Delta + 1$. Un algorithme qui énumère tous les sous-ensembles de G de taille au plus $\Delta + 1$, et essaie d'étendre ces b-bases, résoud le problème en temps $O^*(\Delta n^{\Delta+1} 2^n) = O^*(2^n)$. □

Corollaire 2.22. *Si G est un graphe avec degré moyen borné d, $\chi_b(G)$ peut être calculé en espace $O^*(2^n)$, et à un facteur sous-exponentiel près, en temps $O^*(2^n)$.*

Démonstration. Soit $G = (V, E)$ un graphe avec degré moyen borné $d \in \mathbb{R}_+$, et soit $\mathcal{B}$ une b-base d'une b-coloration optimale, avec $|\mathcal{B}| = k$. Soit $E(\mathcal{B})$ l'ensemble des arêtes incidentes à un sommet de $\mathcal{B}$, c'est-à-dire $E(\mathcal{B}) = \{\{x, y\} \in E \ / \ x \in \mathcal{B}\}$. Puisque pour chaque $b \in \mathcal{B}$, $deg(b) \geq k - 1$, nous avons $|E(\mathcal{B})| \geq k(k-1)/2$. En conséquence, $dn/2 = |E| \geq |E(\mathcal{B})| \geq k(k-1)/2$.
Cela implique que k satisfait l'inégalité $k^2 - k - dn \leq 0$, et donc $k \leq (1 + \sqrt{1 + 4dn})/2 \leq \sqrt{dn} + 1$.
Un algorithme qui énumère tous les sous-ensembles de V de taille au plus k et essaie de les étendre s'exécute, à un facteur $O^*(2^{\sqrt{d}\sqrt{n}\log n}) = O^*(2^{o(n)})$ sous-exponentiel près, en temps $O^*(2^n)$. □

Corollaire 2.23. *Si G est un graphe avec degré moyen dn, et avec $d \in \mathbb{R}_+$, $d \leq 1/9$, alors il y a un algorithme qui calcule $\chi_b(G)$ en temps $O^*((\frac{(1-\alpha)^{\alpha-1}}{\alpha^\alpha} 2^{1-\alpha})^n)$ et espace $O^*(2^n)$, où $\alpha = \sqrt{d}$.*

Démonstration. Soit $G = (V, E)$ un graphe de degré moyen dn. Pour les mêmes raisons que précédemment, k satisfait l'inégalité $k^2 - k - dn^2 \leq 0$, et donc $k \leq (1 + \sqrt{4dn^2 + 1})/2 \leq \sqrt{d}n + 1$.
Il est facile de voir que $\binom{n}{t}2^{n-t} \leq \binom{n}{t+1}2^{n-(t+1)}$ si et seulement si $t \leq (n-2)/3$. Alors, pour $\alpha = \sqrt{d} \leq 1/3$, en énumérant tous les sous-ensembles de V de taille au plus k, l'algorithme a un temps d'exécution de $O^*((\frac{(1-\alpha)^{\alpha-1}}{\alpha^\alpha}2^{1-\alpha})^n)$ - qui est mieux que $O^*(3^n)$ lorsque $d < 1/9$. □

2.6 Nombre Clique-Chromatique

2.6.1 Introduction

Nombre Clique-Chromatique. Soit $G = (V, E)$ un graphe, et $\varphi = \{V_1, \ldots, V_k\}$ une k-coloration de V. Une clique maximale $C \subseteq V$ de G est dite *monochromatique* si C est incluse dans une classe de couleur de φ. Une k-coloration φ de V, avec $k \in \mathbb{N}_{\geq 2}$, est alors appelée une *clique-coloration* de taille k de G, ou encore une k-clique-coloration de G, si aucune clique maximale de G n'est monochromatique. En d'autres termes, pour une k-clique-coloration, toute clique maximale de G contient au moins deux sommets de couleurs différentes. Le *Nombre Clique-Chromatique* de G, noté $\chi_c(G)$, est le plus petit entier k pour lequel G admet une k-clique-coloration.

NOMBRE CLIQUE-CHROMATIQUE
VERSION : *OPTIMISATION*

Entrée : $G = (V, E)$
Sortie : Déterminer le plus petit entier $k = \chi_c(G) \in \mathbb{N}_{\geq 2}$ tel que G admette une k-coloration où aucune classe de couleur ne contienne de clique maximale de G

Ce paramètre est défini pour tout graphe sans sommet isolé, puisqu'un tel graphe admet une n-clique-coloration. Pour cette Section, on notera $\mathcal{H}_c(G) \subseteq 2^V$ l'hypergraphe des cliques maximales d'un graphe $G = (V, E)$. ♣

Le problème de CLIQUE-COLORATION a été introduit par Duffus, Sands, Sauer et Woodrow en 1991, et peut être vu comme un cas spécial de problème de coloration d'hypergraphes puisque le problème est équivalent à colorier l'hypergraphe des cliques maximales de G, c'est-à-dire l'hypergraphe dans lequel les hyperarêtes sont les cliques maximales de G [66].
La complexité du problème de clique-coloration est bien étudiée. Marx a prouvé en 2011 que pour tout entier $k \geq 2$ fixé, décider si un graphe admet une k-clique-coloration est un problème Σ_2^p-complet. En conséquence, le problème du NOMBRE CLIQUE-CHROMATIQUE est Σ_2^p-difficile. Si l'entrée est l'hypergraphe des cliques maximales, c'est-à-dire si le graphe est donné par sa liste de cliques maximales, alors le problème de k-CLIQUE-COLORATION devient NP-complet [150]. En 2004, Bacsó et al. ont montré qu'il est coNP-complet de décider si une coloration de sommet donnée est en fait une 2-clique-coloration [12].
Le problème de clique-coloration a été étudié sur différentes classes de graphes. En 2002, Kratochvíl et Tuza ont montré qu'il est NP-complet de décider si un graphe parfait possède une 2-clique-coloration [130]. Le nombre clique-chromatique des graphes parfaits et de ses sous-classes a également été étudié dans la THÉORIE DESGRAPHES. La motivation principale de ces recherches est la conjecture longuement ouverte que le nombre clique-chromatique de tout graphe parfait est au plus trois. Actuellement, on ne sait même pas s'il existe une borne constante au nombre clique-chromatique des graphes parfaits.

Defossez fournit une collection de classes de graphes 2-clique-colorables [55], parmi lesquelles les graphes *fortement parfaits*. En conséquence, tous les graphes des classes suivantes sont 2-clique-colorables : bipartis, *de comparabilité*, cordaux, *cobipartis*, *de cocomparabilité* [18].
Kratochvíl et Tuza ont montré que le nombre clique-chromatique peut être calculé en temps polynomial sur les graphes planaires[130]. Ils ont montré qu'on peut décider en temps polynomial si un graphe planaire admet une 2-clique-coloration, et combiné ce résultat avec celui de Mohar et Škrekovski stipulant que tout graphe planaire admet une 3-clique-coloration [158]. Récemment, Klein et Morgana ont montré qu'il existe un algorithme polynomial pour calculer le nombre clique-chromatique des graphes contenant peu de P_4 [124].

2.6.2 Obliques et Transversaux

Dans cette Sous-Section, nous introduisons la notion d'obliques pour les hypergraphes, qui définit tous les sous-ensembles de sommets qui ne sont pas des transversaux. Nous nous intéressons aux aspects algorithmiques pour décider si un sous-ensemble de sommets est un transversal ou un oblique, ainsi que pour décider respectivement de leur minimalité ou de leur maximilité au sens de la Section 1.1. On s'intéressera également à l'énumération de toutes ces familles de sous-ensembles de sommets, que ce soit dans un hypergraphe quelconque ou dans un hypergraphe de cliques maximales. En particulier, nous montrerons que dans l'hypergraphe des cliques maximales $\mathcal{H}_c(G)$ d'un graphe $G = (V, E)$, toutes ces familles ainsi que leurs complémentaires peuvent être énumérés en temps et espace $O^*(2^n)$. Ce résultat nous permettra d'appliquer le Théorème 1.4 du principe d'*Inclusion-Exclusion* pour construire dans la Sous-Section 2.6.3 un algorithme exact en temps et espace $O^*(2^n)$ pour calculer le nombre clique-chromatique d'un graphe. Dans cette Section, nous poserons $n = \mathcal{U}$ lorsque nos résultats sont établis pour un univers $\mathcal{U}$ d'objets considérés. Nous établissons tout d'abord le Lemme 2.24 qui nous sera utile pour la suite.

Lemme 2.24. *Soit $G = (V, E)$ un graphe et $X \subseteq V$. Il y a un algorithme qui décide en temps polynomial $O(n|X|)$ si X est une clique ou une clique maximale de G.*

Démonstration. Une clique X est strictement contenue dans une autre Y si et seulement pour tout $W \subseteq Y - X : X \cup W$ est une clique. En particulier, X est une clique maximale si et seulement si pour tout $v \in V - X : X \cup \{v\}$ n'est pas une clique. L'algorithme décide donc dans un premier temps si X est une clique de G en temps $O(|X|^2)$. Si X n'est pas une clique, avec $|X| \leq n$, l'algorithme s'est exécuté en temps $O(n|X|)$. Sinon si X est une clique, l'algorithme poursuit pour décider si X est maximale, c'est-à-dire avec l'observation initiale X n'est pas maximale s'il existe $v \in V - X$ pour lequel $X \cup \{v\}$ est une clique. Puisque X est une clique, cette opération s'effectue pour tout $v \in V - X$ en temps $O(|X|)$. Le temps total de l'algorithme est alors $O(|X|^2 + (n - |X|) \cdot |X|) = O(n|X|)$. □

Obliques. Un sous-ensemble des sommets $O \subseteq V$ est un *oblique* d'un hypergraphe $\mathcal{H}$ si ce n'est pas un transversal de $\mathcal{H}$. Ainsi $O \subseteq V$ est un oblique de $\mathcal{H}$ s'il existe $X \in \mathcal{H}$ tel que $O \cap X = \emptyset$. Les obliques maximaux et maximums sont définis au sens de la Section 1.1. ♣

Propriété 2.25. *Soit $\mathcal{H}$ un hypergraphe, les propriétés suivantes sont immédiates :*
◇ *$O \subseteq V$ est un oblique de $\mathcal{H}$ si et seulement si $X \subseteq \overline{O}$, pour une hyperarête X de $\mathcal{H}$.*
◇ *$O \subseteq V$ est un oblique de $\mathcal{H}$ si et seulement si $O \subseteq \overline{X}$, pour une hyperarête X de $\mathcal{H}$.*
◇ *$O \subseteq V$ est un oblique de $\mathcal{H}$ si et seulement si pour tout $X \subseteq O : X$ est un oblique.*
◇ *$T \subseteq V$ est un transversal de $\mathcal{H}$ si et seulement si pour tout X tel que $T \subseteq X : X$ est un transversal.*
◇ *Le complémentaire d'un transversal n'est pas toujours un oblique, et réciproquement.*
◇ *Les transversaux ou les obliques peuvent être des stables de $\mathcal{H}$.*

Dans la suite de l'article, nous dénoterons par $\mathcal{T}_c$ et $\mathcal{O}_c$, respectivement, la famille de tous les transversaux et la famille des obliques de l'hypergraphe des cliques maximales $\mathcal{H}_c(G)$. Nous pourrons également utiliser la notation $\mathcal{T}$ et $\mathcal{O}$ lorsque l'on parlera des hypergraphes $\mathcal{H}$ en général. La définition implique directement différentes propriétés. En particulier tout sous-ensemble de sommet $X \subseteq V$ d'un hypergraphe et soit un oblique soit un transversal. Ainsi $\{\mathcal{T}, \mathcal{O}\}$ constitue une partition de la famille des sous-ensembles de sommets 2^V d'un hypergraphe.

Lemme 2.26. *Soit $G = (V, E)$ un graphe, et soit $X \subseteq V$. Il y a un algorithme qui décide en temps $O^*(3^{(n-|X|)/3})$ et espace polynomial $O(n^{O(1)})$ si $X \in \mathcal{T}_c$ ou bien si $X \in \mathcal{O}_c$.*

Démonstration. Par les Propriétés 2.25, l'algorithme décide si X est un oblique de $\mathcal{H}_c(G)$ en décidant s'il existe $C \in \mathcal{H}_c(G)$ tel que $C \subseteq \overline{O}$. Pour cela, l'algorithme construit l'hypergraphe des cliques maximales de $\mathcal{H}_c(G[\overline{X}])$ du sous-graphe induit de G par le complémentaire de X. Cette construction s'effectue en temps $O^*(3^{(n-|X|)/3})$ et espace polynomial $O(n^{O(1)})$. Pour chaque clique maximale $C \in \mathcal{H}_c(G[\overline{X}])$ rencontrée, en utilisant le Lemme 2.24, l'algorithme décide en temps polynomial si C est strictement contenue dans une plus grande clique de G, autrement dit s'il existe $C^+ \in \mathcal{H}_c(G)$ telle que $C \subset C^+$ et $C^+ \cap X \neq \emptyset$. Si toutes les cliques maximales C de $G[\overline{X}]$ satisfont cette propriété, alors X est un transversal de $\mathcal{H}_c(G)$, sinon X est un oblique. L'algorithme s'exécute finalement en temps $O^*(3^{(n-|X|)/3})$ et espace polynomial. □

Lemme 2.27. *Soit $G = (V, E)$ un graphe, et $X \subseteq V$. Il y a un algorithme en temps $O^*(3^{(n-|X|)/3})$ et espace polynomial pour décider si X est un transversal minimal de $\mathcal{H}_c(G)$.*

Démonstration. En utilisant les Propriétés 2.25, on observe que X est un transversal minimal d'un hypergraphe $\mathcal{H}$ si et seulement si $\forall\ x \in X : X - \{x\}$ n'est pas un transversal de $\mathcal{H}$. Pour notre cas $\mathcal{H} = \mathcal{H}_c(G)$, l'algorithme décide dans un premier temps si X est un transversal de $\mathcal{H}_c(G)$ en utilisant l'algorithme du Lemme 2.26. Puis si X est un transversal, l'algorithme décide donc dans un deuxième temps si X est minimal en faisant au plus $|X|$ appels à l'algorithme du Lemme 2.26. □

Par leur définition, décider si un sous-ensemble de sommets $X \subseteq V$ est un transversal ou un oblique nécessite nécessairement le même temps d'exécution. Ceci est de plus valable dans tout hypergraphe. Cependant, les Lemmes 2.27 et 2.29 montrent qu'il n'en est pas de même pour les transversaux minimaux et les obliques maximaux dans l'hypergraphe des cliques maximales. Le Théorème 2.28 est le coeur de cette propriété.

Théorème 2.28. *Les obliques maximaux de l'hypergraphe des cliques maximales $\mathcal{H}_c(G)$ sont exactement les compléments des cliques maximales de G.*

Démonstration. "⇐" Soit C une clique maximale de G et donc $C \in \mathcal{H}_c(G)$. Soit $X \subseteq V$ le complément de la clique maximale C. Alors $X \cap C = \emptyset$, et par les Propriétés 2.25, X est un oblique de $\mathcal{H}_c(G)$. Nous affirmons que X traverse toutes les cliques maximales de $\mathcal{H}_c(G)$ sauf C. Par contradiction, supposons qu'il y ait $C' \in \mathcal{H}_c(G)$, $C' \neq C$, telle que $X \cap C' = \emptyset$. Alors $C' \subseteq \overline{X}$. Puisque $\overline{X} = C$, et $C' \neq C$, nous avons $C' \subseteq C$, et donc C' n'est pas une clique maximale de G, une contradiction. On en déduit $\forall\ Y \subseteq V$, $X \subset Y : Y \in \mathcal{T}_c$, ce qui montre par définition que X est maximal.
"⇒" Soit $O \subseteq V$ un oblique maximal de $\mathcal{H}_c(G)$. Par définition, il existe $C \in \mathcal{H}_c(G)$ tel que $O \subseteq \overline{C}$. Nous affirmons que $O = \overline{C}$ et le montrons par contradiction. Ainsi, supposons que O est contenu strictement dans $\overline{C}$, donc il existe $Y \subseteq \overline{C}$ tel que $O \cup Y = \overline{C}$ et $Y \neq \emptyset$. Par construction et par définition d'un oblique, $O \cup Y \subseteq \overline{C}$ implique que $O \cup Y$ est un oblique. $O \subset O \cup Y$ car $Y \neq \emptyset$, et $O \cup Y$ un oblique de $\mathcal{H}_c(G)$, montrent que O n'est pas maximal, une contradiction. □

Lemme 2.29. *Soit $G = (V, E)$ un graphe, et $X \subseteq V$. Il y a un algorithme en temps polynomial $O(n^2 - n|X|)$ pour décider si X est un oblique maximal de $\mathcal{H}_c(G)$.*

Démonstration. En utilisant le Théorème 2.28, il suffit de décider si $\overline{X}$ est une clique maximale, ce qui peut être fait en utilisant l'algorithme du Lemme 2.24 ici en temps $O(n(n - |X|))$. □

Les Lemmes 2.27 et 2.29 montrent une différence algorithmique notable entre les reconnaissances des transversaux minimaux et des obliques maximaux dans l'hypergraphe des cliques maximales. Nous allons voir qu'il y a également une différence importante en ce qui concerne l'énumération de ces objets. Jusqu'à présent, le Lemme 2.27 permet d'énumérer les transversaux minimaux de $\mathcal{H}_c(G)$ en temps $O^*(2.4423^n)$. Le Lemme 2.29 permet quant à lui d'énumérer les obliques maximaux de $\mathcal{H}_c(G)$ en temps $O^*(2^n)$. Cependant, une application judicieuse du Théorème 2.28 permet d'établir directement le Lemme 2.30 suivant.

Lemme 2.30. *La famille des obliques maximaux de $\mathcal{H}_c(G)$ peut être énumérée en temps $O^*(3^{n/3})$.*

Le Théorème 2.31 présente l'intérêt algorithmique des clôtures de familles définies en Section 1.1. Le Théorème 2.32 agrémente ces résultats.

Théorème 2.31. *Soit $\mathcal{U}$ un univers d'objets, et $\mathcal{F} \subseteq 2^{\mathcal{U}}$ une famille de sous-ensembles de $\mathcal{U}$. Les Algorithmes 2.5 et 2.6 énumèrent respectivement la clôture descendante $\mathcal{F}_\downarrow$ et la clôture ascendante $\mathcal{F}_\uparrow$ de $\mathcal{F}$ en temps et espace $O^*(|\mathcal{F}_\downarrow|)$ et $O^*(|\mathcal{F}_\uparrow|)$.*

Algorithme 2.5: Algorithme du Théorème 2.31 pour déterminer la clôture descendante $\mathcal{F}_\downarrow$ d'une famille $\mathcal{F}$ de sous-ensembles d'un univers $\mathcal{U}$ quelconque

```
   input  : U, F ⊆ 2^U
   output: F↓
   // Initialisation de F↓
1  F↓ ← F;
   // Parcours par ordre décroissant de taille des sous-ensembles X ∈ F↓
2  for i = n downto 1 do
       // Liste temporaire L qui trace tous les sous-ensembles de F↑ de taille i
3      L ← ∅;
4      foreach X ∈ F↑, |X| = i do
5          L ← L ∪ {X};
6      foreach X ∈ L do
7          foreach x ∈ X do
8              F↓ ← F↓ ∪ {X − {x}};
9  return F↓;
```

Démonstration. La validité de ces algorithmes peut se montrer par récurrence sur la taille des sous-ensembles de sommets (dans l'ordre correspondant).
Pour analyser le temps d'exécution, observons que toute famille $\mathcal{G} \subseteq 2^{\mathcal{U}}$ de sommets peut être encodée en sous-ensemble de l'ensemble d'entier $\{0, \ldots, 2^{|\mathcal{U}|} - 1\}$. Pour stocker les sous-ensembles de $\mathcal{U}$ qui appartiennent aux clôtures respectives, nous utilisons une structure de données D qui supporte les opérations d'insertion et de recherche en temps $O(\log p)$, où p est le nombre de clés stockées. Cette structure peut par exemple prendre la forme d'un arbre rouge et noir [141]. Observons

Algorithme 2.6: Algorithme du Théorème 2.31 pour déterminer la clôture ascendante $\mathcal{F}_\uparrow$ d'une famille $\mathcal{F}$ de sous-ensembles d'un univers $\mathcal{U}$ quelconque

```
   input  : U, F ⊆ 2^U
   output: F↑
1  F↑ ← F;
   // Parcours par ordre croissant de taille des sous-ensembles X ∈ F↑
2  for i = 1 to n − 1 do
       // Liste temporaire L (voir Alg. 2.5)
3      L ← {X ∈ F↑ : |X| = i};
4      foreach X ∈ L do
5          foreach x ∈ V − X do
6              F↑ ← F↑ ∪ {X + {x}};
7  return F↑;
```

que cela implique que D peut être initialisée en temps $O(p \cdot \log p)$. La liste temporaire $\mathcal{L}$ utilisée dans les deux algorithmes peut prendre la forme d'une liste simplement chaînée, qui permet l'insertion d'un élément en temps $O(1)$ et le parcours des éléments en temps $O(|\mathcal{L}|)$. Pour raisons de similarité, nous ne décrivons le support des structures de données que pour l'Algorithme 2.5 qui construit la clôture descendante $\mathcal{F}_\downarrow$ de $\mathcal{F}$.
Observons dans un premier temps que l'ensemble des opérations qui impliquent $\mathcal{L}$ s'effectuent en temps total $O(|\mathcal{F}_\downarrow|)$. Dans un deuxième temps, observons que pour un ensemble $X \in \mathcal{F}_\downarrow$, les opérations qui impliquent D représentent à la ligne 8 au plus n recherches et une insertion. D est ainsi impliquée dans au plus $n \cdot |\mathcal{F}_\downarrow|$ recherches et $|\mathcal{F}_\downarrow|$ insertions dans sa structure. Au total, l'Algorithme 2.5 s'exécute donc en temps $O(n \cdot \log n \cdot |\mathcal{F}_\downarrow| \cdot \log |\mathcal{F}_\downarrow|)$, et puisque $|\mathcal{F}_\downarrow| \leq 2^n$ nous avons $\log |\mathcal{F}_\downarrow| \leq n$. Le temps d'exécution $T(n)$ de l'algorithme satisfait donc $T(n) = O^*(|\mathcal{F}_\downarrow|)$. □

Théorème 2.32. *Soit $\mathcal{U}$ un univers d'objets, et soient $\mathcal{F}, \mathcal{G} \subseteq 2^\mathcal{U}$ deux familles de sous-ensembles de l'univers. S'il y a un algorithme en temps $O^*(t(n))$ pour énumérer $\mathcal{F}$, alors il y a un algorithme en temps $O^*(2^n + t(n))$ et espace $O^*(2^n)$ pour énumérer $\mathcal{G}$ si :*
◇ $\mathcal{G} = \mathcal{F}$
◇ $\mathcal{G} = \mathcal{F}_\downarrow$ *ou* $\mathcal{G} = \mathcal{F}_\uparrow$
◇ $\mathcal{G}$ *est constituée des complémentaires des éléments de* $\mathcal{F}$*, c'est-à-dire* $\mathcal{G} = \{X \subseteq \mathcal{U} \ : \ \overline{X} \in \mathcal{F}\}$
◇ $\mathcal{G}$ *est la famille complémentaire de* $\mathcal{F}$*, c'est-à-dire* $\mathcal{G} = 2^\mathcal{U} - \mathcal{F}$
◇ $\mathcal{F} = \mathcal{G}_\downarrow$ *avec pour tous* $X, Y \in \mathcal{G}$ *distincts :* X *n'est pas inclus dans* Y
◇ $\mathcal{F} = \mathcal{G}_\uparrow$ *avec pour tous* $X, Y \in \mathcal{G}$ *distincts :* X *n'est pas inclus dans* Y

Démonstration. Soient $\mathcal{U}, \mathcal{F}$ et $\mathcal{G}$ définis comme dans le théorème. Nous mettons à disposition une table T de 2^n valeurs dans $\{0, 1\}$, une valeur $T[X]$ pour chaque sous-ensemble $X \subseteq \mathcal{U}$.
◇ Si $\mathcal{G} = \mathcal{F}$ le résultat est immédiat.
◇ Si $\mathcal{G} = \mathcal{F}_\downarrow$ ou $\mathcal{G} = \mathcal{F}_\uparrow$, le résultat découle immédiatement du Théorème 2.31.
◇ Si $\mathcal{G} = \{X \subseteq \mathcal{U} \ : \ \overline{X} \in \mathcal{F}\}$, un parcours direct de $\mathcal{F}$ permet de construire $\mathcal{G}$.
◇ Si $\mathcal{G} = 2^\mathcal{U} - \mathcal{F}$, on initialise $\forall\ \mathcal{U} \subseteq V : T[\mathcal{U}] = 1$. Puis on énumère $\mathcal{F}$ en opérant $\forall\ X \in \mathcal{F} : T[X] = 0$. Finalement, il suffit pour construire $\mathcal{G}$ de parcourir toutes les valeurs de T avec la relation $\forall\ X \subseteq \mathcal{U} : T[X] = 1$ si et seulement si $X \in \mathcal{G}$.
◇ Si $\mathcal{F} = \mathcal{G}_\downarrow$ avec pour tous $X, Y \in \mathcal{G}$ distincts : X n'est pas inclus dans Y, on initialise $T[X] = 1$ si et seulement si $X \in \mathcal{F}$. Enfin, $X \in \mathcal{G}$ si et seulement si $\forall\ x \in V - X : T[X \cup \{x\}] = 0$.

◇ Si $\mathcal{F} = \mathcal{G}_{\uparrow}$ avec pour tous $X, Y \in \mathcal{G}$ distincts : X n'est pas inclus dans Y, on initialise $T[X] = 1$ si et seulement si $X \in \mathcal{F}$. Enfin, $X \in \mathcal{G}$ si et seulement si $\forall\, x \in X : T[X - \{x\}] = 0$.

□

Les Propriétés 2.33 qui découlent des Propriétés 2.25 mettent en avant les relations de clôture des familles d'objets que nous étudions dans cette Section. Les définitions des obliques et des transversaux, avec leurs Propriétés 2.33 et 2.25, conjugués aux Théorèmes 2.31 et 2.32, ainsi qu'au Lemme 2.30, permettent d'établir le Théorème 2.34 pour l'énumération de tous les objets de cette Section dans l'hypergraphe des cliques maximales.

Propriété 2.33. *Soit $\mathcal{H}$ un hypergraphe, les propriétés suivantes sont satisfaites :*
◇ *La famille des obliques est la clôture descendante des obliques maximaux.*
◇ *La famille des transversaux est la clôture ascendante des transversaux minimaux.*

Théorème 2.34. *Soit $G = (V, E)$ un graphe. Pour l'hypergraphe $\mathcal{H}_c(G)$, il y a un algorithme en temps et espace $O^*(2^n)$ pour énumérer : les obliques, les obliques maximaux, les transversaux, les transversaux minimaux, le complémentaire des objets de chacune de ces familles, ou encore les complémentaires de chacune de ces familles dans 2^V.*

Démonstration. Le Lemme 2.30 permet d'obtenir en temps $t(n) = O^*(3^{n/3}) = O^*(2^n)$ les obliques maximaux de $\mathcal{H}_c(G)$. Avec la Propriété 2.33, les obliques sont la clôture descendante des obliques maximaux. Par le Théorème 2.31, les obliques maximaux peuvent donc être énumérés en temps $O^*(2^n)$. Par définition, les transversaux sont le complémentaire de la famille des obliques dans 2^V. Nous pouvons donc énumérer les transversaux de $\mathcal{H}_c(G)$ en temps $O^*(2^n)$ par le Théorème 2.32. Les transversaux étant la clôture ascendante des transversaux minimaux, et par définition deux transversaux minimaux ne pouvant être inclus l'un dans l'autre, la famille des transversaux minimaux peut donc être obtenue en temps $O^*(2^n)$ à partir des transversaux via le Théorème 2.32. Finalement, puisque chacune de ces familles peut être obtenue en temps $O^*(2^n)$, le complémentaire des objets de chacune de ces familles ainsi que les complémentaires de ces familles peuvent être déterminées dans le même temps et espace $O^*(2^n)$ par le Théorème 2.32. □

2.6.3 Algorithme Exact

Nous présentons un algorithme d'*Inclusion-Exclusion* pour résoudre le problème du Nombre Clique-Chromatique d'un graphe. Pour appliquer le Théorème 1.4, le Lemme 2.35 caractérise la famille $\mathcal{F}$ à laquelle toutes les classes de couleurs d'une k-clique-coloration appartiennent.

Lemme 2.35. *$\{V_1, \ldots, V_k\}$ est une k-clique-coloration de G si et seulement si $\{V_1, \ldots, V_k\}$ est une k-partition de G et, pour chaque $1 \leq i \leq k$, $\overline{V_i} = V \backslash V_i$ est un transversal de $\mathcal{H}_c(G)$.*

Démonstration. Soit $\{V_1, \ldots, V_k\}$ une k-clique-coloration, et supposons qu'il existe un V_i tel que son complément $\overline{V_i}$ ne soit pas un transversal $\mathcal{H}_c(G)$. Alors il existe une clique maximale $C \in \mathcal{H}_c(G)$ telle que pour tout j avec $1 \leq j \neq i \leq k$ qui satisfait $V_j \cap C = \emptyset$. En conséquence, $C \cap \bigcup_{j=1\ldots k}^{j \neq i} V_j = \emptyset$, qui implique que C est un sous-ensemble du complément de l'union de tous les V_j satisfaisant $1 \leq j \neq i \leq k$, qui est exactement V_i. Ainsi $C \subseteq V_i$ et donc C est une clique maximale monochromatique w.r.t. $\{V_1, \ldots, V_k\}$. Cela contredit l'hypothèse que $\{V_1, \ldots, V_k\}$ est une k-clique-coloration.
Inversement, soit $\{V_1, \ldots, V_k\}$ une k-partition de G telle que le complément de chaque classe de couleur V_i soit un transversal de l'hypergraphe des cliques maximales $\mathcal{H}_c(G)$. Supposons qu'il existe une clique maximale monochromatique $C \in \mathcal{H}_c(G)$, c'est-à-dire, il existe i tel que $1 \leq i \leq k$ et $C \subseteq V_i$. Alors $\overline{V_i}$ n'est pas un transversal de $\mathcal{H}_c(G)$ puisque $C \subseteq V_i$ implique $C \cap \overline{V_i} = \emptyset$, une contradiction. □

Les résultats obtenus à la Sous-Section 2.6.2 permettent immédiatement d'établir le Théorème 2.36 suivant.

Théorème 2.36. *Il y a un algorithme pour calculer le nombre clique-chromatique $\chi_c(G)$ d'un graphe $G = (V, E)$ en temps et espace $O^*(2^n)$.*

Démonstration. L'algorithme commence par construire la famille $\mathcal{F}$ des classes de couleurs possibles pour toute clique-partition de V. D'après le Lemme 2.35, la famille $\mathcal{F}$ des classes de couleurs est exactement la famille des complémentaires des transversaux de l'hypergraphe des cliques maximales $\mathcal{H}_c(G)$. Nous pouvons alors construire $\mathcal{F}$ de manière explicite en temps et espace $O^*(2^n)$ par l'intermédiaire du Théorème 2.34. Finalement, nous pouvons déterminer le nombre clique-chromatique de G, c'est-à-dire le plus petit entier $k \in \mathbb{N}_{\geq 2}$ pour lequel G admet une k-clique-coloration, en effectuant au plus k appels à l'algorithme du Théorème 1.4. Notre algorithme pour résoudre le problème du Nombre Clique-Chromatique s'exécute donc en temps et espace $O^*(2^n)$. □

2.6.4 2-Clique-Colorabilité

En 2006 Défossez a montré qu'il est NP-difficile de décider si un K_4-free perfect graph admet une 2-clique-coloration [55]. Dans cette Sous-Section, nous présentons un algorithme exact pour calculer une 2-clique-coloration, s'il en existe une, en supposant que les cliques maximales du graphe donné G ont une cardinalité bornée, i.e., $\omega(G) \leq c$ pour une certaine constante c positive. Nous montrons comment ce problème sur les hypergraphes de cliques peut être transformé en un problème d'énumération de transversaux minimaux d'un hypergraphe de rang borné c. Cela mène à un meilleur temps d'exécution meilleur que ceux de $O^*(2^n)$ établis par l'algorithme d'inclusion-exclusion présenté dans la Sous-Section 2.6.3. Plus précisement, pour chaque $c \geq 2$, il y a une constante λ_c telle que tous les transversaux minimaux d'un hypergraphe dont les hyperarêtes ont une cardinalité bornée par c peuvent être énumérés en temps $O^*((\lambda_c)^n)$, où $\lambda_c < 2$. Tandis que l'algorithme de la Sous-Section 2.6.3 utilise le principe d'*Inclusion-Exclusion*, la technique principalement utilisée ici est la technique de *Branchement*. Le Lemme 2.37 appuie notre résultat.

Lemme 2.37. *Tout graphe $G = (V, E)$ admet une clique-coloration optimale telle qu'au moins une classe de couleur est le complément d'un tranvsersal minimal de l'hypergraphe des cliques maximales $\mathcal{H}_c(G)$.*

Démonstration. Soit $G = (V, E)$ un graphe et $k = \chi_c(G)$ son nombre clique-chromatique. Soit $\{V_1, \ldots, V_k\}$ une clique-coloration optimale de G. Par le Lemme 2.35, le complément de chaque classe de couleur V_i est un transversal de $\mathcal{H}_c(G)$. Supposons qu'aucune classe de couleur V_i n'est le complément d'un transversal minimal. L'idée est de transformer la première classe de couleur en un complément de transversal minimal. Nous avons supposé que $\overline{V_1}$ est un transversal qui n'est pas minimal, alors il existe $X_1 \subset \overline{V_1}$ tel que X_1 est un transversal minimal de $\mathcal{H}_c(G)$. Nous imposons $V_1' = \overline{X_1}$, et pour chaque $i \geq 2$, nous fixons $V_i' = V_i \backslash V_1'$. Au cours de cette opération, pour chaque $i \geq 2$, nous n'avons pas augmenté la taille des V_i, et donc nous n'avons pas diminué la taille de leur complément qui étaient déjà des transversaux. Donc pour chaque $i \geq 2$, le complément de chaque V_i' est un transversal. Par construction de V_1', et par le Lemme 2.35, nous obtenons que $\{V_1', V_2', \ldots, V_k'\}$ est une clique-coloration de G. □

Théorème 2.38. *Il y a un algorithme de Branchement qui s'exécute en temps $O^*((\lambda_c)^n)$ et espace polynomial pour décider si un graphe de taille de clique maximum $\omega(G) \leq c$ bornée est 2-clique-colorable, où $\lambda_c < 2$.*

2.7 Autres Problèmes

2.7.1 Nombre Pseudo-a-Chromatique

Nombre Pseudo-a-Chromatique. La relaxation du problème du NOMBRE a-CHROMATIQUE, dans lequel la propriété d'ensembles stables des classes de couleurs n'est pas requise, est appelé le problème du NOMBRE PSEUDO-a-CHROMATIQUE, également connu sous le nom problème de PARTITION COMPLÈTE [128]. Nous appelerons *k-pseudo-a-coloration* une telle k-partition des sommets d'un graphe.

> k-PSEUDO-a-COLORATION
> *VERSION* : *DÉCISION*
>
> **Entrée** : $G = (V, E)$, $k \in \mathbb{N}^*$
> **Sortie** : Existe-t-il une k-coloration de V telle que le graphe complet à k sommets peut être obtenu par contraction des classes de couleurs ?

Le *Nombre Pseudo-a-Chromatique* d'un graphe G, noté $\chi_{pa}(G)$, est le plus grand entier $k \in \mathbb{N}$ tel que G admette une k-pseudo-a-coloration. ♣

Cette notion a été introduite en 1969 [105]. Ce problème est NP-difficile [194], et pour tout graphe G nous avons naturellement la relation : $\chi_a(G) \leq \chi_{pa}(G)$. Des bornes de ce nombre sur certaines classes de graphes ont été étudiées [14, 193]. Même si cela semble être admis, il n'est pas clair actuellement si ce problème demeure NP-difficile sur les arbres [128, 68]. A l'instar du nombre a-chromatique, le problème du NOMBRE PSEUDO-a-CHROMATIQUE est un problème FPT. En effet, pour tout graphe G décider si $\chi_{pa}(G) \geq k$ peut être résolu en temps $O^*(k^{(k-2)(k+1)})$ [36].

Pour l'ALGORITHMIQUE EXACTE EXPONENTIELLE, à l'instar du problème du NOMBRE a-CHROMATIQUE nous ne connaissons pas d'autres algorithmes que l'algorithme trivial en temps $O^*(n^n)$ pour le problème du NOMBRE PSEUDO-a-CHROMATIQUE. Dans ce chapitre, avons proposé en Section 2.4 un algorithme en temps $O^*(k^{\mathbf{pw}(G)} \cdot 2^{k^2/2})$ pour décider si un graphe admet une k-a-coloration. Le Lemme 2.39 et le Corollaire 2.40 proposent d'adapter cet algorithme pour le problème du NOMBRE PSEUDO-a-CHROMATIQUE afin d'obtenir des résultats similaires à ceux obtenus pour le problème du NOMBRE a-CHROMATIQUE.

Lemme 2.39. *L'Algorithme 2.4 du Théorème 2.13 peut être adapté pour décider en temps et espace $O^*(k^{\mathbf{pw}(G)} \cdot 2^{k^2/2})$ si un graphe $G = (V, E)$ admet une k-pseudo-a-coloration.*

Démonstration. Pour le problème de k-a-COLORATION, la ligne 6 de l'Algorithme 2.4 garantie que les classes de couleur soient des stables. La condition de la ligne 6 agit comme un filtre sur les couleurs $k(v)$ possibles du sommet v considéré dans le nœud de type INTRODUCE. Pour adapter l'algorithme au problème de k-PSEUDO-a-COLORATION, il suffit alors de modifier la condition de la ligne 6 et accepter pour v toutes les couleurs parmi $\{1, \ldots, k\}$. □

Corollaire 2.40. *Si sur les classes de graphes considérées, le problème du* Nombre Pseudo-a-Chromatique *demeure* NP*-difficile, alors il y a un algorithme single-exponentiel de Programmation Dynamique qui, à un facteur sous-exponentiel près, calcule en même espace que de temps le nombre pseudo-a-chromatique : d'un arbre ou d'une forêt en $O^*(2^n)$, d'un graphe planaire ou de genre borné par $\overline{g} \in \mathbb{N}$ en $O^*(8^n)$.*

La preuve du Corollaire 2.40 est en tout point similaire à celle du Corollaire 2.15.

2.7.2 Nombre Chromatique Harmonieux

Nombre Chromatique Harmonieux. Une *k-coloration harmonieuse* d'un graphe est une k-coloration propre des sommets telle que chaque paire de couleur apparaît au plus une fois dans les arêtes du graphe colorié. Dans cette Thèse, nous ne considérons jamais les multigraphes. Cette fois-ci uniquement, nous utilisons cette notion pour formuler de manière très claire le problème du Nombre Chromatique Harmonieux, ainsi que sa variante le problème du Nombre Chromatique Pseudo-Harmonieux.

k-Coloration Harmonieuse
Version : *Décision*

Entrée : $G = (V, E)$, $k \in \mathbb{N}^*$
Sortie : Existe-t-il une k-coloration propre de V telle que le multigraphe obtenu par contraction des classes de couleurs soit un graphe simple ?

k-Coloration Pseudo-Harmonieuse
Version : *Décision*

Entrée : $G = (V, E)$, $k \in \mathbb{N}^*$
Sortie : Existe-t-il une k-coloration de V telle que le multigraphe obtenu par contraction des classes de couleurs soit un graphe simple avec au plus une boucle sur chaque sommet ?

Pour un graphe $G = (V, E)$, nous notons le *Nombre Chromatique Harmonieux* $\hbar(G)$ et le *Nombre Chromatique Pseudo-Harmonieux* $\hbar_p(G)$ les plus petits entiers $k \in \mathbb{N}$ pour lesquels G admet respectivement une k-coloration harmonieuse ou une k-coloration pseudo-harmonieuse. Observons que les relations entre les classes de couleur exprimées ici sont similaires à celles exprimées dans les problèmes du Nombre a-Chromatique ou du Nombre Pseudo-a-Chromatique. Ces problèmes diffèrent par la quantification de ces relations et dans leur objectif d'optimisation. ♣

Bien que la notion de nombre chromatique pseudo-harmonieux n'existe pas dans la littérature, il y est fait souvent référence sans distinction de la notion de nombre chromatique harmonieux, les deux notions se retrouvant alors confondues. Pourtant, comme montré par Miller et Pritikin, ces deux problèmes sont différents, et on a pour tout graphe G : $\hbar_p(G) \leq \hbar(G)$ [156]. Nous montrons que tout graphe G vérifie également la relation : $\hbar(G) \leq 2 \cdot \hbar_p(G)$.

Propriété 2.41. *Pour tout graphe G : $\hbar_p(G) \leq \hbar(G) \leq 2 \cdot \hbar_p(G)$.*

Démonstration. Soit φ une k-coloration pseudo-harmonieuse. Tant qu'il y a une classe de couleur qui contient une arête, nous répétons : créer une nouvelle couleur et l'assigner à une des extrémités de cette arête. Or par définition d'une coloration pseudo-harmonieuse, les classes de couleur de φ ne peuvent pas contenir plus d'une arête. Nous avons donc transformé φ en une nouvelle q-coloration, avec $q \leq 2k$, qui est propre et harmonieuse, donc par définition $\hbar(G) \leq q$. Si φ est une k-coloration pseudo-harmonieuse optimale, donc avec $k = \hbar_p(G)$, on a finalement $\hbar(G) \leq q \leq 2 \cdot \hbar_p(G)$. □

Contrairement au nombre a-chromatique, ici c'est bien la notion de coloration pseudo-harmonieuse qui a été introduite en premier par Frank, Harary et Plantholt en 1982. Hopcroft et Krishnamoorthy ont montré en 1983 que le problème du Nombre Chromatique Pseudo-Harmonieux est NP-difficile [114]. La notion de coloration harmonieuse quant à elle été introduite plus tard par Lee

et Mitchem en 1987 [157]. Résoudre le problème du Nombre Chromatique Harmonieux sur un chemin est un problème *facile* [140]. Edwards et Mc Diarmid ont montré en 1995 que c'est un problème NP-difficile si le graphe est un arbre [68]. Plus particulièrement, pour tout $k \in \mathbb{N}$, ils ont montré que décider si $\hbar(G) \leq k$ est un problème NP-complet même pour un arbre de rayon 3.

Le Lemme 2.42 et le Corollaire 2.43 montrent comment l'Algorithme 2.4 de *Programmation Dynamique Arborescente* du Théorème 2.13 peut être adapté aux problèmes du Nombre Chromatique Harmonieux et du Nombre Chromatique Pseudo-Harmonieux, afin d'obtenir des résultats similaires au Corollaire 2.15.

Lemme 2.42. *L'Algorithme 2.4 du Théorème 2.13 peut être adapté pour décider en temps et espace* $O^*(k^{\mathbf{pw}(G)} \cdot 2^{k^2/2})$ *si un graphe* $G = (V, E)$ *admet une* k*-coloration harmonieuse ou une* k*-coloration pseudo-harmonieuse.*

Démonstration. Pour décider si G admet une k-coloration harmonieuse, il suffit dans l'Algorithme 2.4, après avoir généré ξ à la ligne 7, de vérifier si $\xi \cap E_k = \emptyset$ avant d'ajouter la nouvelle caractéristique à la ligne 8. De manière similaire au Lemme 2.39, supprimer la condition de la ligne 6 revient alors à décider si G admet une k-coloration pseudo-harmonieuse. □

Corollaire 2.43. *Soit* $G = (V, E)$ *un graphe satisfaisant* $\Delta(G) = O(\sqrt(n))$*. Alors il y a un algorithme de Programmation Dynamique pour calculer* $\hbar(G)$ *ou* $\hbar_p(G)$ *qui, à un facteur sous-exponentiel près, s'exécute en temps et espace single-exponentiel si* G *est un arbre, une forêt, un graphe planaire ou un graphe de genre borné.*

Démonstration. Nous utilisons la variante de l'Algorithme 2.4 présentée au Lemme 2.42. Soit $G = (V, E)$ un graphe satisfaisant $\Delta(G) = O(\sqrt(n))$ et appartenant à une des classes mentionnée dans ce corollaire. Par la Sous-Section 1.3.2, nous avons $\mathbf{pw}(G) = O(\sqrt{n} \cdot \log n) = o(n/\log n)$. Nous observons alors, avec $\hbar_p(G) \leq \hbar(G) \leq n$, que le facteur $k^{\mathbf{pw}(G)} = 2^{o(n/\log n)\cdot \log n} = 2^{o(n)}$ du temps d'exécution de l'Algorithme du Lemme 2.42 représente un facteur sous-exponentiel. En 1994, Edwards et McDiarmid ont montré - aux Corollaires 2.3, 2.4 et 2.5 - que si G est défini comme dans ce corollaire, alors $\hbar(G) \leq O(\sqrt{m}+\Delta)$ [67]. Lorsqu'il est appliqué à notre graphe G, avec $m = O(n)$, l'Algorithme du Lemme 2.42 s'exécute donc, à un facteur sous-exponentiel près, en temps et espace *single-exponentiel* $O^*(2^{O(n)})$. □

2.8 Conclusion

Tout au long de ce Chapitre, nous avons étudié différents problèmes de coloration, et nous nous sommes attachés à les résoudre de manière exacte. Ces problèmes se distinguent soit par la difficulté d'appartenance d'un sous-ensemble de sommets à une classe de couleur, soit par la difficulté engendrée par les relations entre les classes de couleur. Pour résoudre nos problèmes, nous mettons principalement en œuvre les techniques de *Programmation Dynamique* et d'*Inclusion-Exclusion*, et nos algorithmes doivent constamment tirer profit de la structure de ces problèmes. Nous renvoyons à la Sous-Section 2.1.2 pour un aperçu de nos résultats. Selon le problème étudié, nos méthodes rencontrent plus ou moins de succès et peuvent être appliquées avec plus ou moins d'efficacité. Paradoxalement, la complexité d'un problème seule ne permet pas de juger le succès d'une méthode plutôt qu'une autre. Nous avons vu que le principe d'*Inclusion-Exclusion* peut être appliqué au problème de Clique Coloration, qui est un problème Σ_2^p-difficile, tandis que nous n'avons pas pu réaliser un algorithme d'*Inclusion-Exclusion* pour résoudre le problème NP-difficile du Nombre a-Chromatique.

Outre la constante recherche d'un algorithme toujours plus efficace pour résoudre les problèmes étudiés, nos travaux dans ce Chapitre s'achèvent par des questions ouvertes. Une question d'intérêt majeur concerne la portée du principe d'*Inclusion-Exclusion*. Aujourd'hui encore, nous ne connaissons pas les limites d'application de cette technique. Au départ utilisée pour des algorithmes de partitionnement où seules les propriétés des classes de couleur avaient une importance, nous cherchons aujourd'hui à connaître si le principe d'*Inclusion-Exclusion* peut être appliqué à des problèmes où les relations entre les classes de couleur revêtent également une importance.
Nous avons appliqué ce principe au problème du Nombre b-Chromatique. Pour cela, nous avons étendu le champ d'application de la technique aux problèmes de Partitions Étiquetées. Cependant, ces problèmes ne mettent pas en avant des relations entre les classes de couleur, mais des attributs supplémentaires aux classes, des étiquettes, qui viennent s'ajouter à un problème de partitionnement sans contraintes entre les classes. C'est ainsi en traduisant une instance du problème du Nombre b-Chromatique en instance de problème de Partition Étiquetée que nous avons pu résoudre ce problème par le principe d'*Inclusion-Exclusion*. Cette traduction nous a ainsi libéré de la contrainte intiale entre les classes de couleurs. Il semble toutefois que cette technique de *Traduction* ait ses propres limites. Derrière notre algorithme de résolution du Nombre b-Chromatique se cache tout un pan d'énumération *Brute-Force*, dont nous ne savons pas si nous pouvons nous libérer. Pour les autres problèmes étudiés qui mettent en avant des relations entre les classes de couleurs, leur traduction en un autre problème dans le champ d'application déjà existant du principe d'*Inclusion-Exclusion* ne semble pas immédiate.

Quelles sont les limites du champ d'application de la technique d'Inclusion-Exclusion aux problèmes de coloration qui font apparaître des contraintes entre les classes de couleur ?

Nous avons vu tout au long de ce Chapitre que la technique de *Programmation Dynamique Classique* peut être appliquée dans certains cas même lorsque le problème présente de fortes relations entre les classes de couleurs. C'est par exemple le cas pour le problème du Nombre Grundy Partiel. Nous ne l'avons pas écrit, mais avant de réaliser un algorithme d'*Inclusion-Exclusion*, notre premier essai pour résoudre le problème du Nombre b-Chromatique était un algorithme de *Programmation Dynamique Classique* en temps $O^*(4^n)$. À l'instar du problème du Nombre Chromatique, certains problèmes de coloration qui admettent un algorithme de *Programmation Dynamique Classique* peuvent donc être résolus par un algorithme d'*Inclusion-Exclusion*, technique dont on rappelle qu'elle est elle-même une variante de *Programmation Dynamique Classique*.

Les algorithmes de Programmation Dynamique Classique pour des problèmes de coloration peuvent-ils tous être améliorés ou à défaut traduits en un algorithme d'Inclusion-Exclusion ?

Il y a des problèmes de coloration pour lesquels nous avons échoué à fournir un algorithme de résolution non trivial. C'est le cas par exemple des problèmes du Nombre a-Chromatique ou de ses variantes en présentées en Section 2.7. Certaines familles de problèmes de coloration se distinguent par la caractérisation du sous-graphe induit que doit former chaque paire de classes de couleur. C'est le cas par exemple du problème de Nombre Étoile-Chromatique ou de Nombre Acyclique-Chromatique, traduits respectivement de l'anglais *star coloring* et *acyclic coloring*.

Pour tous les problèmes cités au paragraphe ci-dessus, peut-on construire un algorithme en temps single exponentiel pour les résoudre ?

Chapitre 3

Enumération de Transversaux d'Hypergraphes

Sommaire

3.1 Introduction

Nous renvoyons à la Section 1.4 pour une introduction à la terminologie des hypergraphes. L'un des plus célèbres problèmes d'hypergraphes est l'énumération des transversaux minimaux. Berge a proposé en 1984 un algorithme séquentiel pour ce problème sur les hypergraphes en général [17]. Pour l'Algorithmique Exacte Exponentielle, le temps d'exécution au pire de cas de cet algorithme est difficile à évaluer.
Il existe une autre mesure du temps d'exécution des algorithmes pour des problèmes d'énumération. Nous avons vu en Sous-Section 2.6.2, que les Algorithmes 2.5 et 2.6 du Théorème 2.31 énumèrent une clôture $\mathcal{F}^*$ d'une famille $\mathcal{F}$ d'objets quelconques en temps $O^*(|\mathcal{F}|)$. À un facteur $O(n^2 \cdot \log n)$ près, le temps d'exécution de ces algorithmes est linéaire en la taille des clôtures. Ces algorithmes sont dits *output-polynomial*. Nous référons à la Définition 1.1 de la Thèse de Maury pour une introduction formalisée aux algorithmes *output-polynomial* et à *délai polynomial* [152].
L'Algorithme de Berge a été évalué sous l'angle de cette analyse différente du temps d'exécution. Takata a montré en 2008 que cet algorithme n'est pas *output-polynomial*. De nombreux nouveaux

essais ont été réalisés afin d'obtenir un algorithme output-polynomial pour le problème d'énumération des transversaux minimaux dans un hypergraphe en général [13]. Parmi eux, Kavvadias et Stavropoulos ont présenté en 2005 un algorithme qui utilise un espace polynomial [122]. Malheureusement, Hagen a montré en 2008 qu'aucun de ces essais n'a été concluant [106]. Le problème s'est donc déporté vers la restriction de ce problème à certaines classes d'hypergraphes. Nous référons à la Thèse de Maury pour une étude exhaustive des classes d'hypergraphes qui admettent un algorithme output-polynomial pour l'énumération des transversaux minimaux [152].

Dans ce Chapitre, nous nous intéressons au temps d'exécution au pire des cas des meilleurs algorithmes exacts exponentiels pour l'énumération des transversaux minimaux d'un hypergraphe $\mathcal{H}$ de rang borné par $k \in \mathbb{N}_{\geq 3}$, pour la version énumération du problème k-Hitting Set. Par les Propriétés 2.25, un algorithme trivial pour ce problème s'exécute en temps $O^*(n^k \cdot 2^n) = O^*(2^n)$.
Eiter, Gottlob et Makino ont proposé en 2003 un algorithme à délai polynomial [69, Thm. 4.1]. Pour cet algorithme, le délai entre la génération de deux solutions est $O^*(n^k \cdot |Tr(\mathcal{H})|)$. Par conséquent, nous pouvons affirmer que l'Algorithme de Eiter, Gottlob et Makino est output-polynomial et s'exécute en temps $O^*(|Tr(\mathcal{H})|^2)$. Malheureusement, le Lemme 3.1 de la Sous-Section 3.1.2 montre que cet algorithme ne conduit pas directement à un algorithme en temps $O^*((1+\varepsilon_k)^n)$ pour la version énumération du problème k-Hitting Set, avec $0 < \varepsilon_k < 1$ une constante réelle qui dépend de $k \in \mathbb{N}_{\geq 2}$.

D'autres versions du problème k-Hitting Set ont montré un grand intérêt, comme c'est particulièrement le cas de la version optimisation, donc du problème de déterminer un transversal minimum dans un hypergraphe de rang $k \in \mathbb{N}_{\geq 2}$. Nous avons mentionné en Sous-Section 1.3.4 que pour $k = 2$, il y a un algorithme en temps $O^*(1.2109^n)$ pour résoudre ce problème [172]. Il y a de nombreuses tentatives pour essayer d'améliorer ce résultat. Valider et accepter de tels résultats, pourtant produits par des auteurs renommés, est très difficile tant les algorithmes deviennent très techniques. À ce titre, on peut citer[1] la tentative d'algorithme en temps $O^*(1.2002^n)$ de Xiao et Nagamochi[2] publiée en 2013 [190]. Pour des hypergraphes de rang supérieur, Wahlström a produit dans sa Thèse en 2003 deux algorithmes respectivement en temps $O^*(1.6538^n)$ et $O^*(1.6316^n)$ avec espace polynomial et exponentiel pour déterminer un transversal minimum dans un hypergraphe de rang $k = 3$ [187, 186]. Le temps du deuxième algorithme a été révisé à $O^*(1.6278^n)$ en 2008 dans l'article de Fomin et al. [82]. La plupart de ces algorithmes sont des algorithmes de *Branchement* qui utilisent la technique de *Mesurer et Conquérir*.
Dans son rapport de Master, Gaspsers a introduit en 2005 des algorithmes génériques pour le problème de Transversal Minimum [95]. Ces algorithmes en espace polynomiaux s'exécutent pour tout $k \in \mathbb{N}_{\geq 2}$ en temps $\tau(1, \ldots, k)$. La technique utilisée par ces algorithmes est inspirée de celle de Moon et Moser développée en 1965 pour l'énumération des transversaux minimaux dans un hypergraphe de rang 2 [159]. En 2014, nous avons publié un article qui montre que les algorithmes de Gaspers peuvent être adaptés pour l'énumération de tous les transversaux minimaux dans un hypergraphe de rang $k \geq 2$ [42]. Ces algorithmes peuvent en plus être modifiés pour s'adapter au problème co$k$-Hitting Set. Les algorithmes obtenus s'exécutent alors en temps $\tau(2, 2, \ldots, k, k) < \tau(1, \ldots, k) < 2$. Ces résultats, par la réduction du Lemme 2.37 présentée en Sous-Section 2.6.4, nous ont permis d'obtenir dans le Théorème 2.38 les meilleurs algorithmes connus jusqu'à présent pour décider la 2-clique-colorabilité d'un graphe G satisfaisant $\omega(G) \leq k$.

1. Citons également la tentative toujours encore en construction de Robson disponible sur son internet personnel depuis 2001, dont l'objectif affiché est d'obtenir un algorithme en temps $O^*(1.1893^n)$ [173].
2. Les auteurs présentent une analyse améliorée en temps $O^*(1.1996^n)$ sur arXiv [189].

Pour ce Chapitre, nous notons $O^*(\alpha_k^n)$, $O^*(\beta_k^n)$, $O^*(\gamma_k^n)$ et $O^*(\lambda_k^n)$ les temps d'exécution des meilleurs algorithmes connus respectivement pour déterminer un transversal minimum, dénombrer les minimums, énumérer les transversaux minimaux et les dénombrer dans un hypergraphe de rang $k \in \mathbb{N}_{\geq 2}$ à n sommets. Nous notons de plus $O^*(\overline{\gamma}_k^n)$ et $O^*(\overline{\lambda}_k^n)$ les temps d'exécution respectifs des meilleurs algorithmes connus pour énumérer et compter dans ces hypergraphes les transversaux minimaux dont le complémentaire est un transversal. La Table 3.1 résume les temps d'exécution de ces meilleurs algorithmes connus jusqu'à présent. Certaines de ces valeurs peuvent être retrouvées dans l'article de Fomin et al. [82] ainsi que dans le livre de Gaspers [96]. Cette Table prend également en compte notre publication de 2014 [42].

k	α_k	β_k	γ_k	λ_k	$\overline{\gamma}_k$	$\overline{\lambda}_k$
2	1.2109	1.2377	1.4423	1.4423	1.4423	1.4423
3	1.6278	1.7198	1.8393*	1.8393*	1.7693*	1.7693*
4	1.8704	1.8997	1.9276*	1.9276*	1.8994*	1.8994*
5	1.9489	1.9594	1.9660*	1.9660*	1.9536*	1.9536*
6	1.9781	1.9824	1.9836*	1.9836*	1.9779*	1.9779*

Table 3.1 – Résumé des temps d'exécution des meilleurs algorithmes connus et publiés à ce jour pour différentes versions associées au problème k-Hitting Set [82, 96, 42]. Les temps d'exécution marqués d'un * ont été publiés en 2014[42].

3.1.1 Nos Résultats

Ce chapitre présente des algorithmes d'énumération de transversaux minimaux dans un hypergraphe de rang k. Les algorithmes que nous allons présenter pour l'énumération des transversaux minimaux dans un hypergraphe de rang k sont des algorithmes de *Branchement* utilisant la technique de *Mesurer et Conquérir*, ou encore utilisant la technique de *Compression Iterative*. Ces techniques sont introduites dans la Sous-Section 1.5.2.

Dans un premier temps, dans la Section 3.2, nous présentons les outils utilisés tout au long de ce chapitre. En particulier, dans la Sous-Section 3.2.1, nous présentons des propriétés fondamentales sur les vecteurs de branchement. Nous introduisons des notations et premières propriétés, des outils de comparaison, et des opérations sur les vecteurs de branchement. Egalement, dans la Sous-Section 3.2.2, nous définissons et étudions les propriétés d'une famille de vecteurs de branchement, les vecteur d'entiers. Finalement, dans la Sous-Section 3.2.3, nous définissons une instance des sous-problèmes de nos algorithmes, et la façon dont nous décrivons nos algorithmes de branchement.

Puis nous établissons entre autres les résultats principaux suivants :

- ◇ Premièrement, dans la Section 3.3, nous utiliserons les techniques de *Branchement* et de *Mesurer et Conquérir* pour construire un algorithme en temps $O^*(1.6755^n)$ pour l'énumération des transversaux minimaux d'un hypergraphe de rang $k = 3$.
- ◇ Deuxièmement, dans la Section 3.4, nous établirons des algorithmes de *Branchement* pour le problème k-Hitting Set, qui constituent une forme généralisée de ceux présentés en 2014 [42].
- ◇ Troisièmement, dans la Section 3.5, nous montrons comment la technique de *Compression Iterative* peut être appliquée à nos problèmes pour améliorer nos résultats.
- ◇ Finalement, en conclusion, nous résumons nos résultats et présentons les améliorations apportées aux algorithmes déjà existants et en considérant toutes les versions de nos problèmes.

La Table 3.2 présente un résumé des résultats obtenus dans cette Thèse pour ce chapitre.

k	α_k	β_k	γ_k	λ_k	$\overline{\gamma}_k$	$\overline{\lambda}_k$
2	1.2109	1.2377	1.4423	1.4423	1.4423	1.4423
3	1.6278	1.6755	1.6755	1.6755	1.6755	1.6755
4	1.8704	1.8866	1.8866	1.8866	1.8866	1.8866
5	1.9489	1.9538	1.9538	1.9538	1.9536*	1.9536*
6	1.9779	1.9779	1.9779	1.9779	1.9779*	1.9779*

Table 3.2 – Résumé des temps d'exécution améliorés dans cette Thèse en comparaison de ceux présentés dans la Table 3.1 pour les problèmes de k-Hitting Set et co$k$-Hitting Set. Les temps d'exécution marqués d'un * ont été publiés en 2014[42].

3.1.2 Bornes Combinatoires

Lemme 3.1. *Pour tout $k \in \mathbb{N}_{\geq 2}$, il y a une famille infinie $\mathcal{F}_k$ d'hypergraphes $\mathcal{H}$ de rang k qui satisfont :*

$$\binom{2k-1}{k}^{1/(2k-1)} \leq \log_{n_{\mathcal{H}}}(|Tr(\mathcal{H})|) \leq 2$$

où $n_{\mathcal{H}}$ est le nombre correspondant de sommets d'un hypergraphe $\mathcal{H}$ de cette famille.

Démonstration. Soit $k \in \mathbb{N}_{\geq 2}$, et soit $\mathcal{H}_k$ l'hypergraphe à $2k-1$ sommets et dont les hyperarêtes sont les sous-ensembles de k sommets.
Nous étudions dans un premier temps des propriétés de $\mathcal{H}_k$. Nous montrons que toute hyperarête $e \in \mathcal{H}_k$ est un transversal de $\mathcal{H}_k$. En effet, supposons qu'il existe une hyperarête $e' \in \mathcal{H}_k$ telle que $e' \subseteq \overline{e}$. Puisque $\mathcal{H}_k$ est composé de $2k-1$ sommets et que par construction e contient exactement k sommets, alors $|e'| \leq 2k-1-k \leq k-1$. Cela contredit que toute hyperarête de $\mathcal{H}_k$ est de taille k, et donc toute hyperarête $e \in \mathcal{H}_k$ est un transversal de $\mathcal{H}_k$. Nous montrons que toute hyperarête $e \in \mathcal{H}_k$ est un transversal minimal. En effet, supposons par contradiction que e ne soit pas minimale. Par les Propriétés 2.25, il existe $v \in e$ tel que $e - v$ soit un transversal de $\mathcal{H}_k$. Cependant, $|\overline{e-v}| = 2k-1-(k-1) = k$. Alors par construction, $\overline{e-v} \in \mathcal{H}_k$ est une hyperarête de $\mathcal{H}_k$, incluse dans le complémentaire de $e-v$. Ceci contredit que $e-v$ soit un transversal de $\mathcal{H}_k$. Nous avons donc montré $\forall\, e \in \mathcal{H}_k : e \in Tr(\mathcal{H}_k)$, autrement dit, nous avons montré $\mathcal{H}_k = Tr(\mathcal{H}_k)$.
À présent, soit $\mathcal{H}^q$ un hypergraphe composé de $n_q = q \cdot (2k-1)$ sommets, $q \in \mathbb{N}^*$, et constitué de q union disjointes de copies de $\mathcal{H}_k$. Un transversal minimal de $\mathcal{H}^q$ est alors une union disjointe de transversaux minimaux de $\mathcal{H}_k$. Pour chaque copie de $\mathcal{H}_k$, il y a $|\mathcal{H}_k|$ transversaux minimaux distincts. Ainsi :

$$|Tr(\mathcal{H}^q)| = |\mathcal{H}_k|^q = \binom{2k-1}{k}^{n_q/(2k-1)}$$

Nous avons donc construit la famille infinie $\mathcal{F}_k = \cup_{q \in \mathbb{N}^*} \mathcal{H}^q$ composée des hypergraphes $\mathcal{H}^q$, $q \in \mathbb{N}^*$, qui satisfont les propriétés stipulées dans le lemme. Remarquons au passage que pour tout $k \in \mathbb{N}_{\geq 2}$, tous les transversaux minimaux de $\mathcal{H}_k$ sont en fait des transversaux minimums, et cela en est de même pour tout hypergraphe $\mathcal{H}^q$ de la famille $\mathcal{F}_k$. □

Le Lemme 3.1 généralise la borne inférieure connue sur le nombre de transversaux minimaux d'un hypergraphe de rang $k = 2$, donc le nombre $O^*(3^{n/3})$ de couvertures de sommets minimales (de

manière équivalente nombre de complémentaires de stables maximaux) dans un graphe (voir Sous-Section 1.3.4). Cependant, contrairement au résultat de Moon et Moser [159], nous ne savons pas si la borne inférieure du Lemme 3.1 constitue également une borne supérieure pour $k \geq 3$. L'intérêt du Lemme 3.1 réside toutefois dans une borne inférieure sur le temps d'exécution d'un algorithme exact exponentiel pour l'énumération des transversaux minimaux d'un hypergraphe de rang k.

Théorème 3.2. *Pour tout $k \in \mathbb{N}_{\geq 2}$, un algorithme pour la version énumération du problème k-*Hitting Set *s'exécute en temps $\Omega(c_k^n)$, où $1 < c_k < 2$ est une constante réelle qui dépend de k : $c_k = ((2k-1)!/(k!(k-1)!))^{1/(2k-1)}$.*

En utilisant la formule de Stirling, on observera que la suite c_k est croissante avec $\lim_{k\to\infty} c_k = 2$. La Table 3.3 présente les valeurs c_k pour certaines valeurs de k.

k	2	3	4	5	6	7	8	9	10
c_k	1.4422	1.5848	1.6618	1.7114	1.7467	1.7734	1.7943	1.8112	1.8253

k	15	20	25	50	75	100	200	500	1000
c_k	1.8709	1.8962	1.9127	1.9495	1.9636	1.9713	1.9839	1.9926	1.9959

Table 3.3 – Bornes inférieures $\Omega(c_k^n)$ du Théorème 3.2 pour le temps d'exécution d'un algorithme en temps $O^*(\gamma_k^n)$ pour la version énumération du problème k-Hitting Set.

3.2 Étude du problème

3.2.1 Algorithmes de branchement

Un vecteur de branchement est un vecteur $A \in (\mathbb{R}_+^*)^{m_A}$ de réels strictement positifs, avec $m_A \in \mathbb{N}_{\geq 2}$. Pour des raisons de compatibilité et pour alléger l'écriture, on pourra définir un tel vecteur lorsque $m_A = 1$. Pour un vecteur $A \in (\mathbb{R}_+^*)^{m_A}$, on note $A = (a_1, \ldots, a_{m_A})$ ce vecteur, et on appelle $m_A \in \mathbb{N}^*$ sa *dimension*. Si $m_A \geq 2$, on note $\tau(A) = \alpha_A > 1$ le nombre de branchement de A, défini comme l'unique racine réelle positive du polynôme $P_A(X) = 1 - \sum_{i=1}^{i=m_A} X^{-a_i}$.
La Proposition 1.1 est fondamentale, elle montre en particulier que tout vecteur de branchement admet bien un nombre de branchement, que ce dernier est unique, et qu'il satisfait certaines propriétés.

Proposition 1.1. *Pour tous vecteurs de branchement $A \in (\mathbb{R}_+^*)^{m_A}$ et $B \in (\mathbb{R}_+^*)^{m_B}$, avec m_A, $m_B \geq 2$, les propriétés suivantes sont satisfaites :*

(i) Le polynôme $P_A(X)$ est croissant sur $\mathbb{R}_+^*$, avec $\lim_{X\to 0} P_A(X) = -\infty$, $P_A(1) \leq -1$, et $\lim_{X\to+\infty} P_A(X) = 1$.

(ii) $P_A(X)$ admet une unique racine réelle α_A qui existe, et tel que $\alpha_A \in \mathbb{R}_{>1}$.

(iii) $\alpha_A < \alpha_B$ si et seulement si $P_A(\alpha_B) > 0$ si et seulement si $P_B(\alpha_A) < 0$.

Démonstration.

(i) En observant que le polynôme dérivé $P'_A(X) = \sum_{i=1}^{i=m_A} a_i X^{-a_i-1}$ de P_A est positif sur $\mathbb{R}_+^*$, on a que P_A est croissant sur ce même intervalle. De plus, on observe que $P_A(1) = 1 - m_A$. Avec $m_A \geq 2$, on a $P_A(1) \leq -1$.

(ii) D'après le Théorème de d'Alembert-Gauss, le corps $\mathbb{C}$ des nombres complexes est algébriquement clos, autrement dit tout polynôme à coefficients complexes admet une solution dans $\mathbb{C}$. P_A est un polynôme à coefficients réels, et $\mathbb{R}$ étant un sous-corps de $\mathbb{C}$, P_A est donc en particulier également un polynôme à coefficients complexes, donc admet entre une et m_A solutions dans $\mathbb{C}$. Nous affirmons qu'au moins une de ses solutions α_A est dans $\mathbb{R}$, avec $\alpha_A > 1$. En effet, d'après (i), le polynôme $P_A(X)$ est croissant sur $\mathbb{R}_+^*$, avec $P_A(1) = 1 - m_A \leq -1$ et $\lim_{X\to+\infty} P_A(X) = 1$. Le Théorème des valeurs intermédiaires nous permet de déduire qu'il existe $\alpha_A \in \mathbb{R}_{>1}$ tel que $P_A(\alpha_A) = 0$. De plus, ce même théorème nous permet de conclure que α_A est l'unique racine réelle de P_A.

(iii) Les équivalences sont une conséquence de la croissance de $P_A(X)$ et $P_B(X)$ sur $\mathbb{R}_+^*$.

□

Nous définissons dans un premier temps des opérateurs de comparaison sur les vecteurs de branchement.

Définition 3.3. Soient $A \in (\mathbb{R}_+^*)^{m_A}$ et $B \in (\mathbb{R}_+^*)^{m_B}$ deux vecteurs de branchement.

(i) *égalité de vecteurs* : Si $m_A = m_B$ et s'il existe une permutation $\pi \in \mathfrak{S}_{m_A}$ telle que $\forall\ 1 \leq i \leq m_A : a_i = b_{\pi(i)}$, alors on dit que $A =_\pi B$. De plus, si π est la permutation identité, alors on note $A = B$.

(ii) *inégalité de vecteurs* : Si $\tau(A) > \tau(B)$, on dit que $A >_\tau B$ ou que $B <_\tau A$. De même, si $\tau(A) \geq \tau(B)$, on dit que $A \geq_\tau B$ ou que $B \leq_\tau A$. Finalement, si $\tau(A) = \tau(B)$, on dit que $A =_\tau B$.

Remarquons que les opérateurs d'égalité $=_\tau$, $=_\pi$ et $=$ définissent des relations d'équivalence sur l'ensemble des vecteurs de branchement, tandis que les opérateurs de comparaison $\leq_\tau$ et $\geq_\tau$ définissent quant à eux des relations d'ordre sur ce même ensemble. Remarquons également que si $A <_\tau B$, alors $A \leq_\tau B$, de même que si $A >_\tau B$, alors $A \geq_\tau B$. Remarquons finalement que $A = B \Rightarrow A =_\pi B \Rightarrow A =_\tau B$, et que $A =_\tau B \Rightarrow A \leq_\tau B$, de même que $A =_\tau B \Rightarrow A \geq_\tau B$.
La proposition suivante étudie des conditions d'égalité ou d'inégalité de vecteurs de branchement.

Proposition 3.4. *Soit $A \in (\mathbb{R}_+^*)^{m_A}$ un vecteur de branchement. Etant donnée une permutation $\pi \in \mathfrak{S}_{m_A}$, on note $A_\pi = (a_{\pi(1)}, \ldots, a_{\pi(m_A)})$. Alors, pour tous vecteurs de branchement $A \in (\mathbb{R}_+^*)^{m_A}$ et $B \in (\mathbb{R}_+^*)^{m_B}$, les propriétés suivantes sont satisfaites :*

(i) $\forall\ \pi \in \mathfrak{S}_{m_A}\ :\ A_\pi =_\tau A$

(ii) $\forall\ \varepsilon \in \mathbb{R}_+^*\ :\ (a_1, \ldots, a_{m_A}, \varepsilon) >_\tau A$

(iii) $\forall\ 1 \leq i \leq m_A,\ \forall\ \varepsilon \in \mathbb{R}_+^*\ :\ (a_1, \ldots, a_{i-1}, a_i + \varepsilon, a_{i+1}, \ldots, a_{m_A}) <_\tau A$

(iv) Si $\tau(A) = \tau(B)$, alors
$\forall\ 1 \leq i \leq m_A\ :\ A =_\tau (a_1, \ldots, a_{i-1}\ ,\ a_i + b_1, \ldots, a_i + b_{m_B}\ ,\ a_{i+1}, \ldots, a_{m_A})$

Démonstration.

(i) En observant que $P_A = P_{A_\pi}$, on a $P_{A_\pi}(\alpha_A) = P_A(\alpha_A) = 0$. Et ainsi $\alpha_{A_\pi} = \alpha_A$, ce qui montre que $A_\pi =_\tau A$. De plus par définition de l'égalité $=$ des vecteurs de branchement, il vient $A_\pi = A$.

(ii) On note $P_{A,\varepsilon}$ le polynôme caractéristique du vecteur de branchement $(a_1, \ldots, a_{m_A}, \varepsilon)$. On observe alors que $P_{A,\varepsilon}(X) = P_A(X) - X^{-\varepsilon}$. $P_{A,\varepsilon}(\alpha_A) = -\alpha_A^{-\varepsilon} < 0$ pour tout $\varepsilon \in \mathbb{R}_+^*$. Ainsi, pour tout $\varepsilon \in \mathbb{R}_+^*,\ P_{A,\varepsilon}(\alpha_A) < 0$, alors d'après la Proposition 1.1 (iii), on a $\alpha_A < \tau(a_1, \ldots, a_{m_A}, \varepsilon)$.

(iii) On note $P_{A,i,\varepsilon}$ le polynôme caractéristique du vecteur de branchement $A_{i,\varepsilon} = (a_1, \ldots, a_{i-1}, a_i + \varepsilon, a_{i+1}, \ldots, a_{m_A})$. On a alors :

$$\begin{array}{rcl} P_{A,i,\varepsilon}(X) & = & 1 - \sum_{j=1}^{j=m_A, j\neq i} X^{-a_j} - X^{-a_i-\varepsilon} \\ & = & 1 - \sum_{j=1}^{j=m_A} X^{-a_j} - X^{-a_i-\varepsilon} + X^{-a_i} \\ & = & P_A(X) - X^{-a_i-\varepsilon} + X^{-a_i} \\ & = & P_A(X) + X^{-a_i}(1 - X^{-\varepsilon}) \end{array}$$

Avec $P_A(\alpha_A) = 0$, on calcule $P_{A,i,\varepsilon}(\alpha_A)$:

$$\begin{array}{rcl} P_{A,i,\varepsilon}(\alpha_A) & = & P_A(\alpha_A) + \alpha_A^{-a_i}(1 - \alpha_A^{-\varepsilon}) \\ P_{A,i,\varepsilon}(\alpha_A) & = & \alpha_A^{-a_i}(1 - \alpha_A^{-\varepsilon}) \end{array}$$

$P_{A,i,\varepsilon}(\alpha_A)$ se présente donc comme un produit de deux facteurs. Puisque $\alpha_A > 1$ d'après la Proposition 1.1 (ii), on en déduit pour le premier membre $\alpha_A^{-a_i} > 0$. Avec $\alpha_A > 1$, et $\varepsilon > 0$, on déduit $\varepsilon \ln(\alpha_1) > 0$, et donc $\alpha_A^{\varepsilon} > 1$, ou encore $\alpha_A^{-\varepsilon} < 1$. Ainsi, pour le deuxième membre, on a $1 - \alpha_A^{-\varepsilon} > 0$. Les deux membres facteurs du produit $P_{A,i,\varepsilon}(\alpha_A)$ étant strictement positifs, on déduit $P_{A,i,\varepsilon}(\alpha_A) > 0$. En conséquence, on déduit d'après la Proposition 1.1 (iii) que $\alpha_{A,i,\varepsilon} < \alpha_A$.

(iv) On considère le vecteur de branchement $A_i = (a_1, \ldots, a_{i-1}\ ,\ a_i + b_1, \ldots, a_i + b_{m_B}\ ,\ a_{i+1}, \ldots, a_{m_A})$. On note $P_{A,i}$ son polynôme caractéristique, et $\alpha_{A,i}$ son vecteur de branchement. Soit $\tau(A) = \tau(B)$.
On a alors :

$$\begin{array}{rcl} P_{A,i}(X) & = & 1 - \sum_{j=1}^{j=m_A, j\neq i} X^{-a_j} - \sum_{j=1}^{j=m_B} X^{-a_i-b_j} \\ & = & 1 - \sum_{j=1}^{j=m_A, j\neq i} X^{-a_j} - X^{-a_i}(\sum_{j=1}^{j=m_B} X^{-b_j}) \end{array}$$

Par définition de α_B, on a $P_B(\alpha_B) = 0$. Puisque $\alpha_A = \alpha_B$, on a $P_B(\alpha_A) = 0$.
Avec $1 - P_B(X) = \sum_{j=1}^{j=m_B} X^{-bj}$, on obtient $\sum_{j=1}^{j=m_B} \alpha_A^{-bj} = 1 - P_B(\alpha_A) = 1$.

$$\begin{array}{rcl} P_{A,i}(\alpha_A) & = & 1 - \sum_{j=1}^{j=m_A, j\neq i} \alpha_A^{-a_j} - \alpha_A^{-a_i}(\sum_{j=1}^{j=m_B} \alpha_A^{-b_j}) \\ & = & 1 - \sum_{j=1}^{j=m_A, j\neq i} \alpha_A^{-a_j} - \alpha_A^{-a_i} \\ & = & 1 - \sum_{j=1}^{j=m_A} \alpha_A^{-a_j} \\ & = & P_A(\alpha_A) \end{array}$$

Nous avons donc montré que $P_{A,i}(\alpha_A) = P_A(\alpha_A) = 0$, ce qui montre que α_A est racine réelle positive de $P_{A,i}$, et donc que $\alpha_{A,i} = \alpha_A$.

□

On définit les opérations $\odot$, $\oplus$, et $\otimes$ sur les vecteurs de branchement. L'introduction de ces opérations a pour objectif de simplifier par la suite l'écriture de vecteurs de branchement obtenus par une combinaison de ces opérations.

Définition 3.5. Le *décalage scalaire* $\odot$ d'un vecteur de branchement $A \in (\mathbb{R}_+^*)^{m_A}$ par un réel positif ou nul $\varepsilon \in \mathbb{R}_+$ est defini par :

$$\begin{array}{rccccc} \odot & : & \mathbb{R}_+ & \times & (\mathbb{R}_+^*)^m & \to & (\mathbb{R}_+^*)^m \\ & & \varepsilon & \odot & (a_1, \ldots, a_{m_A}) & \mapsto & (a_1 + \varepsilon, \ldots, a_{m_A} + \varepsilon) \end{array}$$

Définition 3.6. La *concaténation vectorielle* $\bigoplus$ d'un vecteur de branchement $A \in (\mathbb{R}_+^*)^{m_A}$ par un vecteur de branchement $B \in (\mathbb{R}_+^*)^{m_B}$ est définie par :

$$\begin{array}{ccccccc} \bigoplus & : & (\mathbb{R}_+^*)^{m_1} & \times & (\mathbb{R}_+^*)^{m_2} & \to & (\mathbb{R}_+^*)^{m_1+m_2} \\ & & (a_1,\dots,a_{m_1}) & \bigoplus & (b_1,\dots,b_{m_2}) & \mapsto & (a_1,\dots,a_{m_1},b_1,\dots,b_{m_2}) \end{array}$$

Définition 3.7. L'*extension vectorielle* $\bigotimes$ d'un vecteur de branchement $A \in (\mathbb{R}_+^*)^{m_A}$ par un vecteur de branchement $B \in (\mathbb{R}_+^*)^{m_B}$ est définie par :

$$\begin{array}{ccccccc} \bigotimes & : & (\mathbb{R}_+^*)^{m_1} & \times & (\mathbb{R}_+^*)^{m_2} & \to & (\mathbb{R}_+^*)^{m_1\times m_2} \\ & & (a_1,\dots,a_{m_1}) & \bigotimes & (b_1,\dots,b_{m_2}) & \mapsto & \bigoplus_{i=1}^{i=m_1} (a_i \bigodot (b_1,\dots,b_{m_2})) \end{array}$$

La proposition suivante énumère un ensemble de propriétés fondamentales sur les opérations précédemment définies. Ces propriétés sont les bases de nos futures démonstrations. Elles s'appuyent sur les propriétés des opérations dans le corps des réels, et sur la Proposition 3.4.

Proposition 3.8. *Pour tous ε, ε_1, $\varepsilon_2 \in \mathbb{R}_+$, et pour tous A, B, C, D vecteurs de branchement de dimensions respectives m_A, m_B, m_C et m_D, les propriétés suivantes sont satisfaites au regard des opérateurs de comparaison des vecteurs de branchement :*

(i) $\varepsilon_1 \bigodot (\varepsilon_2 \bigodot A) = (\varepsilon_1 + \varepsilon_2) \bigodot A = \varepsilon_2 \bigodot (\varepsilon_1 \bigodot A)$

(ii) La concaténation et l'extension vectorielle sont commutatives :
$A \bigoplus B =_\pi B \bigoplus A$
$A \bigotimes B =_\pi B \bigotimes A$

(iii) La concaténation vectorielle est associative :
$A \bigoplus (B \bigoplus C) = (A \bigoplus B) \bigoplus C$

(iv) Le décalage scalaire est distributif par rapport à la concaténation :
$\varepsilon \bigodot (A \bigoplus B) = (\varepsilon \bigodot A) \bigoplus (\varepsilon \bigodot B)$

(v) $\varepsilon \bigodot (A \bigotimes B) = (\varepsilon \bigodot A) \bigotimes B = A \bigotimes (\varepsilon \bigodot B)$

(vi) L'extension vectorielle est distributive par rapport à la concaténation :
$A \bigotimes (B \bigoplus C) =_\pi (A \bigotimes B) \bigoplus (A \bigotimes C)$

3.2.2 Vecteurs d'entiers

On se restreint aux vecteurs de branchement de la forme $(\mathbb{N}^*)^m$, avec $m \in \mathbb{N}_{\geq 2}$. On appelle *vecteur d'entiers* un tel vecteur de branchement. Soit $A \in (\mathbb{N}^*)^{m_A}$ un vecteur d'entiers. On note $A = (a_1,\dots,a_{m_A})$ ce vecteur, avec $\forall\ 1 \leq i \leq m_A : a_i \in \mathbb{N}^*$, et m_A est appelée la *dimension* du vecteur d'entiers A. Pour faciliter la compréhension, on pourra écrire $(a_1,\dots,a_{m_A})_{m_A}$ le vecteur A pour mettre en avant sa dimension.

Définition 3.9. $\forall\ k \in \mathbb{N}_{\geq 2}$, on définit le vecteur d'entiers $\mathbb{1}^{\to}_{k} = (1,\dots,k) \in (\mathbb{N}^*)^k$.

Proposition 3.10. $\forall\ k \in \mathbb{N}_{\geq 2}$, $\forall\ 1 \leq \ell \leq k-1$: $\mathbb{1}^{\to}_{k} = \mathbb{1}^{\to}_{\ell} \bigoplus \ell \bigodot \mathbb{1}^{\to}_{k-\ell}$

Théorème 3.11. *([42, Thm 4]) Soit $k \in \mathbb{N}_{\geq 2}$. Il y a un algorithme de Branchement en temps $O^*(\gamma_k^n)$ avec espace polynomial pour énumérer les transversaux minimaux d'un hypergraphe de rang k, avec $\gamma_k = \tau(1,\dots,k) < 2$.*

Proposition 3.12. $\forall\ k \in \mathbb{N}_{\geq 2}$, *la propriété suivante est satisfaite :* $(1,1) >_\tau \mathbb{1}^{\to}_{k+1} >_\tau \mathbb{1}^{\to}_{k}$. *De plus,* $\lim_{k\to+\infty} \tau(\mathbb{1}^{\to}_{k}) = 2$.

Démonstration. Dans un premier temps, montrons que $\forall\ k \geq 2$, nous avons $\mathbb{1}_{\underset{k+1}{\rightarrow}} >_\tau \mathbb{1}_{\underset{k}{\rightarrow}}$.
En effet, $\mathbb{1}_{\underset{k+1}{\rightarrow}} = (1, \ldots, k, k+1) >_\tau (1, \ldots, k) = \mathbb{1}_{\underset{k}{\rightarrow}}$, d'après la Proposition 3.4(ii). Dans un deuxième temps observons, puisque $(1,1) =_\tau (1,1)$, alors par la même Proposition 3.4(iv), nous avons que $(1,1) =_\tau (1,2,2)$. Par récurrence et par cette Proposition 3.4(iv), nous montrons que $\forall\ k \geq 2$, si $(1,1) =_\tau (1, \ldots, k, k)_{k+1}$, alors $(1,1) =_\tau (1, \ldots, k+1, k+1)_{k+2}$. Finalement, par Proposition 3.4(ii), on a $\forall\ k \geq 2\ :\ (1,1) =_\tau (1, \ldots, k, k)_{k+1} >_\tau (1, \ldots, k) = \mathbb{1}_{\underset{k}{\rightarrow}}$. Ainsi $\forall\ k \geq 2,\ (1,1) >_\tau \mathbb{1}_{\underset{k}{\rightarrow}}$. Nous avons donc que la suite $(\tau(\mathbb{1}_{\underset{k}{\rightarrow}}))_{\mathbb{N}_{\geq 2}}$ est croissante et bornée par $\tau(1,1) = 2$. D'après les Théorèmes 3.11 et 3.2, il vient que quelque soit $k \in \mathbb{N}_{\geq 2} : \tau(1, \ldots, k) > c_k$, où c_k est définie comme dans le Théorème 3.2. La suite des valeurs $(c_k)_{k \geq 2}$ est donc une suite minorante de la suite de valeurs $(\tau(\mathbb{1}_{\underset{k}{\rightarrow}}))$. Par la formule de Stirling, $(c_k)_{k \geq 2}$ est une suite croissante, et $\lim\limits_{k \to +\infty} c_k = 2$. En rassemblant ces observations avec $\tau(\mathbb{1}_{\underset{k}{\rightarrow}}) < 2$, nous concluons que $\lim\limits_{k \to +\infty} \tau(\mathbb{1}_{\underset{k}{\rightarrow}}) = 2$. □

Théorème 3.13. *([42, Thm 5]) Soit $k \in \mathbb{N}_{\geq 3}$. Il y a un algorithme de Branchement en temps $O^*(\overline{\gamma_k}^n)$ avec espace polynomial pour énumérer les transversaux minimaux dont le complément est un transversal d'un hypergraphe de rang k, avec $\overline{\gamma_k} = \tau((2,2, \ldots, k, k)_{2k-2}) < \tau(1, \ldots, k)$.*

La Table 3.1 précise les valeurs de γ_k et $\overline{\gamma_k}$ des Théorèmes 3.11 et 3.13 pour $3 \leq k \leq 6$.

3.2.3 Instances du problème

L'entrée du problème est un hypergraphe $\mathcal{H} = \{e_1, \ldots, e_m\}$ de rang k sur un sous-ensemble de sommets V, c'est-à-dire $\forall\ 1 \leq i \leq m,\ |e_i| \leq k$. On se propose d'énumérer les transversaux minimaux de $\mathcal{H}$.

Dans un premier temps, remarquons qu'il y a un algorithme qui s'exécute en temps polynomial pour décider si un sous-ensemble $X \subseteq V$ de sommets est un transversal de $\mathcal{H}$. En effet, le rang de $\mathcal{H}$ est borné par k, ce qui signifie que $\mathcal{H}$ est composé d'au plus $O(n^k)$ hyperarêtes. Pour une hyperarête $e \in \mathcal{H}$, et un sous-ensemble de sommets $X \subseteq V$, on peut déterminer en temps $O(n)$ si $X \cap V \neq \emptyset$. Ainsi, il y a un algorithme qui s'exécute en temps $O(n^{k+1})$ pour décider si un sous-ensemble de sommets est un transversal de $\mathcal{H}$.
Dans un deuxième temps, remarquons qu'il existe de même un algorithme qui s'exécute en temps polynomial pour décider si un sous-ensemble $X \subseteq V$ de sommets est un transversal minimal de $\mathcal{H}$. En effet, observons que l'ensemble des transversaux d'un hypergraphe représente la clôture ascendante de la famille des transversaux minimaux de cet hypergraphe. Ainsi, un transversal $X \subseteq V$ de $\mathcal{H}$ est minimal si et seulement si pour tout $x \in X$, $X_x = X - \{x\}$ n'est pas un transversal de $\mathcal{H}$. Déterminer si $X \subseteq V$ est un transversal minimal de $\mathcal{H}$ peut donc s'effectuer en déterminant dans un premier temps si X est un transversal, puis si tel est le cas, il reste à effectuer au plus $|X|$ tests de transversalité, le test de X_x pour chaque sommet $x \in X$. Avec la première observation, on remarque que l'ensemble de ces opérations peut être exécuté en temps polynomial.
Ces deux remarques sur l'existence d'algorithmes polynomiaux pour les tests de transversalité et de transversalité minimale d'un sous-ensemble $X \subseteq V$ dans un hypergraphe de rang k ont plusieurs conséquences et sont importantes pour la suite. La première conséquence est que cela rajoute un facteur polynomial $O(n^k)$ aux temps d'exécution exponentiels des algorithmes que nous présentons. Ces facteurs sont cachés dans le * de la complexité $O^*(f(n))$. Plus important cependant, l'exécution d'un nombre polynomial de fois de l'un ou l'autre de ces tests à des nœuds quelconques de l'arbre de recherche de nos algorithmes n'en augmente pas la complexité exponentielle. La deuxième conséquence est que l'algorithme trivial d'énumération des transversaux minimaux dans un hypergraphe

de rang k s'exécute en temps $O^*(2^n)$. L'objectif est alors de proposer un algorithme plus performant, ici dépendant de k.

Pour notre problème, une *instance* de sous-problème du problème originel est un triplet $(\mathcal{H}', C, D)$, où $(V[\mathcal{H}'], C, D)$ est une 3-partition de V, et où $\mathcal{H}' = \{e - D \ : \ e \in \mathcal{H}, \ e \cap C = \emptyset\}$. On note $V' = V[\mathcal{H}']$. Le sous-problème consiste à énumérer tous les transversaux minimaux de $\mathcal{H}$ qui contiennent les sommets de C, mais aucun sommet de D, ce qui revient à énumérer tous les transversaux minimaux de $\mathcal{H}'$, avec la propriété que pour tout transversal $X \subseteq V'$ de $\mathcal{H}'$, le sous-ensemble $X \cup C$ est un transversal de $\mathcal{H}$.
On mesure la taille d'un sous-problème $(\mathcal{H}', C, D)$ par le nombre de sommets contenus dans le sous-hypergraphe $\mathcal{H}'$ de $\mathcal{H}$, c'est-à-dire la taille d'un sous-problème est donnée par $|V'|$.
Les algorithmes fournis dans la Section 3.4 sont des algorithmes de branchement. Les règles de branchement transforment l'instance $(\mathcal{H}', C, D)$ d'un sous-problème en une collection d'instances de nouveaux (sous-)sous-problèmes $(\mathcal{H}'', C', D')$. Les instances $(\mathcal{H}'', C', D')$ des nouveaux (sous-)sous-problèmes vérifient les propriétés suivantes :

1. H'' est un sous-hypergraphe de H'
2. $C \subseteq C'$, $D \subseteq D'$, et $D' - D \subseteq V'$
3. $(C' - C, D' - D)$ est une 2-partition de $V' - V''$
4. $\forall\ e \in \mathcal{H}'$, si $e \cap (C' - C)$, alors $e \notin C$, sinon $e - (D' - D) \in \mathcal{H}''$

Description des Algorithmes

Pour obtenir le triplet $(\mathcal{H}'', C', D')$ correspondant à l'instance d'un nouveau sous-problème, nous utilisons les procédures `ADD` et `RMV` que nous décrivons ci-dessous. Pour créer un sous-problème, soit un sous-ensemble $X \subseteq V'$ est ajouté à C, soit un sous-ensemble $Y \subseteq V'$ est ajouté à D, ou alors les deux possibilités en même temps avec $X \cap Y = \emptyset$. L'instance $(\mathcal{H}'', C', D')$ du nouveau sous-problème est alors déterminée par l'application successive des procédures $\texttt{ADD}(X)$ et $\texttt{RMV}(Y)$ décrites ci-dessous :

$\texttt{ADD}(X)$	$\texttt{RMV}(Y)$
$\mathcal{H}'' \leftarrow \{e \in \mathcal{H}' : e \cap X = \emptyset\}$	$\mathcal{H}'' \leftarrow \{e - Y : e \in \mathcal{H}''\}$
$C' \leftarrow C \cup X$	$C' \leftarrow C$
$D' \leftarrow D$	$D' \leftarrow D \cup Y$

Si $X = \{x_1, \ldots, x_p\}$ et $Y = \{y_1, \ldots, y_q\}$, on pourra écrire respectivement $\texttt{ADD}(x_1, \ldots, x_p)$ et $\texttt{RMV}(y_1, \ldots, y_q)$.

Les règles de branchement de nos algorithmes sont décrites sous formes d'un tableau, où à chaque ligne correspond un nouveau sous-problème, la ligne décrivant alors l'instance de ce nouveau sous-problème. A chaque case du tableau correspond un sommet de V', qui pour le nouveau sous-problème, est soit ajouté à l'ensemble C si la case vaut 1, soit ajouté à l'ensemble D si la case vaut 0, ou alors ignoré si la case est vide. Lorsque la règle de branchement est suffisamment simple, et pour en améliorer la compréhension, ce tableau pourra se présenter sous deux colonnes, la deuxième colonne indiquant l'utilisation des procédures `ADD` et `RMV` pour obtenir le nouveau sous-problème. Nous illustrons cette notation par les Tables 3.4 et 3.5 correspondantes respectivement aux Algorithmes des Théorèmes 4 et 5 de notre publication de 2014 [42].

$e = \{x_1, \ldots, x_p\}$							*procedures*
x_1	x_2	$\ldots$	x_i	$\ldots$	x_{q-1}	x_q	
0	1						ADD(x_2), RMV(x_1)
0	0	0	1				ADD(x_i), RMV$(x_1, \ldots, x_{i-1})$
0	0	$\ldots$	0	0	1		ADD(x_{q-1}), RMV$(x_1, \ldots, x_{q-2})$
0	0	$\ldots$	0	$\ldots$	0	1	ADD(x_q), RMV$(x_1, \ldots, x_{q-1})$
1							ADD(x_1)

Table 3.4 – Sous-problèmes construits par l'unique règle de branchement de l'Algorithme du Théorème 3.11 pour le problème k-Hitting Set [42, Thm 4].

$e = \{x_1, \ldots, x_p\}$							*procedures*
x_1	x_2	$\ldots$	x_i	$\ldots$	x_{q-1}	x_q	
0	1						ADD(x_2), RMV(x_1)
0	0	0	1				ADD(x_i), RMV$(x_1, \ldots, x_{i-1})$
0	0	$\ldots$	0	0	1		ADD(x_{q-1}), RMV$(x_1, \ldots, x_{q-2})$
0	0	$\ldots$	0	$\ldots$	0	1	ADD(x_q), RMV$(x_1, \ldots, x_{q-1})$
1	0						ADD(x_1), RMV(x_2)
1	1	1	0				ADD$(x_1, \ldots, x_{i-1})$, RMV(x_i)
1	1	$\ldots$	1	1	0		ADD$(x_1, \ldots, x_{q-2})$, RMV(x_{q-1})
1	1	$\ldots$	1	$\ldots$	1	0	ADD$(x_1, \ldots, x_{q-1})$, RMV(x_q)

Table 3.5 – Sous-problèmes construits par par la règle de branchement R5.5 de l'Algorithme du Théorème 3.13 pour le problème co$k$-Hitting Set [42, Thm 5].

3.3 Algorithme pour 3-Hitting Set

Dans cette Section, nous présentons l'Algorithme 3.1 pour l'énumération des transversaux minimaux d'un hypergraphe de rang $k = 3$. Cet algorithme utilise les techniques de *Branchement* et de *Mesurer et Conquérir*. Nous référons à la Section 1.5 pour une introduction à ces techniques. Cette section est consacrée à la construction, la preuve de validité et l'analyse de l'Algorithme 3.1. En conséquence du Théorème 3.17, nous améliorons les valeurs β, γ_3, $\overline{\gamma_3}$, λ_3, et $\overline{\lambda_3}$ de la Table 3.1.
Nous présentons préalablement le Lemme 3.14, dont la preuve est similaire à celle du Lemme 3.21.

Lemme 3.14. *Soit $\mathcal{H}$ un hypergraphe, et soit $C, D \subseteq V$, avec $C \cap D = \emptyset$.*
Soit $\mathcal{H}' = \{e - D \ : \ e \in \mathcal{H}, \ e \cap C = \emptyset\}$. Soit $X \subseteq V$, et avec V' les sommets de $\mathcal{H}'$, on a :

$$X \in Tr(\mathcal{H}) \Rightarrow X - \overline{V'} \in Tr(\mathcal{H}')$$

Afin de prouver le Théorème 3.17, nous procédons en deux étapes. Tout d'abord, nous montrons par le Lemme 3.15 que notre algorithme est valide. Puis, après avoir introduit la mesure que nous utilisons, nous analyserons son temps d'exécution dans le Lemme 3.16.

Lemme 3.15. *Les règles de réduction et de branchement de l'Algorithme 3.1 sont correctes.*

Démonstration. Soit $(\mathcal{H}', C, D)$ une instance de sous-problème. On note C^* un transerval minimal de $\mathcal{H}$ contenant C avec $C^* \cap D = \emptyset$.
R0.0. Par construction, si $\mathcal{H}' = \emptyset$ alors C est un transversal de $\mathcal{H}$.
R0.1. Par construction, si $e \in \mathcal{H}'$, alors il existe $e^+ \in \mathcal{H}$, $e \subseteq e^+$, telle que $C \cap e^+ = \emptyset$. Or $e = \emptyset$ implique quelque soit $Y \subseteq V' : (Y + C) \cap e^+ = \emptyset$. Par conséquent, C ne peut être contenu dans un transversal de $\mathcal{H}$, et donc cette instance n'admet pas de solution.

Algorithme 3-**HS**$(\mathcal{H}', C, D)$

Règles de Réduction.

R0.0. Si $\mathcal{H}' = \emptyset$: `STOP`, C est un transversal de $\mathcal{H}$, et $C \in Tr(\mathcal{H})$ si et seulement si C est minimal.

R0.1. S'il existe $e \in \mathcal{H}'$ avec $e = \emptyset$: `STOP`, cette instance n'admet pas de solution.

R1.0. S'il existe $v \in V'$ de degré $d_{\mathcal{H}'}(v) = 0$: $\texttt{RMV}(v)$.

R1.1. S'ils existent $e_1, e_2 \in E'$ avec $e_1 \subseteq e_2$ et $|e_2| = 3$: $\mathcal{H}'' \leftarrow \mathcal{H}' - e_2$.

R1.2. S'il existe $e \in \mathcal{H}'$ avec $e = \{v\}$: $\texttt{ADD}(v)$.

Règles de Branchement. Chaque règle de branchement de l'algorithme crée différents sous-problèmes. Pour une règle de branchement, nous écrivons $\langle \mathcal{O}_1 \mid\mid \ldots \mid\mid O_p \rangle$ pour exprimer le fait que l'algorithme crée p instances de sous-problèmes. Le i-ème sous-problème est obtenu en appliquant sur l'instance $(\mathcal{H}', C, D)$ les opérations décrites dans l'ensemble $\mathcal{O}_i$.

B1. S'il existe un sommet $v \in V'$ de degré $d_{\mathcal{H}'}(v) = 1$, alors nous distinguons trois cas.

(a) Si v est contenu dans une hyperarête $e = \{v, u\} \in \mathcal{H}'$ de taille 2 :
$\langle \{\texttt{ADD}(v), \texttt{RMV}(u)\} \mid\mid \{\texttt{RMV}(v), \texttt{ADD}(u)\} \rangle$

(b) Si v est contenu dans une hyperarête $e = \{v, u, w\} \in \mathcal{H}'$ de taille 3, avec $d_{\mathcal{H}'}(u) = d_{\mathcal{H}'}(w) = 1$:
$\langle \{\texttt{ADD}(v), \texttt{RMV}(\{u, w\})\} \mid\mid \{\texttt{RMV}(v), \texttt{ADD}(u), \texttt{RMV}(w)\} \mid\mid \{\texttt{RMV}(\{v, u\}), \texttt{ADD}(w)\} \rangle$

(c) Si v est contenu dans une hyperarête $e = \{v, u, w\} \in \mathcal{H}'$ de taille 3, avec $d_{\mathcal{H}'}(u) \geq 1$:
$\langle \{\texttt{ADD}(v), \texttt{RMV}(\{u, w\})\} \mid\mid \texttt{RMV}(v) \rangle$

B2. Si $\mathcal{H}'$ contient au moins une hyperarête de taille 2.
Soit $v \in V'$ un sommet de degré $d_{\mathcal{H}'}(v)$ maximum parmi les sommets contenus dans le plus grand nombre $q \in \mathbb{N}^*$ d'hyperarêtes distinctes de taille 2 dans $\mathcal{H}'$. Soient $\{v, u_1\}, \ldots, \{v, u_q\}$ les q hyperarêtes de $\mathcal{H}'$ contenant v, alors :
$\langle \texttt{ADD}(v) \mid\mid \{\texttt{RMV}(v), \texttt{ADD}(u_1, \ldots, u_q)\} \rangle$

B3. Si $\mathcal{H}'$ ne contient que des hyperarêtes de taille 3. Soit $v \in V'$ un sommet de degré maximum dans $\mathcal{H}'$. Nous distinguons alors trois cas.

(a) Si $d_{\mathcal{H}'}(v) \geq 3$:
$\langle \texttt{ADD}(v) \mid\mid \texttt{RMV}(v) \rangle$

(b) Si $d_{\mathcal{H}'}(v) = 2$, et s'il existe $u \in V'$ avec $\mathcal{H}'_v = \mathcal{H}'_u$:
$\langle \{\texttt{ADD}(v), \texttt{RMV}(u)\} \mid\mid \texttt{RMV}(v) \rangle$

(c) Sinon si $d_{\mathcal{H}'}(v) = 2$. En notant $e_1 = \{v, u_1, w_1\}$ et $e_2 = \{v, u_2, w_2\}$ les deux hyperarêtes de $\mathcal{H}'$ contenant v :
$\langle \{\texttt{ADD}(v, u_1), \texttt{RMV}(u_2, w_2)\} \mid\mid \{\texttt{ADD}(v), \texttt{RMV}(u_1)\} \mid\mid \texttt{RMV}(v) \rangle$

Algorithme 3.1 – Algorithme de Branchement du Théorème 3.17 pour la version énumération du problème 3-Hitting Set. Afin d'énumérer les transversaux minimaux d'un hypergraphe $\mathcal{H}$ de rang $k = 3$, nous éxecutons cet algorithme sur l'instance initiale $(\mathcal{H}, \emptyset, \emptyset)$.

R1.0. Si $C \cup \{v\}$ est contenu dans un transversal minimal C^* de $\mathcal{H}$, alors $C^* - \{v\}$ est un transversal de $\mathcal{H}$, ce qui contredit que C^* est minimal. Ainsi, v n'appartient à aucun transversal minimal de $\mathcal{H}$ qui contient C.

R1.1. Observons que X^* est un transversal minimal de $\mathcal{H}''$ si et seulement si X^* est un transversal minimal de $\mathcal{H}'$. En effet, si X^* est un transversal de $\mathcal{H}''$, alors $X^* \cap e_1 \neq \emptyset$. Avec $e_1 \subseteq e_2$, il

vient $X^* \cap e_2 \neq \emptyset$, et nous avons X^* est un transversal de $\mathcal{H}'$. Supposons que X^* soit minimal pour $\mathcal{H}''$, et supposons par contradiction que X^* ne soit pas minimal pour $\mathcal{H}'$. On pose $Y^* \subset X^*$ transversal de $\mathcal{H}'$. Alors Y^* est un transversal de $\mathcal{H}''$, car $\mathcal{H}''$ est un sous-hypergraphe de $\mathcal{H}'$. En conséquence, X^* n'est pas un transversal minimal de $\mathcal{H}''$, une contradiction. Réciproquement, si X^* est un transversal minimal de $\mathcal{H}'$, alors X^* est en particulier un transversal de $\mathcal{H}''$, puisque $\mathcal{H}''$ est un sous-hypergraphe de $\mathcal{H}'$. Supposons par contradiction que X^* ne soit pas minimal pour $\mathcal{H}''$. Supposons alors que $Y^* \subset X^*$ soit un transversal minimal de $\mathcal{H}''$. Par les observations précédentes, Y^* est un transversal de $\mathcal{H}''$ a pour conséquence que Y^* est un transversal de $\mathcal{H}'$. Et donc X^* n'est pas un transversal minimal de $\mathcal{H}'$, une contradiction.
R1.2. Si $v \notin C^*$, alors C^* n'est pas un transversal de $\mathcal{H}$.
B1.a. Dans la première branche, il suffit d'observer que sélectionner u et v implique que $C^* - \{u\}$ est un transversal de $\mathcal{H}$, et donc C^* n'est pas minimal, une contradiction. Dans la deuxième branche, sélectionner u après avoir écarté v est justifié par la règle `R1.2`.
B1.b. Dans la première branche, écarter u et w après avoir sélectionné v est justifié par la même observation que la règle `B1.a`. Les opérations réalisées dans les autres branches sont justifiées symétriquement.
B1.c. Les opérations de la première branche sont justifiées de manière identique à la règle `B1.a`.
B2. Les opérations de la deuxième branche sont justifiées par la règle `R1.2`.
B3.b. Les opérations de la première branche sont justifiées de manière identique à la règle `B1.a`.
B3.c. Les opérations de la première branche sont justifiées de manière identique à la règle `B1.a`. Les opérations effectuées dans les deux autres branches permettent d'évaluer tous les autres cas possibles. □

Nous introduisons à présent la mesure que nous utilisons pour l'analyse de notre algorithme. Pour un problème, une *mesure* μ de l'entrée est une fonction qui attribue à chaque instance du problème un réel positif. Pour une mesure μ définie, on appelle le *poids* d'une instance l'évaluation de la mesure μ appliquée à l'instance considérée. En conséquence, le poids d'une instance de sous-problème se doit toujours d'être inférieur au poids de l'instance originelle.
Attribuer des poids distincts aux sommets d'un graphe en fonction de leur degré semble être devenu une technique standard pour définir une mesure d'un algorithme de *Branchement* [90]. Pour notre problème, la version énumération du problème 3-Hitting Set, nous évaluons la taille de l'hypergraphe $\mathcal{H}$ de l'entrée du problème en tenant compte des degrés (ou encore des fréquences) des sommets de $\mathcal{H}$, mais aussi en tenant compte du nombre de ses hyperarêtes de taille ≤ 2, appelées 2-*hyperarêtes*.

Soit $\mathcal{H}$ un hypergraphe de rang $k = 3$. On note n_i le nombre de sommets de degré $i \in \mathbb{N}$. On note m_i le nombre d'hyperarêtes de taille $i \in \{0, \ldots, 3\}$, et on note $m_{\leq i} = \sum_{j=0}^{i} m_j$. Formellement, la mesure utilisée par notre algorithme est définie par :

$$\mu(\mathcal{H}) = \Psi(m_{\leq 2}) + \sum_{i=0}^{\infty} \omega_i n_i \ ,$$

où les poids ω_i sont des réels positifs. Le fil directeur de notre algorithme est d'obtenir en permanence une instance avec le moins possible d'hyperarêtes de taille $k = 3$. Pour cela, nous souhaitons que le poids d'une instance diminue à mesure que le nombre de 2-hyperarêtes augmente. En conséquence, la fonction $\Psi : \mathbb{N} \to \mathbb{R}_{\geq 0}$ est une fonction positive décroissante.
Pour un sommet $v \in V$, on pourra noter $\omega(v) = \omega_{d_{\mathcal{H}}(v)}$ le poids de v dans la mesure μ.

Afin de faciliter l'analyse du temps d'exécution de notre algorithme, nous définissons :

$$\omega_5 = \omega_i \qquad \text{pour tout } i \geq 6, \tag{3.1}$$
$$\Psi(i) = 0 \qquad \text{pour tout } i \geq 6, \tag{3.2}$$
$$\Delta\omega_i = \omega_i - \omega_{i-1} \qquad \text{pour tout } i \geq 1, \text{ et} \tag{3.3}$$
$$\Delta\Psi(i) = \Psi(i) - \Psi(i-1) \qquad \text{pour tout } i \geq 1. \tag{3.4}$$

et nous ajoutons les contraintes suivantes à notre mesure :

$$\omega_1 \geq \Psi(0) \tag{3.5}$$
$$0 \leq \Delta\omega_{i+1} \leq \Delta\omega_i \qquad \text{pour tout } i \geq 1, \text{ et} \tag{3.6}$$
$$0 \geq \Delta\Psi(i+1) \geq \Delta\Psi(i) \qquad \text{pour tout } i \geq 1. \tag{3.7}$$
$$\Psi(0) - \Psi(i) \leq \omega_i \qquad \text{pour tout } 1 \leq i \leq 6. \tag{3.8}$$

Par les équations (3.6) et (3.7), nous pouvons observer :

$$\omega_i - \omega_{i-k} \geq k \cdot \Delta\omega_i \qquad \text{pour tout } 0 \leq k \leq i. \tag{3.9}$$
$$\Psi(i+j+k) - \Psi(i+j) \geq \Psi(i+k) - \Psi(i) \qquad \text{pour tout } i, j, k \geq 0. \tag{3.10}$$

Pour un algorithme standard qui utilise la technique *Mesurer et Conquérir*, les contraintes et les équations que nous venons d'introduire ont plusieurs raisons d'exister. Dans un premier temps, elles garantissent que le poids d'une instance est toujours défini et positif. Dans un deuxième temps, certaines contraintes ont pour objectif de faciliter l'analyse du temps d'exécution. Bien entendu, toutes sont introduites avec pour objectif d'obtenir le meilleur temps d'exécution possible pour notre algorithme.

Lemme 3.16. *L'Algorithme 3.1 s'exécute en temps* $O^*(1.6755^n)$.

Démonstration. Nous exprimons les vecteurs et nombres de branchement de notre algorithme en fonction de la mesure μ utilisée. Pour une règle de branchement qui crée p instances de sous-problèmes, on note $\tau(\eta_1, \ldots, \eta_p)$ son nombre de branchement, où η_i représente la différence de poids entre l'instance originelle et la i-ème instance de sous-problème. Pour une règle de réduction, on note simplement $\eta_1 = \eta$. Dans notre mesure μ, le poids d'une instance de sous-problème sera toujours inférieur au poids de l'instance originelle, autrement dit pour tout $1 \leq i \leq p : \eta_i \geq 0$.
Pour toute i-ème instance de sous-problème d'une règle de branchement ou de réduction, on note $\partial\Psi$ la contribution dans η_i de la variation dans la mesure μ du nombre de 2-hyperarêtes. En notant $m^i_{\leq 2}$ le nombre de 2-hyperarêtes présentes dans la i-ème instance de sous-problème, ou $m'_{\leq 2}$ si $p = 1$, on a alors : $\partial\Psi = \Psi(m_{\leq 2}) - \Psi(m^i_{\leq 2})$. Si $m^i_{\leq 2} \leq m_{\leq 2}$, nous avons toujours $\partial\Psi \leq 0$, puisque Ψ est décroissante. De même, pour tout $0 \leq \varepsilon \leq m^i_{\leq 2}$, observons que $\partial\Psi \geq \Psi(m_{\leq 2}) - \Psi(m^i_{\leq 2} - \varepsilon)$. Pour une instance de sous-problème, et pour un sommet $x \in V$, on note $\partial\omega(x)$ la différence entre le poids $\omega(x)$ de x dans l'instance originelle, et son poids $\omega(x) - \partial\omega(x)$ dans l'instance de sous-problème considérée. On note de plus $d_{\mathcal{H}',\leq 2}(x)$ le nombre de 2-hyperarêtes de $\mathcal{H}'$ qui contiennent x.
Nous montrons dans un premier temps que pour toute règle de réduction : $\eta \geq 0$.
R1.0. $\eta = \omega(v) = \omega_0$. Or par définition, $\forall\ i \geq 0 : \omega_i \in \mathbb{R}_+$. En particulier $\omega_0 \geq 0$, et donc $\eta \geq 0$.
R1.1. Observons que puisque $|e_2| = 3$, le nombre de 2-hyperarêtes ne diminue pas en créant l'instance $(\mathcal{H}'', C, D)$. Nous avons donc $\eta = \sum_{x \in e_2} \partial\omega(x) = \sum_{x \in e_2} \Delta\omega(x)$. Par la contrainte (3.6), nous avons $\forall\ x \in e_2 : \Delta\omega(x) \geq 0$, et il vient $\eta \geq 0$.

R1.2. Le poids de l'instance diminue de $\eta = \omega(v)+\sum_{x\in N_{\mathcal{H}'}(v)} \partial\omega(x)+\partial\Psi$. Par le même raisonnement que pour la règle `R1.1`, et par la contrainte (3.6), nous avons $\sum_{x\in N_{\mathcal{H}'}(v)} \partial\omega(x) \geq 0$. Ainsi :

$$\begin{aligned}
\eta &\geq \omega(v) + \partial\Psi \\
&\geq \omega(v) + \Psi(m_{\leq 2}) - \Psi(m_{\leq 2} - d_{\mathcal{H}'}(v)) && \text{car } m'_{\leq 2} \geq m_{\leq 2} - d_{\mathcal{H}'}(v), \\
&\geq \omega(v) + \Psi(d_{\mathcal{H}'}(v)) - \Psi(0) && \text{par l'équation (3.10)}, \\
&\geq \omega(v) - \omega(v) && \text{par la contrainte (3.8)}.
\end{aligned}$$

Nous exprimons à présent les vecteurs et nombres de branchement des règles de branchement notre algorithme. Pour toutes ces règles, on observera que si η est de la forme $\eta \geq \omega_i - \Psi(j) + \varepsilon$, pour $i \in \mathbb{N}^*$, $j \in \mathbb{N}$ et $\varepsilon \in \mathbb{R}_+$, alors par construction de notre mesure μ et par la contrainte (3.5) : $\eta \geq \omega_1 - \Psi(0) \geq 0$. Cette observation permet de valider que le poids de chaque instance de sous-problème dans nos règles de branchement se retrouve toujours réduit par rapport au poids de l'instance originelle.

B1.a. Pour ce cas : $\omega(v) = \omega_1$. Dans la première branche, une unique 2-hyperarête est supprimée dans la nouvelle instance puisque $d_{\mathcal{H}'}(v) = 1$. Ainsi : $\eta_1 = \omega_1 + \omega(u) + \Delta\Psi(m_{\leq 2})$.
Dans la deuxième branche, pour les mêmes raisons que pour la règle `R1.2`, nous négligeons $\sum_{x\in N(u)-\{v\}} \partial\omega(x) \geq 0$ dans η_2. Dans cette branche, le nombre de 2-hyperarêtes restantes dans la nouvelle instance est au moins $m^2_{\leq 2} \geq \max(m_{\leq 2} - d_{\mathcal{H}'}(u), 0)$. Ainsi : $\eta_2 \geq \omega(v) + \omega(u) + \partial\Psi \geq \omega_1 + \omega(u) + \Psi(m_{\leq 2}) - \Psi(\max(m_{\leq 2} - d_{\mathcal{H}'}(u), 0))$.
Notre analyse se porte donc sur les valeurs $1 \leq d_{\mathcal{H}'}(u) \leq 6$ et $1 \leq m_{\leq 2} \leq 6$.

B1.b. Avec $d_{\mathcal{H}'}(v) = d_{\mathcal{H}'}(u) = d_{\mathcal{H}'}(w) = 1$, et en observant que le nombre de 2-hyperarêtes est stable dans chaque branche, on a : $\eta_1 = \eta_2 = \eta_3 = \omega(v) + \omega(u) + \omega(w) = 3\omega_1$.

B1.c. $\eta_1 = \omega(v) + \omega(u) + \omega(w) \geq 2\omega_1 + \omega_2$, et $\eta_2 = \omega(v) - \max_{i\geq 1} \Delta\Psi(i) \geq \omega_1$.

B2. Dans la première branche, nous avons : $\eta_1 = \omega(v) + \sum_{i=1}^{q} \partial\omega(u_i) + \partial\Psi$, et $\omega(v) \geq \omega_q$. De plus, nous avons $m^1_{\leq 2} = m_{\leq 2} - d_{\mathcal{H}',\leq 2}(x) = m_{\leq 2} - q$, et pour tout $1 \leq i \leq q$: $\partial\omega(u_i) = \Delta\omega(u_i)$. En conséquence, nous déduisons : $\eta_1 \geq \omega(v) + \sum_{i=1}^{q} \Delta\omega(u_i) + \Psi(m_{\leq 2}) - \Psi(m_{\leq 2} - q)$.
Dans la deuxième branche, nous négligeons dans η_2 la quantité $\sum_{x\in N(\{u_1,\ldots,u_p\})-\{v\}} \partial\omega(x) \geq 0$. Pour cette branche, la quantité $\partial\Psi$ peut être positive.
Finalement, notons que $m_{\leq 2} \geq q$, et observons que puisque la règle `B1` ne s'est pas appliquée, pour tout $x \in V$: $d_{\mathcal{H}'}(x) \geq 2$, ou encore $\omega(x) \geq 2$. Pour simplifier, nous distinguons trois cas.

- ◇ Si $q = 1$. Alors d'après le choix de v : $d_{\mathcal{H}'}(v) \geq d_{\mathcal{H}'}(u_1) \geq 2$, et donc : $\omega(v) \geq \omega(u_1) \geq \omega_2$. Dans la première branche, nous avons $\eta_1 \geq \omega(v) + \Delta\omega(u_1) + \partial\Psi \geq \omega(u_1) + \Delta\omega(u_1) + \Delta\Psi(m_{\leq 2})$.
  Dans la deuxième branche, nous supprimons une 2-hyperarête en sélectionnant u_1. Cependant en écartant v, nous en créons $d_{\mathcal{H}'}(v) - 1$ nouvelles qui ne peuvent se retrouver contenues dans d'autres 2-hyperarêtes déjà existantes, sinon la règle `R1.1` se serait appliquée. Ainsi donc, il se crée un nombre $d_{\mathcal{H}'}(v) - 2 \geq d_{\mathcal{H}'}(u_1) - 2 \geq 0$ de 2-hyperarêtes, et la quantité $\partial\Psi$ est postive. Ψ est une fonction décroissante, alors si $a \geq b$, on a $\Psi(a) \leq \Psi(b)$, et encore $-\Psi(a) \geq -\Psi(b)$. En posant $a = m_{\leq 2} + d_{\mathcal{H}'}(v) - 2$ et $b = m_{\leq 2} + d_{\mathcal{H}'}(u_1) - 2$, on a donc $\eta_2 \geq \omega(v) + \omega(u_1) + \partial\Psi \geq 2\omega(u_1) + \Psi(m_{\leq 2}) - \Psi(m_{\leq 2} + d_{\mathcal{H}'}(u_1) - 2)$.
  Notre analyse se porte alors sur les valeurs $2 \leq d_{\mathcal{H}'}(u_1) \leq 6$ et $1 \leq m_{\leq 2} \leq 6$.
- ◇ Si $q = 2$. Dans la première branche, et sans observation supplémentaire, nous avons $\eta_1 \geq \omega(v) + \Delta\omega(u_1) + \Delta\omega(u_2) + \Psi(m_{\leq 2}) - \Psi(m_{\leq 2} - 2)$.
  Dans la deuxième branche, nous observons qu'en sélectionnant u_1 et u_2, le nombre de 2-hyperarêtes ne décroît que d'au plus $\min(m_{\leq 2}, d_{\mathcal{H}',m_{\leq 2}}(u_1) + d_{\mathcal{H}',m_{\leq 2}}(u_2)) \leq \min(m_{\leq 2}, 2q) \leq \min(m_{\leq 2}, 4)$. De plus, pour les mêmes raisons que dans le premier cas, il se crée un nombre $d_{\mathcal{H}'}(v) - q$ de 2-hyperarêtes en écartant v. Nous avons donc $m^2_{\leq 2} \geq m_{\leq 2} - \min(m_{\leq 2}, 4) + d_{\mathcal{H}'}(v) - 2$, ce qui s'écrit encore $m^2_{\leq 2} \geq \max(m_{\leq 2} - 4, 0) + d_{\mathcal{H}'}(v) - 2$. Ainsi, nous avons $\eta_2 \geq \omega(v) + \omega(u_1) +$

$\omega(u_2) + \Psi(m_{\leq 2}) - \Psi(\max(m_{\leq 2} - 4, 0) + d_{\mathcal{H}'}(v) - 2)$.
Notre analyse se porte en conséquence sur les valeurs $2 \leq d_{\mathcal{H}'}(v), d_{\mathcal{H}'}(u_1), d_{\mathcal{H}'}(u_2) \leq 6$ et $2 \leq m_{\leq 2} \leq 6$.

◇ Si $q \geq 3$. Dans la première branche, et sans observation supplémentaire, nous avons $\eta_1 \geq \omega(v) + \sum_{i=1}^{p} \Delta\omega(u_i) + \Psi(m_{\leq 2}) - \Psi(m_{\leq 2} - q) \geq \omega_q + \sum_{i=1}^{3} \Delta\omega(u_i) + \Psi(m_{\leq 2}) - \Psi(m_{\leq 2} - q)$.
Dans la deuxième branche, nous négligeons $m^2_{\leq 2} \geq 0$, et nous observons $\sum_{i=1}^{p} \omega(u_i) \geq \sum_{i=1}^{3} \omega(u_i) + (q-3)\omega_2$ par l'équation (3.6). En conséquence, nous avons $\eta_2 \geq \omega_q + \sum_{i=1}^{3} \omega(u_i) + (q-3)\omega_2 + \Psi(m_{\leq 2}) - \Psi(0)$.
Notre analyse se porte alors sur les valeurs $3 \leq q \leq 6$ et $2 \leq d_{\mathcal{H}'}(u_i) \leq 6$ pour $1 \leq i \leq 3$.

B3.a. Dans la première branche, puisque v est de degré maximum et par la contrainte (3.6), nous avons $\sum_{x \in N_{\mathcal{H}'}(v)} \partial\omega(x) \geq 2 \cdot d_{\mathcal{H}'}(v) \cdot \Delta\omega(v)$. Aucune 2-hyperarête n'étant créée, nous avons $\eta_1 \geq \omega(v) + 2 \cdot d_{\mathcal{H}'}(v) \cdot \Delta\omega(v)$.
Dans la deuxième branche, le nombre de 2-hyparête augmente de 0 à $d_{\mathcal{H}'}(v)$, nous avons donc $\eta_2 \geq \omega(v) + \Psi(0) - \Psi(d_{\mathcal{H}'}(v))$.
Notre analyse porte alors sur les valeurs $3 \geq d_{\mathcal{H}'}(v) \geq 6$.

B3.b. Si $d_{\mathcal{H}'}(v) = 2$, et s'il existe $u \in V'$ avec $\mathcal{H}'_v = \mathcal{H}'_u$. Observons que pour ce cas comme pour le cas `B3.c`, nous avons $d_{\mathcal{H}'}(u) = 2$ pour tout $u \in V'$. Dans la première branche, nous avons alors : $\eta_1 = \omega(v) + \omega(u) + \sum_{x \in N_{\mathcal{H}'}(v) \cap N_{\mathcal{H}'}(u)} \partial\omega(x) = 2\omega_2 + 2\Delta\omega_2$. Et dans la deuxième branche, nous avons : $\eta_2 = \omega(v) + \Psi(0) - \Psi(2)$.

B3.c. Si $d_{\mathcal{H}'}(v) = 2$, et s'il existe $u \in V'$ avec $\mathcal{H}'_v = \mathcal{H}'_u$. Dans la première branche, le degré de w_1 diminue de 1. Étant donné que tous les sommets sont de degré 2, et qu'aucune paire de sommets ne peut partager plus qu'une hyperarête, écarter u_2 et w_2 après avoir sélectionner v et u_1 crée au moins une 2-hyperarêtes. Ainsi : $\eta_1 \geq 4\omega_2 + \Delta\omega_2 - \Delta\Psi(1)$.
Dans la deuxième branche, les degrés de w_1, u_2, et w_2 diminuent de 1 et la taille de l'autre hyperarête contenant u_1 diminue de 1. Nous avnos donc $\eta_2 = 2\omega_2 + 3\Delta\omega_2 - \Delta\Psi(1)$. Dans la troisième branche, nous avons $\eta_3 = \omega_2 + \Psi(0) - \Psi(2)$. □

Nous sommes désormais prêts à présenter le Théorème 3.17 suivant.

Théorème 3.17. *L'Algorithme 3.1 énumère les transversaux minimaux d'un hypergraphe de rang $k = 3$ en temps $O^*({\gamma_3}^n)$ avec espace polynomial, où $\gamma_3 = 1.6755$.*

Démonstration. Le tableau suivant donne les valeurs des poids utilisés par notre analyse, et permettent de vérifier le temps d'exécution de notre l'Algorithme 3.1 selon la mesure utilisée.

i	ω_i	$\Psi(i)$
0	0	0.566096928
1	0.580392137	0.436314617
2	0.699175718	0.306532603
3	0.730706814	0.211986294
4	0.742114220	0.119795899
5	0.744541491	0.035202514
6	0.744541491	0

Nous rappelons que puisque pour tout hypergraphe $\mathcal{H}$, notre mesure vérifie $\mu(\mathcal{H}) \leq \omega_5 \cdot n + \varepsilon$, avec $0 \leq \varepsilon = \Psi(m_{\leq 2}) \leq \Psi(0)$, notre algorithme s'exécute en temps $O^*(2^{\mu(\mathcal{H})}) = O^*(1.6755^n)$. □

3.4 Algorithmes pour k-Hitting Set

Dans cette Section, nous présentons des algorithmes de *Branchement* pour les versions énumération du problème k-Hitting Set. Cette Section est décrite et developpée dans le contexte défini par la Sous-Section 3.2.3. Nous invitons le lecteur à se référer à la Section 3.2 pour une introduction aux outils ainsi qu'aux notations utilisées dans cette Section.

Nous décrirons dans la Sous-Section 3.4.2 nos algorithmes de branchement pour ces problèmes. Tout d'abord, dans la Sous-Section 3.4.1, nous décrivons les règles de branchement utilisées par nos algorithmes. Ces règles sont au nombre de deux, et sont nommées LME$^{[1]}$, EMM$^{[2]}$. La règle EMM$^{[2]}$ peut être vue comme une généralisation à deux hyperarêtes de la règle de branchement `R5.4` utilisée par l'Algorithme présenté au Théorème 3.11, publié en 2014 [42, Thm 4] et dont nous avons donné une description dans la Table 3.4.

3.4.1 Règles de Branchement

Soit $(\mathcal{H}', C, D)$ une instance de sous-problème du problème k-Hitting Set, et soit $x \in V'$.

LME$^{[1]}$

Soit $\mathcal{H}_x \subseteq \mathcal{H}'$ tel que $\mathcal{H}_x = \{e\}$. On pose $e = \{x_1, \ldots, x_q\}$, avec $2 \leq q \leq k$ et $x = x_1$.
La règle LME$^{[1]}$ créé alors les sous-problèmes définis dans le tableau 3.6. Le Lemme 3.18 analyse le vecteur de branchement $L^{[1]}_{\underset{k}{\rightarrow}}$ de la règle LME$^{[1]}$.

$e = \{x_1, \ldots, x_q\}$					*procedures*
x_1	x_2	$\ldots$	x_{q-1}	x_q	
1	0	$\ldots$	0	0	`ADD`(x_1), `RMV`$(x_2, \ldots, x_q)$
0					`RMV`(x_1)

Table 3.6 – Sous-problèmes construits par la règle de branchement LME$^{[1]}$ définie dans la sous-section 3.4.1 pour le problème k-Hitting Set.

Lemme 3.18. $\forall\ k \geq 2\ :\ L^{[1]}_{\underset{k}{\rightarrow}} =_\tau (1, 2)$.

Démonstration. Par observation du tableau 3.6 décrivant les sous-problèmes de la règle LME$^{[1]}$, il vient aisément que pour $q \in \mathbb{N}_{\geq 2}$, le vecteur de branchement de cette règle est $(1, q)$. Par la Proposition 3.4 (iii), il vient que pour tout $q' \geq q$: $(1, q) <_\tau (1, q')$. Le pire des cas pour la règle LME$^{[1]}$ est donc atteint, pour tout $k \in \mathbb{N}_{\geq 2}$, en $q = 2$. □

EMM$^{[2]}$

Soit $\mathcal{H}_x \subseteq \mathcal{H}'$ tel que $e_1, e_2 \in \mathcal{H}_x$, avec $|e_1| = q$, $|e_2| = q'$. On pose $e_1 \cap e_2 = \{x_1, \ldots, x_\ell\}$ avec $1 \leq \ell \leq k-1$ et $x = x_1$. Finalement, on pose $e_1 = \{x_1, \ldots, x_\ell, y_1, \ldots, y_q\}$ et $e_2 = \{x_1, \ldots, x_\ell, z_1, \ldots, z_{q'}\}$ avec $1 \leq q' \leq q$, et $q + \ell \leq k$.
La règle EMM$^{[2]}$ créé alors les sous-problèmes définis dans le tableau 3.7. Le Lemme 3.19 analyse le vecteur de branchement $M^{[2]}_{\underset{k}{\rightarrow}}$ de la règle EMM$^{[2]}$.

Lemme 3.19. $\forall\ k \geq 2\ :\ \tau(M^{[2]}_{\underset{k}{\rightarrow}}) = \max_{1 \leq \ell \leq k-1} \left\{ \mathbb{1}_{\underset{\ell}{\rightarrow}} \bigoplus \ell \bigodot \left[\mathbb{1}_{\underset{k-\ell}{\rightarrow}} \bigotimes \mathbb{1}_{\underset{k-\ell}{\rightarrow}} \right] \right\}$.

e_1										e_2									
x_1	$\dots$	x_i	$\dots$	x_ℓ	y_1	$\dots$	y_j	$\dots$	y_q	x_1	$\dots$	x_i	$\dots$	x_ℓ	z_1	$\dots$	$z_{j'}$	$\dots$	$z_{q'}$
0	0	1								0	0	1							
0	$\dots$	0	0	1						0	$\dots$	0	0	1					
0	$\dots$	0	$\dots$	0	1					0	$\dots$	0	$\dots$	0	1				
0	$\dots$	0	$\dots$	0	1					0	$\dots$	0	$\dots$	0	0	0	1		
0	$\dots$	0	$\dots$	0	1					0	$\dots$	0	$\dots$	0	0	$\dots$	0	0	1
0	$\dots$	0	$\dots$	0	0	0	1			0	$\dots$	0	$\dots$	0	1				
0	$\dots$	0	$\dots$	0	0	0	1			0	$\dots$	0	$\dots$	0	0	0	1		
0	$\dots$	0	$\dots$	0	0	0	1			0	$\dots$	0	$\dots$	0	0	$\dots$	0	0	1
0	$\dots$	0	$\dots$	0	0	$\dots$	0	0	1	0	$\dots$	0	$\dots$	0	1				
0	$\dots$	0	$\dots$	0	0	$\dots$	0	0	1	0	$\dots$	0	$\dots$	0	0	0	1		
0	$\dots$	0	$\dots$	0	0	$\dots$	0	0	1	0	$\dots$	0	$\dots$	0	0	$\dots$	0	0	1
1										1									

Table 3.7 – Sous-problèmes construits par la règle de branchement $\text{EMM}^{[2]}$ définie dans la sous-section 3.4.1 pour le problème k-Hitting Set.

Démonstration. Par observation du tableau 3.7 décrivant les sous-problèmes de la règle $\text{EMM}^{[2]}$, il vient aisément que pour q, q' comme définis précédement, le vecteur de branchement de cette règle est $V^{[2]}_{\substack{\rightarrow \\ q,q'}} = \mathbb{1}_{\substack{\rightarrow \\ \ell}} \bigoplus \ell \bigodot \left\{ \mathbb{1}_{\substack{\rightarrow \\ q}} \bigotimes \mathbb{1}_{\substack{\rightarrow \\ q'}} \right\}$. Par les Propositions 3.10, 3.8 (vi), et 3.4 (ii), il vient $V^{[2]}_{\substack{\rightarrow \\ q,q'}} \leq_\tau V^{[2]}_{\substack{\rightarrow \\ k-\ell,k-\ell}}$, où $V^{[2]}_{\substack{\rightarrow \\ k-\ell,k-\ell}} = \mathbb{1}_{\substack{\rightarrow \\ \ell}} \bigoplus \ell \bigodot \left[\mathbb{1}_{\substack{\rightarrow \\ k-\ell}} \bigotimes \mathbb{1}_{\substack{\rightarrow \\ k-\ell}} \right]$. □

3.4.2 Algorithmes Généralisés

Algorithm k**-HS**$(\mathcal{H}', C, D)$

Règles de Réduction.

R0.0. Si $\mathcal{H}' = \emptyset$: `STOP`, C est un transversal de $\mathcal{H}$, et $C \in Tr(\mathcal{H})$ si et seulement si C est minimal.

R0.1. S'il existe $e \in \mathcal{H}'$ avec $e = \emptyset$: `STOP`, cette instance n'admet pas de solution.

R1.0. S'il existe $v \in V'$ de degré $d_{\mathcal{H}'}(v) = 0$: `RMV`(v).

R1.1. S'ils existent $e, e' \in E'$ avec $e \subseteq e'$: $\mathcal{H}'' \leftarrow \mathcal{H}' - e'$.

R1.2. S'il existe $e \in \mathcal{H}'$ avec $e = \{v\}$: `ADD`(v).

Règles de Branchement. Soit $x \in V'$ de degré maximum dans $\mathcal{H}'$.

B1. Si $\deg_{\mathcal{H}'}(x) = 1$: exécuter $\text{LME}^{[1]}$.

B2. Sinon : exécuter $\text{EMM}^{[2]}$.

Algorithme 3.2 – Algorithme de Branchement du Théorème 3.20 pour la version énumération du problème k-Hitting Set.

Théorème 3.20. *L'Algorithme 3.2 énumére les transversaux minimaux d'un hypergraphe de rang $k \in \mathbb{N}_{\geq 3}$ en temps $\tau(M^{[2]}_{\substack{\rightarrow \\ k}})$.*

Démonstration. Les règles de réduction de l'Algorithme 3.2 étant les mêmes que celles de l'Algorithme 3.1, elles se justifient de la même façon que dans le Lemme 3.15. La règle de branchement LME$^{[1]}$ est correcte. En effet, dans la première branche, un transversal C^* de $\mathcal{H}$ qui contiendrait à la fois x et un sommet x_i, $i \geq 2$, ne serait pas minimal, car $C^* - \{x\}$ serait un transversal de $\mathcal{H}$. La règle de branchement EMM$^{[2]}$ est correcte, elle énumère toutes les possibilités pour un transversal C^* de traverser à la fois e_1 et e_2. Le nombre de branchement de l'Algorithme 3.2 est donné par le Lemme 3.19, en observant que $V^{[2]}_{\substack{\rightarrow \\ k-1,k-1}} \geq_\tau L^{[1]}_{\substack{\rightarrow \\ k}}$. □

Le Théorème 3.20 permet d'améliorer certaines valeurs de la Table 3.1 pour $4 \leq k \leq 6$. La Table 3.9 présente certaines de ces valeurs.

k	4	5	6
γ_k	1.9004	1.9538	1.9779

Table 3.8 – Valeurs γ_k obtenues par le Théorème 3.20 pour le temps d'exécution d'un algorithme en temps $O^*(\gamma_k^n)$ pour la version énumération du problème k-Hitting Set.

3.5 Compression Iterative

La technique de *Compression Iterative* présentée en Sous-Section 1.5.6 est à l'origine une technique pour obtenir des algorithmes FPT [82]. En 2008, Fomin, Gaspers, Kratsch et al. ont appliqué cette technique pour obtenir des algorithmes exacts exponentiels pour la résolution de certains problèmes NP-difficiles [82]. En particulier, ils ont montré que, étant donné un algorithme pour dénombrer les transversaux minimums d'un hypergraphe de rang k en temps $O^*(\beta_k^n)$, nous pouvons construire un algorithme en temps $O^*(\beta_{k+1}^n)$ pour dénombrer les mêmes objets dans un hypergraphe de rang $k+1$. Avec la propriété que si $\beta_k < 2$ alors $\beta_{k+1} < 2$, les résultats de Fomin et al. conduisent à des algorithmes vérifiant $\beta_k < 2$ pour tout $k \in \mathbb{N}_{\geq 2}$, puisque $\beta_2 \leq 1.2377$ [187]. Ils montrent de plus que cette technique peut être appliquée à la version optimisation, c'est-à-dire étant donné un algorithme pour déterminer un transversal minimum d'un hypergraphe de rang $k \in \mathbb{N}_{\geq 2}$ en temps $O^*(\alpha_k^n)$, on peut construire un algorithme en temps $O^*(\alpha_{k+1}^n)$ pour déterminer un transversal minimum d'un hypergraphe de rang $k+1$.

Dans cette Sous-Section, nous étendons le champ d'application de la technique de *Compression Iterative* aux problèmes de Hitting Set. Nous montrons ainsi que cette technique peut être utilisée pour obtenir des algorithmes d'énumération des transversaux minimaux d'un hypergraphe de rang $k \in \mathbb{N}_{\geq 3}$ en temps $O^*(\gamma_k^n)$ à partir d'un algorithme d'énumération des transversaux minimaux d'un hypergraphe de rang $k-1$ en temps $O^*(\gamma_{k-1}^n)$. Pour se faire, nous montrons préalablement le Lemme 3.21 suivant.

Lemme 3.21. *Soit $\mathcal{H}$ un hypergraphe, et soit $T \in Tr(\mathcal{H})$. Soit $X \subseteq V$, $T' = X \cap T$, $X' = X - T$, et soit $\mathcal{H}' = (V - T, E')$, avec $E' = \{e \backslash (T - T') \ : \ e \in E, e \cap T' = \emptyset\}$. Alors on a :*

$$X \in Tr(\mathcal{H}) \Rightarrow X' \in Tr(\mathcal{H}')$$

Démonstration. Nous montrons ce lemme par l'absurde. Supposons par contradiction que X' ne soit pas un transversal de $\mathcal{H}'$. Alors il existe une hyperarête $e' \in \mathcal{H}'$ telle que $e' \cap X' = \emptyset$. Par définition, $e' \in \mathcal{H}'$ implique qu'il existe $\varepsilon \subseteq T - T'$ tel que $e = e' + \epsilon \in E$ et $e \cap T' = \emptyset$. Avec

$e = e' + \varepsilon$, $X' \cap e' = \emptyset$ par hypothèse, et $X' \cap \varepsilon = \emptyset$ car $X' \subseteq V - T$ et $\varepsilon \subseteq T - T' \subseteq T$, impliquent ensembles que $e \cap X' = \emptyset$. Avec $X = X' + T'$, $e \cap T' = \emptyset$ et $e \cap X' = \emptyset$, montrent que $X \cap e = \emptyset$. La déduction $X \cap e = \emptyset$ contredit que X est un transversal de $\mathcal{H}$, et nous avons donc montré que X' est un transversal de $\mathcal{H}'$. Montrons à présent que X' est un transversal minimal de $\mathcal{H}'$, c'est-à-dire montrons $X' \in Tr(\mathcal{H}')$. Supposons par contradiction que X' ne soit pas minimal, donc il existe $Y' \subset X'$ transversal de $\mathcal{H}'$. Alors nous affirmons que $Y' \cup T'$ est un transversal de $\mathcal{H}$. En effet, pour toute hyperarête $e \in \mathcal{H}$, si $e \cap T' = \emptyset$, alors $e' \cap Y' \neq \emptyset$, avec $e' \subseteq e$, donc $e \cap Y' \neq \emptyset$. Il vient que $Y' \cup T' \subset X$ est un transversal de $\mathcal{H}$, ce qui contredit que X est minimal. □

Le Lemme 3.22 étend le champ d'application de la technique de *Compression Iterative*.

Lemme 3.22. *Soit $O^*(\gamma_k^n)$ le temps d'exécution d'un algorithme pour énumérer les transversaux minimaux d'un hypergraphe de rang $k \geq \mathbb{N}_{\geq 2}$. Alors il y a un algorithme en temps $O^*(\gamma_{k+1}^n)$ pour énumérer les transversaux minimaux d'un hypergraphe de rang $k+1$, avec :*

$$\gamma_{k+1} = \min_{1/2<x\leq 1} \left(\frac{(1-x)^{x-1}}{x^x} + 2^x \cdot \gamma_k^{1-x} \right)$$

Démonstration. Soit $\mathcal{H} = (V, E)$ un hypergraphe de rang $(k+1)$. Tant qu'il ne trouve pas de transversal de taille xn, notre algorithme commence par énumérer tous les ensembles de sommets de taille xn.
Dans un premier cas, supposons que $\mathcal{H}$ n'admette pas de transversal de taille xn. Alors l'algorithme énumère par *Brute-Force* tous les sous-ensembles de taille yn, $x < y \leq 1$, et décide pour chacun d'entre eux s'il appartient à $Tr(\mathcal{H})$. Dans ce cas, cette façon d'énumérer $Tr(\mathcal{H})$ est correcte, puisque tous les transversaux de $\mathcal{H}$, et en particulier les minimaux, sont de taille au moins xn.
Dans un deuxième cas, supposons que $\mathcal{H}$ admette un transversal de taille xn, et soit T un tel transversal obtenu par l'algorithme. Alors pour tout sous-ensemble $T' \subseteq T$ de sommets, l'algorithme énumère les transversaux minimaux de $\mathcal{H}$ qui contiennent T' mais qui ne contiennent pas $T - T'$. Pour se faire, notre algorithme créé une nouvelle instance $\mathcal{H}'$ comme définie dans le Lemme 3.21. Puis pour chaque $X' \subseteq V - T$ transversal minimal de $Tr(\mathcal{H}')$, notre algorithme décide en temps polynomial si $X \cup T'$ s'il est minimal. Dans ce cas, cette façon d'énumérer $Tr(\mathcal{H})$ est également correcte, et s'appuie sur le Lemme 3.21.
Pour analyser le temps d'exécution de l'algorithme, observons préalablement que décider si un sous-ensemble de sommet est un transversal minimal de $\mathcal{H}$ peut être exécuté en temps $O^*(n^k)$. Dans le premier cas, le temps de l'algorithme est $\sum_{i=xn}^{n} \binom{n}{i} \leq n\binom{n}{xn}$. Dans le deuxième cas, le temps d'exécution de l'algorithme est $\binom{n}{xn}$ pour déterminer T. Puis l'algorithme énumère les 2-partitions $(T', T - T')$ de T en temps 2^{xn}. Finalement, pour $\mathcal{H}'$ fixé, l'algorithme énumère les transversaux minimaux $Tr(\mathcal{H}')$ en temps $O^*(\gamma_k^{(1-x)n})$. En effet, si $\mathcal{H}$ est de rang $k+1$, puisque T est un transversal de $\mathcal{H}$, alors pour tout $T' = T \cup Y$, $Y \neq \emptyset$: $\mathcal{T}'$ est de rang k car au moins un sommet de chaque hyperarête est supprimé dans $\mathcal{T}'$. Le temps total de l'algorithme est alors $O^*(\binom{n}{xn} + 2^{xn} \cdot \gamma_k^{(1-x)n})$ pour $1/2 \leq x \leq 1$ fixé. Finalement, la formule pour γ_{k+1} vient du fait que $\binom{n}{xn} = O^*((\frac{(1-x)^{x-1}}{x^x})^n)$. □

En reprenant la valeur $\gamma_3 = 1.6755$ du Théorème 3.17, le Théorème 3.23, en s'appuyant sur le Lemme 3.22, montre le pouvoir de la technique de *Compression Iterative* et permet d'améliorer certaines valeurs de la Table 3.1 pour des problèmes de k-Hitting Set.

Théorème 3.23. *Il y a un algorithme en temps $O^*({\gamma_4}^n)$ avec espace polynomial pour énumérer les transversaux minimaux d'un hypergraphe de rang $k = 4$, avec $\gamma_4 = 1.8866$.*

k	4	5	6
γ_k	1.8866	1.9550	1.9806

Table 3.9 – Valeurs γ_k obtenues par le Théorème 3.20 pour le temps d'exécution d'un algorithme de *Compression Iterative* en temps $O^*(\gamma_k^n)$ pour la version énumération du problème k-Hitting Set.

3.6 Conclusion

Dans ce chapitre, nous avons amélioré des algorithmes d'énumération pour nos problèmes de k-Hitting Set, en plus des résultats que nous avions précédemment publiés [42]. Ces améliorations ont été possibles par l'utilisation des techniques de *Branchement*, *Mesurer et Conquérir* ou de *Compression Iterative*. Nous renvoyons à la Table 3.2 pour un aperçu des temps d'exécution que nous avons obtenu.

Nous avons vu en Sous-Section 3.1.2 une borne inférieure sur le temps d'exécution d'un algorithme d'énumération pour le problème k-Hitting Set. Cependant, les temps d'exécution des algorithmes pour la version énumération du problème k-Hitting Set que nous présentons dans cette Thèse sont très loin de cette borne inférieure. Nous savons que pour $k = 2$, cette borne inférieure représente également une borne supérieure. Pour toutes les autres valeurs de $k \geq 3$, nous sommes donc intrigués par l'optimalité de cette borne.

Peut-on construire une famille d'hypergraphes de rang $k \in \mathbb{N}_{\geq 3}$ tels que les nombres de transversaux minimums ou minimaux soient plus grands encore que ceux présentés dans le Lemme 3.1 et correspondants dans la Table 3.3 ?

Les techniques que nous avons mises en œuvre dans ce Chapitre avaient déjà rencontré différents succès pour différentes versions du problème k-Hitting Set. Les algorithmes présentés en Section 3.4 sont des algorithmes généralisés. En outre, cela signifie que ces algorithmes s'appuyent sur la même idée, mais que leur application et leur temps d'exécution diffère selon la valeur de $k \in \mathbb{N}_{\geq 2}$. La technique de *Mesurer et Conquérir* est une technique efficace pour le problème de 3-Hitting Set. Cependant, nous ne savons pas si nous pouvons construire des algorithmes généralisés qui utilisent la technique de *Branchement*, et dont le temps d'exécution est analysé avec une *mesure* μ_k non triviale également généralisée.

Peut-on construire des algorithmes généralisés de Branchement et Mesurer et Conquérir, et dont l'analyse utilise une mesure généralisée pour une des versions du problème k-Hitting Set*, $k \geq 4$?*

Les algorithmes de la Section 3.4 constituent une généralisation à deux hyperarêtes des algorithmes publiés en 2014 [42]. Dans la littérature, plusieurs tentatives existent pour analyser l'imbrication de deux règles de *Branchement* successives. Analyser cette approche est en général très compliqué. Pour notre part, notre analyse prend en compte tous les cas d'imbrication possibles pour deux hyperarêtes particulières. L'analyse de la règle EMM$^{[2]}$ au Lemme 3.19 a été possible grâce à l'introduction des outils présentés en Section 3.2. Notre analyse s'est donc réduite à $k-1$ cas distincts possibles. Nous pensons que le pire des cas pour nos algorithmes est obtenu au cas $\ell = 1$.

Pour tout $k \in \mathbb{N}_{\geq 3}$, le pire cas pour le temps d'exécution des Algorithmes de la Section 3.4 est-il obtenu en $\ell = 1$?

Répondre à la question précédente nous permettrait d'améliorer définitivement pour tout $k \geq 7$ tous les algorithmes publiés en 2014. Une autre question se pose sur la possible amélioration de ces

algorithmes. Bien que la tâche paraisse ardue, nous pensons que nos outils peuvent nous permettre d'analyser une généralisation de nos algorithmes à trois hyperarêtes ou plus. Une question naturelle qui se pose est donc la suivante.

Peut-on pour tout $k \in \mathbb{N}_{\geq 3}$, généraliser à $p_k \geq 2$ hyperarêtes les algorithmes de la Section 3.4 ?

Finalement, notre dernière question se porte sur le problème co$k$-Hitting Set. Les résultats publiés en 2014 présentaient des algorithmes d'énumération plus efficaces pour le problème co$k$-Hitting Set que ceux présentés pour le problème k-Hitting Set. Nous pensons qu'il est possible d'adapter nos algorithmes de la Section 3.4 à ce problème. Cependant, nous ne savons pas jusqu'à quel point les résultats obtenus seront améliorés pour ce problème. Nous ne savons pas non plus si nous pouvons adapter notre algorithme d'énumération pour le problème 3-Hitting Set au problème co3-Hitting Set.

*Peut-on adapter chacun de nos algorithmes au problème co$k$-*Hitting Set *afin d'améliorer les valeurs $\overline{\gamma_k}$ pour tout $k \in \mathbb{N}_{\geq 3}$?*

Chapitre 4

Algorithmes Paramétrés pour les Racines de Graphes

Sommaire

4.1 Introduction

Racine Carrée. Les carrés et les racines carrées sont des notions classiques en théorie des graphes et sont définis comme suit. Le *carré* G^2 d'un graphe $G = (V_G, E_G)$ est un graphe avec pour ensemble de sommets V_G tel que toute paire de sommets distincts $u, v \in V_G$ sont voisins dans G^2 si et seulement si u et v sont à distance au plus 2 dans G. Un graphe H est une *racine carrée* de G si $G = H^2$.

> RACINE CARRÉE
> *VERSION* : *DÉCISION*
>
> **Entrée** : $G = (V, E)$
> **Sortie** : Existe-t-il H sous-graphe de G tel que $H^2 = G$?

Il existe des graphes qui n'admettent pas de racine carrée, des graphes qui possède une racine carrée unique, de même qu'il existe des graphes qui admettent plusieurs racines carrées. Par exemple, l'étoile $K_{1,n-1}$ engendre n racines carrées isomorphes du graphe complet K_n, et tout graphe à n sommets qui contient $K_{1,n-1}$ comme sous-graphe est également une racine carrée de K_n. Autre exemple, aucun chemin P_k, $k \in \mathbb{N}_{\geq 3}$, n'admet de racine carrée, pas plus qu'un graphe qui contiendrait un chemin induit P_3 avec pour sommet intérieur un sommet de degré $d = 2$, ou qu'un graphe

qui contiendrait un sommet de degré $d = 1$ au sein d'une composante connexe constituée d'au moins 3 sommets. ♣

Mukhopadhyay [161] a montré en 1967 qu'un graphe connexe G avec n sommets $v_1, \ldots, v_n$ possède une racine carrée si et seulement s'il existe un ensemble de n sous-graphes complets $K^1, \ldots, K^n$ de G avec $\bigcup_i V_{K^i} = V_G$ tels que K^i contient v_i pour tout $1 \leq i \leq n$, et K^i contient v_j si et seulement si K^j contient v_i pour tout $1 \leq i < j \leq n$. Cette caractérisation ne conduit pas à un algorithme en temps polynomial pour la reconnaissance des carrés de graphes. En réalité, Motwani et Sudan [160] ont montré en 1994 que le problème Racine Carrée, qui est de décider si un graphe admet une racine carrée, est NP-complet. Ce résultat fondamental a permis de nombreuses recherches sur la difficulté informatique à reconnaître des carrés de graphes et à calculer des racines carrées selon l'hypothèse de contraintes concernant diverses propriétés structurelles. En particulier, les deux problèmes Racine Carrée de Graphes et Carré de Graphes ont attiré beaucoup d'attention, et ont une complexité différente en fonction des familles de graphes $\mathcal{G}$ et $\mathcal{H}$ de l'entrée.

Racine Carrée de Graphes

Version : *Décision*

Entrée : $G = (V, E)$, $\mathcal{G}$ famille de graphes, avec $G \in \mathcal{G}$
Sortie : Existe-t-il H sous-graphe de G tel que $H^2 = G$?

Carré de Graphes

Version : *Décision*

Entrée : $G = (V, E)$, $\mathcal{H}$ famille de graphes
Sortie : G est-il le carré d'un graphe $H \in \mathcal{H}$?

Ross et Harary [175] ont caractérisé les carrés des arbres et ont prouvé que, si parmi les racines carrées d'un graphe connexe il n'y en a qu'une qui soit un arbre, alors cette racine carrée est unique à un isomorphisme près. Lin et Skiena [145] ont déterminé un algorithme linéaire de reconnaissance des carrés des arbres, et un algorithme linéaire pour décider si un graphe planaire possède une racine carrée, autrement dit un algorithme linéaire pour la reconnaissance des graphes planaires qui ont une racine carrée. Les résultats sur les arbres [145, 175] ont été généralisés aux graphes blocs par Le et Tuy [137]. Lau [134] a produit un algorithme en temps polynomial pour la reconnaissance des carrés des graphes bipartis. Lau et Corneil [135] ont produit un algorithme en temps polynomial pour reconnaître les carrés de graphes d'intervalles propres et ont montré que les problèmes de reconnaître les carrés de graphes cordaux, carrés de graphes split, et les graphes cordaux qui possèdent une racine carrée sont tous les trois NP-complets. Le et Tuy [138] ont produit un algorithme en temps quadratique pour reconnaître les carrés des graphes split fortement cordaux. Milanic et Schaudt [154] ont fournit un algorithme linéaire pour la reconnaissance des graphes trivialement parfaits et des graphes threshold qui possèdent une racine carrée. Adamaszek et Adamaszek [2] ont prouvé que si un graphe possède une racine carrée de maille (*girth*) au moins 6, alors cette racine carrée est unique à un isomorphisme près. Farzad, Lau, Le et Tuy [77] ont montré que reconnaître les graphes avec une racine carrée de circonférence au moins g est soluble en temps polynomial si $g \geq 6$ et NP-complet si $g = 4$. Le cas restant $g = 5$ a été démontré NP-complet par Farzad et Karimi [76].
Pour notre chapitre, nous introduisons et étudierons une résolution à paramètre fixe d'une variante du problème Racine Carrée, à savoir le problème de Racine Carrée Étiquetée.

RACINE CARRÉE ÉTIQUETÉE
VERSION : DÉCISION

Entrée : $G = (V, E)$ connexe, et $R, B \subseteq E_G$ avec $R \cap B = \emptyset$
Sortie : Existe-t-il un graphe H tel que $G = H^2$, $R \subseteq E_H$, et $B \cap E_H = \emptyset$?

4.1.1 Préliminaires

Nous référons aux livres de Downey et Fellows [65], Flum et Grohe [81], et Niedermeier [164] pour une introduction détaillée de la THÉORIE DE LA COMPLEXITÉ PARAMÉTRÉE.
Pour ce Chapitre, en supplément des notions déjà définies en Section 1.3, nous introduisons les terminologies de graphe suivantes. Pour alléger la notation, on pourra écrire pour un graphe $G = (V, E)$: $xy \in E$ au lieu de $\{x, y\} \in E$. Un sommet de degré $d = 1$ est appelé un sommet *pendant*. Deux sommets $u, v \in V$ sont dits *vrais jumeaux* si $N_G[u] = N_G[v]$. Si $N_G(u) = N_G(v)$ alors u et v sont dits *faux jumeaux*. Soit G un graphe connexe. Soit $S \subset V_G$, et soient X et Y deux sous-ensembles de sommets non vides et disjoints de $G - S$. Alors S est un *séparateur* de G si $G - S$ n'est pas connexe, S est un *(X,Y)-séparateur* si $G - S$ n'admet pas de chemin qui relie un sommet de X à un sommet de Y. S est un (X, Y)-séparateur *minimal* s'il est minimal au sens de la Section 1.1. De plus, G est 2-*connexe* si et seulement si $|V_G| \geq 3$ et G ne contient pas de séparateur de taille un.

Réduction de Noyau et Noyau Généralisé

Une technique bien connue pour montrer qu'un problème Π est soluble à paramètre fixé est de trouver une *Réduction* à un *Noyau*. Cette technique, traduit de l'anglais *kernelization*, remplace une instance (I, k) de Π par une instance réduite (I', k') de Π, appelé *noyau*, de sorte que les trois conditions suivantes sont satisfaites :
i) $k' \leq k$ et $|I'| \leq g(k)$ pour une certaine fonction calculable g
ii) la réduction de (I, k) à (I', k') est calculable en temps polynomial
iii) (I, k) est une yes-instance de Π si et seulement si (I', k') est une yes-instance de Π.
Dans la littérature, si nous modifions légèrement la définition en permettant à l'instance (I', k') d'appartenir à un problème différent de Π, alors (I', k') est appelé *noyau généralisé* de Π. Ce concept a été introduit et appelé *binoyau* par Alon, Gutin et al. [6]. Une borne supérieure $g(k)$ sur la taille $|I'|$ de l'instance I' est appelée la *taille du noyau*. Un noyau est dit *linéaire* si sa taille est linéaire en k et *quadratique* si sa taille est quadratique en k. Un résultat bien connu est qu'un problème est soluble à paramètre fixe si et seulement s'il dispose d'un noyau [164].

4.1.2 Nos résultats

Les résultats présentés dans ce Chapitre ont fait l'objet en 2013 d'une publication [41]. Nous entamons ce Chapitre par la Section 4.2, dans laquelle nous listons un ensemble de propriétés structurelles des racines carrées de graphes, qui nous seront utiles ensuite dans différentes sections. Ces résultats sont fondamentaux pour la construction de nos algorithmes.

Dans la Section 4.3, nous construisons un algorithme en temps polynomial pour reconnaître les graphes de degré maximum 6 qui admettent une racine carrée. Si une racine carrée existe, alors notre algorithme en détermine une avec un nombre minimum d'arêtes, et ainsi nous résolvons la version optimisation du problème RACINE CARRÉE MINIMUM pour les graphes de degré maximum 6. Il est possible de montrer que les graphes de degré maximum au plus 5 qui admettent une racine

carrée possèdent une largeur en chemin bornée, ce qui amène à un algorithme en temps linéaire pour la reconnaissance de tels graphes. Cependant, ce n'est pas le cas pour les graphes de degré maximum 6 : considérons le carré d'un *mur*, traduit de l'anglais *wall*, avec des arêtes subdivisées. Notre approche est de prétraiter un graphe d'entrée G de degré maximum au plus 6 dans le but d'obtenir un graphe de largeur en chemin bornée.

Dans les Sections 4.4 et 4.5, nous paramétrons le problème de racines carrées de graphes. À notre connaissance, il s'agit des premiers résultats de ce type pour ces problèmes. Le problème classique Racine Carrée est un problème de décision. Nous introduisons deux variantes d'optimisation dans le but de pouvoir discuter de la *paramétrisation* des racines carrées. Nous renvoyons à la Section 1.2 pour une définition de la classe de problèmes **FPT**. La racine carrée d'un graphe est l'union disjointe de racines carrées de ses composantes connexes, nous nous restreignons aux graphes connexes par simplicité. Nos résultats FPT peuvent s'étendre à des graphes non connexes. Nous considérons deux choix naturels de paramètre $k \in \mathbb{N}^*$ pour nos versions d'optimisation du problème Racine Carrée, et nous obtenons de cette manière les premiers algorithmes FPT pour les problèmes de racines carrées de graphes.

Racine Carrée Minimum
Version : *Paramétrée*

Entrée : $G = (V, E)$ connexe
Paramètre : $k \in \mathbb{N}$
Sortie : Existe-t-il un graphe H, tel que $G = H^2$ et $|E_H| \leq n - 1 + k$?

Racine Carrée Maximum
Version : *Paramétrée*

Entrée : $G = (V, E)$ connexe
Paramètre : $k \in \mathbb{N}$
Sortie : Existe-t-il un graphe H, tel que $G = H^2$ et $|E_H| \geq m - k$?

Dans la Section 4.4, nous paramétrons le problème Racine Carrée Minimum. Puisque toute racine carrée d'un graphe connexe G à n sommets est un sous-graphe connexe couvrant de G, toute racine carrée de G possède au moins $n - 1$ arêtes. La définition du problème Racine Carrée Minimum est donc naturelle, tout comme le choix du paramètre k qui mesure la distance en nombre d'arêtes d'une racine carrée de G à un arbre couvrant. Notre résultat principal est que le problème Racine Carrée Minimum est FPT pour le paramètre k.
Nous prouvons ce résultat en montrant qu'une instance du problème Racine Carrée Minimum peut être réduite à une instance d'un problème plus général, dans lequel nous imposons des conditions supplémentaires sur les arêtes, qui sont de devoir être incluses ou exclues de la racine carrée. Nous montrons que la taille de la nouvelle instance est quadratique en k. En d'autres termes, nous montrons que le problème Racine Carrée Minimum dispose d'un noyau généralisé de taille quadratique. Ce résultat est par la suite motivé par l'observation que le problème Racine Carrée Minimum généralise celui de reconnaître les carrés des arbres (avec $k = 0$). Une autre conséquence de notre résultat FPT est que le problème de reconnaître les carrés des graphes de la classe $\mathcal{G}_k = \{G \mid G$ est un graphe obtenu à partir d'un arbre avec adjonction d'au plus k arêtes$\}$ est soluble en temps polynomial pour tout $k \geq 0$ fixé. De cette manière, notre résultat peut aussi être vu comme une extension des résultats connus sur la reconnaissance des carrés de graphes [145].

Finalement, dans la Section 4.5, nous paramétrons le problème Racine Carrée Maximum, qui est de tester si un graphe G avec m arêtes possède une racine carrée avec au moins $m-k$ arêtes, pour un certain entier k donné. Nous montrons que ce problème est FPT de paramètre k. La définition du problème Racine Carrée Maximum et son paramètre sont également naturels, puisque toute racine carrée d'un graphe est un sous-graphe de lui-même. Un graphe G possède une racine carrée avec nombre d'arêtes au moins $m-k$ si cette racine peut être obtenue de G par au plus k suppressions d'arêtes. Notre paramètre k mesure en conséquence l'éloignement en nombre d'arêtes d'une racine carrée de G à lui-même. Ainsi, notre deuxième résultat FPT est en relation avec l'étude grandissante des résultats paramétrés concernant les problèmes d'édition de graphes, qui constitue un champ de problèmes bien étudié au sein de l'Algorithmique et de la Théorie des Graphes. Nous présentons également dans la Section 4.5 un algorithme exact exponentiel en temps $O^*(3^{m/3})$ pour le problème Racine Carrée Maximum, qui peut être vu comme une amélioration de l'algorithme induit par la caractérisation de Mukhopadhyay [161].

4.2 Propriétés Structurelles

Cette section est consacrée à des propriétés structurelles fondamentales pour nos résultats. Ils concernent la structure des racines carrées de graphes, et seront utilisés par la suite dans les différentes sections pour appuyer nos résultats. Nous commençons par l'observation suivante que nous utiliserons fréquemment.

Observation 4.1. Soit H une racine carrée d'un graphe connexe G.
i) Si u est un sommet pendant de H, alors u est un sommet simplicial de G.
ii) Si u, v sont des sommets pendants de H voisins du même sommet, alors u, v sont des vrais jumeaux dans G.
iii) Si u, v sont des sommets pendants de H voisins de sommets différents, alors u et v ne sont pas voisins dans G à moins que $H = K_2$.

Nous formulons à présent cinq lemmes qui seront utiles pour la suite. Les Lemmes 4.2 et 4.3 peuvent être trouvés implicitement dans le papier de Ross et Harary [175]. Ross et Harary [175] ont considéré les racines carrées d'arbres, tandis que nous nous intéressons à trouver des racines carrées en général. Pour cette raison nous donnons les formulations explicites des Lemmes 4.2 et 4.3. Nous donnons également une preuve du Lemme 4.3, celle du Lemme 4.2 est directe.

Lemme 4.2. *Soit H une racine carrée d'un graphe G. Soit $\{u_1, \ldots, u_r\} \subseteq V_H$ un sous-ensemble de sommets qui induisent une étoile dans H avec sommet central u_1, pour un certain entier $r \geq 3$. Soit $u_3, \ldots, u_r$ des sommets pendants dans H et soit $\{u_2\}$ un $(\{u_1, u_3, \ldots, u_r\}, V_H \setminus \{u_1, \ldots, u_r\})$-séparateur de H. Alors $\{u_1, \ldots, u_r\}$ est une clique de G, et $\{u_1, u_2\}$ est un $(\{u_3, \ldots, u_r\}, V_G \setminus \{u_1 \ldots, u_r\})$-séparateur minimal de G.*

Lemme 4.3. *Soit G un graphe connexe de racine carrée H. On pose $U = \{u_3, \ldots, u_r\}$ et $U^* = \{u_1, u_2\} + U$, tels que U^* est une clique de G de taille $r \geq 3$, et $\{u_1, u_2\}$ est un $(U, V_G - U^*)$-séparateur minimal de G. On pose $X = \{x_1, \ldots, x_p\} = N_G(u_1) - U^*$ pour un certain $p \geq 1$, et $Y = \{y_1, \ldots, y_q\} = N_G(u_2) - U^*$ pour un certain $q \geq 1$, comme montré dans la Figure 4.1. Notons que l'on n'interdit pas X et Y de s'intersecter. Alors les trois propriétés suivantes sont satisfaites :*
i) Figure 4.2 i : $u_1u_2 \in E_H$ et, soit $\forall\, u \in U, \forall\, x \in X$: $\{u_1, u\} \in E_H$, $\{u_2, u\} \notin E_H$, $\{u_1, x\} \notin E_H$, et $\{u_2\}$ est un $(U + \{u_1\}, V_H - U^*)$-séparateur minimal dans H, ou alors symétriquement $\forall\, u \in U$, $\forall\, y \in Y$: $\{u_1, u\} \notin E_H$, $\{u_2, u\} \in E_H$, $\{u_2, y\} \notin E_H$ et $\{u_1\}$ est un $(U + \{u_2\}, V_H - U^*)$-séparateur minimal dans H.

ii) Figure 4.2 ii : Si u_1, u_2 sont des vrais jumeaux dans G, alors soit $\forall\ x \in X\ :\ \{u_1, x\} \in E_H$ ou alors $\forall\ x \in X\ :\ \{u_2, x\} \in E_H$. De plus, G est l'union de deux graphes complets avec ensemble de sommets respectifs U^* et $\{u_1, u_2\}+X$, et G admet deux racines carrées (isomorphes) avec ensemble d'arêtes respectifs $\{u_1u_2, \ldots, u_1u_r\} \cup \{u_2x_1, \ldots, u_2x_p\}$ et $\{u_2u_1, u_2u_3, \ldots, u_2u_r\} \cup \{u_1x_1, \ldots, u_1x_p\}$.

iii) Figure 4.2 iii : Si $N_G[u_2] \setminus N_G[u_1] \neq \emptyset$, alors $u_1u_2 \in E_H$, $\forall\ u \in U$, $\forall x \in X\ : \{u_1, u\} \in E_H$, $\{u_2, u\} \notin E_H$, et $\{u_1, x\} \notin E_H$. De plus, le graphe H' obtenu à partir de H en supprimant toutes les arêtes u_iu_j, $3 \leq i < j \leq r$, est une racine carrée de G (dans laquelle U^* induit une étoile de sommet central u_1 et avec feuilles $U + \{u_2\}$ qui sont, hormis u_2, des sommets pendants).

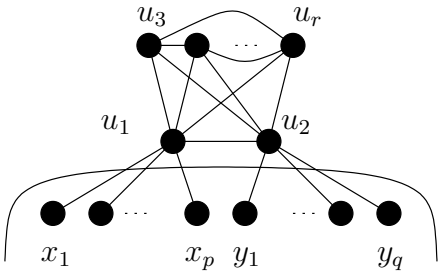


Figure 4.1 – Le graphe G du Lemme 4.3. Observons que $p \geq 1$ et $q \geq 1$, puisque $\{u_1, u_2\}$ est un $(\{u_3, \ldots, u_r\}, V_G \setminus \{u_1, \ldots, u_r\})$-séparateur minimal de G. Observons également qu'il est possible que $x_i = y_j$ pour certaines valeurs $1 \leq i \leq p$ et $1 \leq j \leq q$.

Démonstration. Nous commençons par montrer la propriété i). $\{u_1, u_2\}$ est un $(\{u_3, \ldots, u_r\}, V_G \setminus \{u_1, \ldots, u_r\})$-séparateur de G, donc au moins un sommet u_i avec $r \geq 3$ est adjacent à u_1 ou u_2 dans H. Par symétrie, nous pouvons supposer qu'il s'agit de u_1. Alors $u_1x_1, \ldots, u_1x_p \notin E_H$; en effet, si u_1 est adjacent à un certain x_j dans H, alors $u_ix_j \in E_G$, ce qui contredit le fait que $\{u_1, u_2\}$ est un $(\{u_3, \ldots, u_r\}, V_G \setminus \{u_1, \ldots, u_r\})$-séparateur de G. De plus, puisque $u_1x_1, \ldots, u_1x_p \notin E_H$, au moins un sommet y_h doit être voisin à u_2 dans H (autrement H ne serait pas connexe et donc ne pourrait pas être racine carrée de G, dont on rappelle qu'il est connexe). Ensuite, du fait que $\{u_1, u_2\}$ est un $(\{u_3, \ldots, u_r\}, V_G \setminus \{u_1, \ldots, u_r\})$-séparateur de G, nous déduisons que $u_3u_2, \ldots, u_ru_2 \notin E_H$. En conséquence, $u_1u_2 \in E_H$, et $\{u_2\}$ est un $(\{u_1, u_3, \ldots, u_r\}, V_H \setminus \{u_1, \ldots, u_r\})$-séparateur minimal dans H. Supposons qu'il y ait un sommet u_i, $3 \leq i \leq r$, tel que $u_iu_1 \notin E_H$. $u_3, \ldots, u_r$ ne sont pas voisins de u_2, alors il vient que tout chemin reliant u_2 à u_i dans H a une longueur au moins 3, ce qui n'est pas possible puisque $u_2u_i \in E_G$. Nous concluons que $u_3u_1, \ldots, u_ru_1 \in E_H$. Nous avons donc montré i).

Nous montrons à présent ii). Observons que $\{x_1, \ldots, x_p\} = \{y_1, \ldots, y_q\}$ avec $p = q$. Par la propriété i), u_1 ou u_2 n'est voisin d'aucun x_i. Dans le premier cas, u_2 doit être voisin de tous les x_i dans H, autrement il n'y aurait aucun chemin (nécessaire) de longueur au plus 2 dans H entre x_i et u_1. Pour les mêmes raisons, dans le deuxième cas, u_1 doit être voisin de tous x_i dans H. Ceci implique que $\{u_1, u_2, x_1, \ldots, x_p\}$ est une clique dans G. Si H a une arête x_iz avec $z \notin \{u_1, \ldots, u_r, x_1, \ldots, x_p\}$, alors $zu_2 \in E_G$, ce qui n'est pas possible. Ceci signifie que G est l'union de deux graphes complets avec ensembles de sommets respectifs $\{u_1, \ldots, u_r\}$ et $\{u_1, u_2, x_1, \ldots, x_p\}$. Nous observons alors G possède deux racines carrées (isomorphes) avec ensembles d'arêtes respectifs $\{u_1u_2, \ldots, u_1u_r\}$ $\cup\{u_2x_1, \ldots, u_2x_p\}$ et $\{u_2u_1, u_2u_3, \ldots, u_2u_r\} \cup\{u_1x_1, \ldots, u_1x_p\}$. Nous avons donc montré ii).

Nous montrons finalement iii). Soit $y_i \in N_G[u_2] \setminus N_G[u_1] \neq \emptyset$. Par i) nous avons que $u_1u_2 \in E_H$, et que soit $u_3u_1, .., u_ru_1 \in E_H$, $u_3u_2, \ldots, u_ru_2 \notin E_H$, $u_1x_1, \ldots$, $u_1x_p \notin E_H$, ou alors que $u_3u_1, \ldots, u_ru_1 \notin E_H$, $u_3u_2, \ldots, u_ru_2 \in E_H$, $u_2y_1, \ldots, u_2y_q \notin E_H$. Si le dernier cas se présente, alors

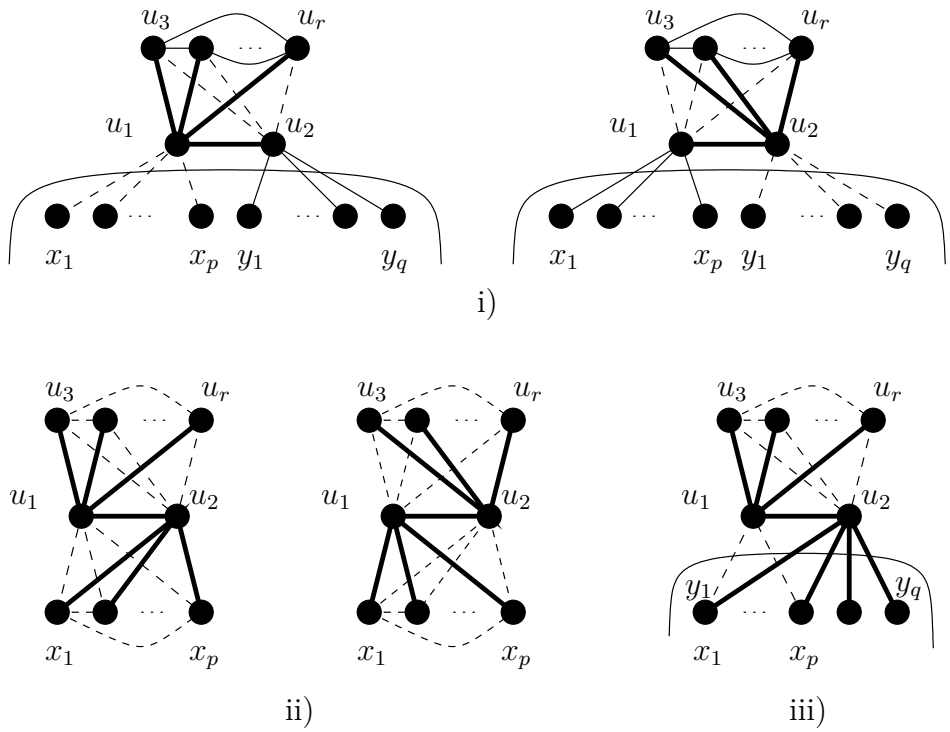


Figure 4.2 – Racines carrées du graphe G correspondant respectivement aux trois propriétés i)–iii) du Lemme 4.3. Les arêtes de G qui appartiennent à sa racine carrée sont montrées en lignes pleines épaisses, tandis que les arêtes de G qui n'appartiennent pas à sa racine carrée sont montrées en lignes pointillées. En i), les arêtes de G qui peuvent être ajoutées dans la racine carrée de G sont montrées en lignes continues fines. Les racines carrées des points ii) et iii) sont les racines carrées particulières définies dans les propriétés correspondantes ii) et iii) du Lemme 4.3.

tout chemin qui relie u_2 à y_i dans H a une longueur au moins 3, qui n'est pas possible car $u_2y_i \in E_G$. Donc le dernier cas ne peut se présenter. Dans le premier cas, soit H' un graphe obtenu à partir de H en supprimant toutes les arêtes u_iu_j, pour $i, j \in \{3, \dots, r\}$. Nous pouvons alors observer que $H'^2 = H^2 = G$. Nous avons donc montré iii). □

Soit G un graphe contenant (parmi peut-être d'autres sommets) $p+q+r$ sommets distincts $u_1, \dots, u_r$, $x_1, \dots, x_p$, $y_1, \dots y_q$ pour certains $r \geq 3$, $p \geq 1$ et $q \geq 1$, tel que les propriétés suivantes soient satisfaites :
i) $\{u_1, \dots, u_r\}$ est une clique dans G ;
ii) Si $r \geq 4$, alors $\{u_1, u_2, u_3\}$ est un $(\{u_4, \dots, u_r\}, V_G \setminus \{u_1, \dots, u_r\})$-séparateur minimal dans G ;
iii) $\{u_1, u_3, \dots, u_r\} \cup \{x_1, \dots, x_p\} \cup \{y_1, \dots, y_q\} = N_G(u_2)$;
iv) $\{u_2, u_4, u_5, \dots, u_r\} = N_G(u_1) \cap N_G(u_3)$;
v) $\{x_1, \dots, x_p\} \subseteq N_G(u_1)$ et $\{y_1, \dots, y_q\} \subseteq N_G(u_3)$;
vi) $x_iy_j \notin E_G$ pour tous $i = 1, \dots, p$ et $y = 1, \dots, q$.
Nous appelons G un *F-graphe* et $\{u_1, u_2, u_3\}$ un *F-triplet* avec *sommets sortants* u_1 et u_3, voir Figure 4.3 pour un exemple. Ici, F fait référence au graphe de la Figure 4.4. Ces notions sont expliquées plus loin dans les Lemmes 4.4 et 4.5.

Lemme 4.4. *Soit H une racine carrée d'un graphe G. Supposons que H contienne le graphe F de la Figure 4.4 comme sous-graphe. Supposons de plus que $u_4, \dots, u_r$ sont des sommets pendants de H (si $r \geq 4$), $d_H(u_2) = r - 1$, que $u_1u_2u_3$ est un chemin induit dans H qui n'est contenu dans aucun*

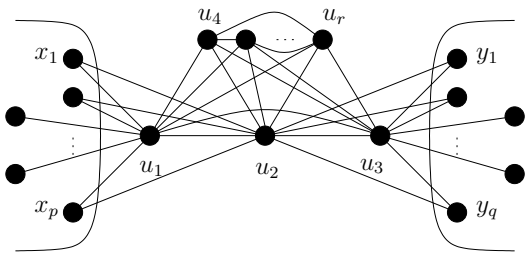


Figure 4.3 – Un exemple d'un F-graphe avec $r \geq 4$. Observons qu'il n'y a aucune arête entre deux sommets x_i et y_j. Observons également que les deux sommets sortants u_1 et u_3 du F-triplet $\{u_1, u_2, u_3\}$ peuvent être adjacents à des sommets non voisins de u_2 (mais aucun voisin commun dans $\{x_1, \ldots, x_p\} \cup \{y_1, \ldots, y_q\}$).

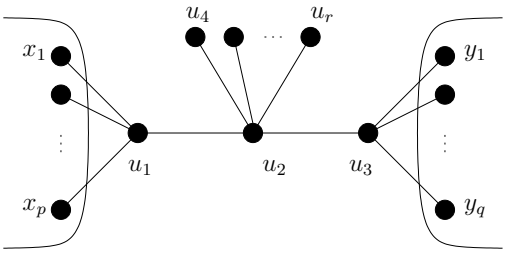


Figure 4.4 – Le graphe $F = F(p, q, r)$ avec $p \geq 1$, $q \geq 1$ et $r \geq 3$; si $r = 3$ alors F ne contient pas de sommet pendant $u_4, \ldots, u_r$. Nous avons représenté F comme un sous-graphe du graphe H dans le Lemme 4.4. Pour être plus précis, le graphe F est exactement le graphe avec les arêtes noires. Dans H les sommets $u_1, \ldots, u_r$ ont seulement des voisins qui sont dans F, alors qu'un sommet x_i ou y_j peut avoir d'autres voisins dans H qui ne sont pas dans F ; cependant, un x_i ou un y_j ne peut avoir de voisin commun dans H. De plus, dans H, les seules arêtes incidentes aux sommets de F sont représentées par les arêtes noire dans la figure (arêtes de F). Les possibles arêtes entre deux sommets x_i, x_j ou y_i,y_j n'ont pas été représentées dans cette figure.

cycle de longueur au plus 6*, que* $\{x_1, \ldots, x_p\} = N_H(u_1) \setminus \{u_2\}$ *et que* $\{y_1, \ldots, y_q\} = N_H(u_3) \setminus \{u_2\}$*. Alors* G *est un* F*-graphe.*

Démonstration. D'après la définition d'un F-graphe, nous observons que les propriétés (i)-(iii) et (v) sont satisfaites. Les propriétés iv) et vi) sont également satisfaites du fait que le chemin $u_1u_2u_3$ n'est contenu dans aucun cycle de longueur au plus 6 dans H. □

Lemme 4.5. *Soit* G *un* F*-graphe connexe. Si* H *est une racine carrée de* G*, alors le graphe* F *de la Figure 4.4 est un sous-graphe de* H *tel que* $d_H(u_2) = r - 1$*,* $\{x_1, \ldots, x_p\} = N_H(u_1) \setminus \{u_2\}$ *et* $\{y_1, \ldots, y_q\} = N_H(u_3) \setminus \{u_2\}$*. De plus, le graphe obtenu à partir de* H *en supprimant toutes les arêtes* u_iu_j*, pour* $4 \leq i < j \leq r$*, est une racine carrée de* G *qui contient* $u_4, \ldots, u_r$ *comme sommets pendants (si* $r \geq 4$*).*

Démonstration. Soit H une racine carrée de G. Nous considérons les trois cas suivants.

Cas 1. $u_1u_2, u_2u_3 \in E_H$. $x_1u_3, \ldots, x_pu_3 \notin E_G$, alors $x_1u_2, \ldots, x_pu_2 \notin E_H$. Symétriquement, $y_1u_2, \ldots, y_qu_2 \notin E_H$. Puisque chaque $x_iu_2 \in E_G$ et puisque $x_iu_2 \notin E_H$, alors H a un chemin de longueur 2 reliant x_i à u_2. Du fait que $d_G(u_2) = p+q+r-1$, le sommet du milieu de ce chemin est dans l'ensemble $\{u_1, u_3, \ldots, u_r\}$ (tout autre sommet aurait pour effet d'augmenter $d_G(u_2)$). Du fait que x_i n'est voisin à aucun sommet u_j dans H, $3 \leq j \leq r$, puisque ce n'est pas le cas dans G, alors le chemin passe par u_1. En d'autres termes, $x_1u_1, \ldots, x_pu_1 \in E_H$ et, par symétrie, $y_1u_3, \ldots, y_qu_3 \in E_H$. Si un sommet $z \notin \{u_2, x_1, \ldots, x_p\}$ est adjacent à u_1 dans H, alors z est adjacent à u_2 et également à x_1 dans G. Du fait que $d_G(u_2) = p + q + r - 1$, nous déduisons que $z \in \{u_3, \ldots, u_r\}$ ou que

$z \in \{y_1, \ldots, y_q\}$. Cependant, aucun des sommets de $\{u_3, \ldots, u_r\}$ n'est voisin de x_1, et aucun des sommets de $\{y_1, \ldots, y_q\}$ n'est voisin de u_2. Nous concluons donc que $\{x_1, \ldots, x_p\} = N_H(u_1) \setminus \{u_2\}$ et en utilisant les mêmes arguments que $\{y_1, \ldots, y_q\} = N_H(u_3) \setminus \{u_2\}$.
Nous montrons maintenant que $u_4u_2, \ldots, u_ru_2 \in E_H$. Supposons qu'un certain u_i, $4 \leq i \leq r$, n'est pas adjacent à u_2 dans H. Alors u_1 et u_i sont à distance au moins 3 dans H ce qui contredit que $u_1u_i \in E_G$. Nous avons déjà déduit que $x_1u_2, \ldots, x_pu_2 \notin E_H$ et que $y_1u_2, \ldots, y_qu_2 \notin E_H$. Par hypothèse, u_2 est voisin de u_1 et de u_3. Puisque $d_G(u_2) = p+q+r-1$, nous avons que $d_H(u_2) = r-1$. Pour achever la preuve de ce cas, il reste à observer que s'il existe u_i et u_j voisins dans H, pour $i, j \in \{4, \ldots, r\}$, alors le graphe H' obtenu à partir de H en supprimant ces arêtes u_i, u_j est une racine carrée de G.

Cas 2. $u_1u_2, u_2u_3 \notin E_H$. Puisque $u_1u_2 \notin E_H$, $u_1u_2 \in E_G$ et $d_G(u_2) = p + q + r - 1$, il existe un sommet $z \in \{x_1, \ldots, x_p\} \cup \{u_4, \ldots, u_r\}$ tel que $u_1z, zu_2 \in E_H$. Parce que z n'est adjacent à aucun sommet $y_1, \ldots, y_q$ dans G, alors nous avons que $y_1u_2, \ldots, y_qu_2 \notin E_H$. Par les mêmes arguments, nous obtenons $x_1u_2, \ldots, x_pu_2 \notin E_H$. Ainsi, $z \in \{u_4, \ldots, u_r\}$. Par symétrie, un certain sommet de l'ensemble $\{u_4, \ldots, u_r\}$ est voisin de u_3 dans H. En conséquence, chaque sommet de $\{u_1, u_2, u_3\}$ est voisin d'un certain sommet de l'ensemble $\{u_4, \ldots, u_r\}$ dans H. Puisque $\{u_1, u_2, u_3\}$ sépare $\{u_4, \ldots, u_r\}$ de $V_G \setminus \{u_1, \ldots, u_r\}$, cela signifie que H ne contient pas d'arêtes qui relie u_1, u_2, u_3 avec les sommets de $V_G \setminus \{u_1, \ldots, u_r\}$; une contradiction. En conséquence, ce cas n'est pas possible.

Par symétrie, il reste à considérer le cas suivant.

Cas 3. $u_1u_2 \in E_H$ et $u_2u_3 \notin E_H$. Parce que $u_1u_2 \in E_H$ et $y_1u_1, \ldots, y_qu_1 \notin E_G$, nous avons que $y_1u_2, \ldots, y_qu_2 \notin E_H$. Parce que $y_1u_2 \in E_G$, cela signifie que H contient un chemin de longueur 2 qui relie y_1 à u_2. $u_2u_3 \notin E_H$ et $d_G(u_2) = p + q + r - 1$, alors un tel chemin doit passer par un des sommets de l'ensemble $\{u_1, u_4, \ldots, u_r\} \cup \{x_1, \ldots, x_p\}$. Cependant, aucun de ces sommets n'est voisin de y_1 dans G, et donc dans H non plus ; une contradiction. Nous concluons que ce cas n'est pas possible non plus. □

Lemme 4.6. *Soient u, v deux vrais jumeaux dans un graphe connexe G qui contient au moins trois sommets. Soit G' le graphe obtenu à partir de G en supprimant v. Les deux propriétés suivantes sont alors satisfaites :*
i) Si H' est une racine carrée de G', alors le graphe H obtenu à partir de H' en ajoutant v avec $N_H(v) = N_{H'}(u)$ (c'est-à-dire, en ajoutant un faux jumeaux de u) est une racine carrée de G.
ii) Si H est une racine carrée de G tel que u, v sont faux jumeaux dans H, alors le graphe H' obtenu à partir de H en supprimant v est une racine carrée de G'.

Démonstration. Nous montrons i). Soit H' une racine carrée de G', et soit H le graphe obtenu à partir de H' en ajoutant un faux jumeau v de u. G est un graphe connexe avec au moins trois sommets, alors u est voisin dans H' à un certain sommet z. u et v sont donc voisins de z dans H, et donc $d_H(u, v) \leq 2$. Ainsi, uv est une arête de H^2. Il vient ainsi que $G = H^2$. La propriété ii) est une conséquence du fait qu'identifier des faux jumeaux ne modifie la distance d'aucune paire de sommets. □

4.3 Graphes de Degré Maximum 6

Dans cette Section, nous montrons que le problème Racine Carrée Minimum est soluble en temps polynomial lorsque l'entrée est un graphe de degré maximum $\Delta(G) \leq 6$. Nous renvoyons à la Section 4.4 pour une définition du problème Racine Carrée Minimum Étiquetée. Le Lemme 4.7 constitue une base de notre résultat.

Lemme 4.7. *Soit $G = (V, E)$ un graphe. Il y a un algorithme en temps $O(n \cdot f(\mathbf{tw}(G)))$ pour résoudre le problème Racine Carrée Minimum Étiquetée, pour une certaine fonction f.*

Démonstration. Bien que nous pourrions écrire un algorithme de *Programmation Dynamique Arborescente* pour ce problème, nous donnons ici une preuve plus courte non constructive de ce lemme. Nous exprimons le problème de l'existence d'une racine carrée étiquetée en logique **MSOL**. L'existence d'un graphe H tel que $G = H^2$, $R \subseteq E_H$ et $B \cap E_H = \emptyset$ est équivalent à l'existence d'un sous-ensemble $X \subseteq E_G$ d'arêtes qui vérifie $R \subseteq X$, $B \cap X = \emptyset$, et :
◇ pour toute arête $uv \in E_G$, soit $uv \in X$ ou alors il existe $uw, wv \in X$
◇ pour tout couple $uw, vw \in X$ d'arêtes distinctes, $uv \in E_G$.
Notre problème est donc expressible en logique **MSOL**. Il suffit d'appliquer le Théorème de Courcelle [49] pour achever la preuve de notre lemme. □

Soit $G = (V, E)$ un graphe connexe, et soit $u \in V_G$. On note $L_0(u), \ldots, L_{s(u)}(u)$ les différents *niveaux* d'un *parcours en largeur* à partir de u, c'est-à-dire $L_i(u) = \{v \in V_G \mid \text{dist}_G(u, v) = i\}$ pour $i = 1, \ldots, s(u)$, avec $s(u) = \max dist_G(u, v)$ étant l'*excentricité* de u. On pose également $L_i = \emptyset$ pour tout $i > s(u)$. Nous établissons le Lemme 4.8 suivant pour borner la largeur linéaire d'un graphe, que nous appliquons dans le Lemme 4.9 à une certaine famille de graphes de degré maximum $\Delta(G) \leq 6$.

Lemme 4.8. *Soit $G = (V, E)$ un graphe connexe, et soit $u \in V_G$. Alors :*

$$\mathbf{pw}(G^2) + 1 \leq \max_{0 \leq i \leq s(u)} \{|L_i(u) \cup L_{i+1}(u) \cup L_{i+2}(u)|\}$$

Démonstration. Nous construisons une décomposition en chemin (X, P), où P est un chemin ordonné de sommets $0, \ldots, s(u)$. Pour tout $i \in \{0, \ldots, s(u)\}$, on définit $X_i = L_i(u) \cup L_{i+1}(u) \cup L_{i+2}(u)$. Par définition $L_0(u), \ldots, L_s(u)$ constituent une partition de V_G. En conséquence $\bigcup_{i=0}^{s(u)} X_i = V_G$. Pour toute arête $xy \in E_{G^2}$ telle que $x \in L_i(u)$ et $y \in L_j(u)$, il vient $|i - j| \leq 2$. En conséquence par construction de notre décomposition, pour toute arête $xy \in E_{G^2}$: $x, y \in X_i$, pour un certain $i \in \{0, \ldots, s(u)\}$. Finalement, si un sommet $x \in X_i \cap X_j$ avec $i + 1 < j$, alors $i + 1 = j$ et $x \in L_{i+2} \subseteq X_{i+1}$. Cela signifie que l'ensemble $\{i \mid x \in X_i\}$ induit un sous-chemin de P.
Nous achevons la preuve en observant que par construction, la largeur de (X, P) est donnée par $\max_{0 \leq i \leq s}\{|L_i(u) \cup L_{i+1}(u) \cup L_{i+1}(u)|\} - 1$. □

Lemme 4.9. *Soit G un graphe satisfaisant $\Delta(G) \leq 6$, et soit H une racine carrée de G. Si tout chemin induit xyz de H avec $\{d_H(y) = 2, d_H(x) \geq 2, d_H(y) \geq 2\}$ est inclus dans H dans un cycle induit de longueur au plus 6, alors $\mathbf{pw}(G) \leq 71$.*

Démonstration. Sans perte de généralité, nous supposons que G est connexe. Si G n'est pas connexe, nous considérons ses composantes connexes séparément. Soit $u \in V_G$. Nous effectuons un parcours en largeur BFS dans H à partir de u pour obtenir les niveaux $L_0(u), \ldots, L_s(u)$. Un sommet y est appelé un *fils* d'un sommet x dans BFS si $xy \in E_G$, $x \in L_i(u)$ et $y \in L_{i+1}(u)$, pour un certain $i \in \{0, \ldots, s(u) - 1\}$. Dans un tel cas, on dit de plus que x est un *père* de y. Observons qu'un sommet peut avoir plusieurs pères. Pour trois sommets x, y, z, on dit que z est un *petit-fils* de x si y est un fils de x et si z est un fils de y. On dit de plus dans un tel cas que x est un *grand-père* de y. Finalement, on dit que y est un *descendant* de x si $x \in L_i(u)$, $y \in L_j(u)$ avec $i < j$, et il existe un chemin de longueur $j - i$ dans G qui relie x à y. Nous montrons préalablement une série d'affirmations.

Affirmation 0. $\forall\ x \in V$: *x est adjacent dans G à ses fils, petits-fils, pères et grand-pères de BFS.*
Affirmation 1. $\forall\ i \geq 2, \forall\ x \in L_i(u)$ *: la somme du nombre des fils et des petits-fils de x est au plus* 4*.*
L'Affirmation 1 est une conséquence de l'Affirmation 0 et du fait que $\forall\ x \in V : d_G(x) \leq 6$. Ces deux remarques sont importantes pour nos futurs raisonnements.

Affirmation 2. *Si* $x \in L_i$*,* $i \geq 2$*, et si x possède au moins deux petits-fils, alors seul un fils y de x possède des fils.*

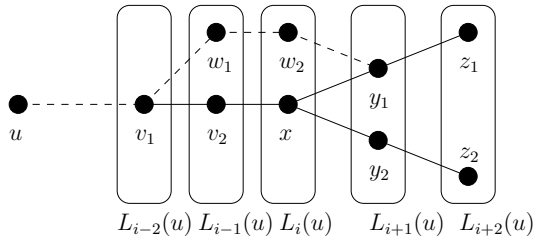


Figure 4.5 – Configuration de x et de ses petits-fils.

Nous montrons cette affirmation par contradiction. Supposons que x possède au moins deux fils distincts $y_1 \neq y_2$, chacun possédant des fils. Par hypothèse, x possède au moins deux petits-fils distincts. Posons $z_1 \neq z_2$ les petits-fils de x pour lesquels y_1 est un père de z_1 et y_2 est un père de z_2. Notons que nous n'excluons pas y_2 d'être un père de z_1, ni même que nous n'excluons pas y_1 d'être un fils de z_2. Avec $i \geq 2$: x possède un père et un grand-père. Soit alors v_2 un père de x, et soit v_1 un père de v_2. Voir Fig. 4.5.
Dans un premier temps, observons que v_1v_2x est un chemin induit de H. En effet, si v_2 est adjacent à un autre sommet z, alors étant donné les niveaux de la configuration, on a $z \neq y_1, y_2, z_1, z_2$. Or par l'affirmation 0, $d_G(x) \geq 7$, ce qui est une contradiction. En conséquence, $d_H(v_2) = 2$.
Observons à présent que puisque $i \geq 2$, alors $d_H(v_1) \geq 2$. Avec $d_H(x) \geq 2$, alors par la condition de H énoncée dans le lemme, v_1v_2x est inclus dans H dans un cycle induit C de longueur au plus 6.
Si x a un voisin $r \neq v_2, y_1, y_2$ dans H, on a $r \neq v_1, z_1, z_2$ sinon r ou x n'est pas dans le bon niveau de BFS. En conséquence, par l'Affirmation 0, $d_G(x) \geq 7$ dans G. Cette contradiction montre que $d_H(x) = 3$, et en conséquence l'arête xy_1 ou xy_2 est contenue dans C. Supposons par symétrie que $xy_1 \in C$. Alors $C = v_1v_2xy_1w_2w_1$ comme montré dans la Fig. 4.5. Par conséquent, w_2 est un nouveau sommet distinct des autres qui satisfait $w_2 \in N_G(x)$, et donc $d_G(x) \geq 7$. Cette contradiction finale achève la preuve de l'Affirmation 2.

Affirmation 3. *Si* $x \in L_i$*,* $i \geq 2$*, alors pour tout* $j > i$ *: x possède au plus* 4 *descendants dans* $L_j(u)$*.*
Par l'Affirmation 1, x possède au plus 4 descendants dans $L_j(u)$ quand $j = i + 1$ et $j = i + 2$. Si $i = s - 1$ ou $i = s - 2$, l'affirmation est donc vérifiée par la précédente observation. Montrons l'affirmation 3 par récurrence. Soit $i < s - 2$, et supposons que l'affirmation soit vérifiée pour tout $k > i$. Par la première observation, la propriété est vérifiée pour $j = i + 1$ et $j = i + 2$.
Si x possède un unique petit-fils z, alors pour tout $j \geq i + 3$, les descendants de x sont exactement les descendants de z dans $L_j(u)$. Par hypothèse de récurrence, notre affirmation est donc vérifiée. Sinon si x possède plus de deux petits-fils, alors par l'Affirmation 2, x possède un unique fils y qui possède des descendants. Et nous concluons avec la même remarque que précédemment avec les descendants de y.

Nous achevons désormais la preuve de notre Lemme 4.9. Trivialement, $|L_0(u)| = 1$. Par l'Affirmation 0, on a alors $|L_1(u)| \leq 6$ et $|L_2(u)| \leq 5$. Soit $p = |L_1(u)|$. Du fait que $\forall\ x \in L_1(u) : d_G(x) \leq 6$, alors pour tout $x \in L_1(u)$: x possède au plus $5 - p$ petit-fils car il est garanti dans G que x est adjacent à u, à tous les autres sommets de $L_1(u)$, ainsi qu'à un de ses fils. $L_3(u)$ vérifie donc $|L_3(u)| \leq p(5-p)$. En observant que si $p = 6$, alors $L_2(u) = L_3(u) = \emptyset$, $L_3(u)$ satisfait alors $|L_3(u)| \leq \max_{1\leq p\leq 5}\{p(5-p)\} = 6$. Par l'Affirmation 3, $|L_4(u)| \leq 4|L_3(u)| \leq 24$. On montre finalement par récurrence en utilisant les Affirmations 2 et 3 que pour tout $i \geq 5$, $|L_i(u)| \leq 4|L_3(u)| \leq 24$. En appliquant le Lemme 4.8, on a alors $\mathbf{pw}(G) \leq 71$. □

Nous pouvons à présent montrer le résultat principal de cette Section.

Théorème 4.10. *Il y a un algorithme en temps polynomial $O(n^5)$ pour résoudre le problème de Racine Carrée Minimum d'un graphe de degré maximum au plus 6.*

Démonstration. Notre algorithme procède en deux étapes.
Soit G un graphe satisfaisant $\Delta(G) \leq 6$. Nous utilisons deux ensembles d'arêtes R et B, et nous essayons de déterminer une racine carrée de G qui contient les arêtes de R et qui ne contient aucune arête de B. En d'autres termes, nous résolvons le problèmes de Racine Carrée Minimum Étiquetée. Nous initialisons $R = \emptyset$ et $B = \emptyset$. Nous appliquons ensuite la Règle de Réduction de Chemins définie ci-après. Ici, nous disons qu'une séquence $u_1, \ldots, u_\ell$ est *maximale* si elle ne peut être étendue en ajoutant de nouveaux sommets au début ou à la fin de la séquence car elle ne remplirait pas l'ensemble des conditions i)–v) de la règle suivante :

Règle de Réduction de Chemins.

1. Déterminer une séquence de sommets $u_1, \ldots, u_\ell$, $\ell \geq 3$ maximale telle que
   i) u_i, u_{i+1}, u_{i+2} sont tous deux à deux adjacents, $i \in \{1, \ldots, \ell - 2\}$
   ii) les ensembles $\{x_1, \ldots, x_p\} = N_G(u_1) \cap N_G(u_2) \setminus \{u_1, u_2, u_3\}$ et $\{y_1, \ldots, y_q\} = N_G(u_{\ell-1}) \cap N_G(u_\ell) \setminus \{u_{\ell-2}, u_{\ell-1}, u_\ell\}$ ne sont pas vides
   iii) $d_G(u_2) = p+q+2$ si $\ell = 3$, et $d_G(u_2) = p+3$, $d_G(u_{\ell-1}) = q+3$, $d_G(u_i) = 4$ for $i \in \{3, \ell-2\}$ si $\ell \geq 4$
   iv) $N_G(u_i) \cap N_G(u_{i+2}) = \{u_{i+1}\}$, pour $i \in \{1, \ldots, \ell - 2\}$
   v) si $\ell \leq 4$, alors $x_1u_\ell, \ldots, x_pu_\ell \notin E_G$, $y_1u_1, \ldots, y_qu_1 \notin E_G$, et si $\ell = 3$, alors $x_iy_j \notin E_G$ pour $i \in \{1, \ldots, p\}$ et $j \in \{1, \ldots, q\}$.
2. On pose $R' = \{u_1u_2, u_2u_3, \ldots, u_{\ell-1}u_\ell\} \cup \{x_1u_1, \ldots, x_pu_1\} \cup \{y_1u_\ell, \ldots, y_qu_\ell\}$ et $B' = \{x_1u_2, \ldots, x_pu_2\} \cup \{y_1u_2, \ldots, y_qu_2\} \cup \{u_iu_{i+2} | 1 \leq i \leq \ell - 2\} \cup \{zu_1 | z \in N_G(u_1) \setminus \{u_2, u_3, x_1, \ldots, x_p\}\} \cup \{zu_\ell | z \in N_G(u_\ell) \setminus \{u_{\ell-2}, u_{\ell-1}, y_1, \ldots, y_q\}\}$.
3. Si $R \cap B' \neq \emptyset$ ou $R' \cap B \neq \emptyset$, alors arrêter et retourner `no`.
4. Supprimer les sommets $u_2, u_4 \ldots, u_\ell$ de G, et supprimer également l'arête u_1u_3 si $\ell = 3$. Modifier $R = (R \cup R') \cap E_G$ et $B = (B \cup B') \cap E_G$. Modifier $s = s - \ell + 1$.

Pour montrer que la Règle de Réduction de Chemins est valide, considérons une instance (G, R, B, s) du problème Racine Carrée Minimum Étiquetée et supposons que $u_1, \ldots, u_\ell$ soit une séquence de sommets qui satisfasse i)–v) de l'étape 1. Par le Lemme 4.5, pour toute racine carrée H de G (si elle existe) : $R' \subseteq E_H$ et $B' \cap E_H = \emptyset$ pour les ensembles R' et B' construits à l'étape 2. En conséquence, si $R \cap B' \neq \emptyset$ ou $R' \cap B \neq \emptyset$, alors nous avons une no-réponse. Supposons que nous ne nous soyons pas arrêté à l'étape 3, et notons $(\hat{G}, \hat{R}, \hat{B}, \hat{s})$ l'instance du problème Racine Carrée Minimum Étiquetée obtenu à l'étape 4. Soit H une solution de (G, R, B, s). Puisque $R' \subseteq E_H$ et $B' \cap E_H = \emptyset$ par le Lemme 4.5, il suffit de vérifier directement si le graphe $\hat{H}$ obtenu à partir de H par suppression de $u_2, \ldots, u_{\ell-1}$ est une solution de $(\hat{G}, \hat{R}, \hat{B}, \hat{s})$. D'un autre côté, si $\hat{H}$ est une

solution de $(\hat{G}, \hat{R}, \hat{B}, \hat{s})$, alors H obtenu en joignant u_1 et u_ℓ par un chemin de longueur $\ell - 1$ est une solution pour (G, R, B, s).
Nous appliquons la Règle de Réduction de Chemins de manière récursive le plus longtemps possible. Supposons que nous ne nous soyons pas arrêté et retourné `no`. Pour simplifier la notation, supposons que (G, R, B, s) est l'instance obtenue du problème Racine Carrée Minimum Étiquetée. Puisque nous ne pouvons plus appliquer la Règle de Réduction de Chemins, alors par le Lemme 4.4, nous pouvons conclure que pour toute racine carrée H de G : H ne possède pas de chemin induit $u_1u_2u_3$ avec $d_H(u_2) = 2$, $d_H(u_1) \geq 2$ et $d_H(u_3) \geq 2$ qui ne soit pas inclus dans H dans un cycle induit de longueur au plus 6, autrement nous pourrions appliquer la règle pour la séquence maximale qui contient u_1, u_2, u_3. Par le Lemme 4.9, si G admet une racine carrée alors $\mathbf{pw}(G) \leq 71$. En utilisant l'Algorithme de Bodlaender [23], nous pouvons vérifier $\mathbf{pw}(G) \leq 71$. Si $\mathbf{pw}(G) > 71$, alors nous pouvons conclure que G n'admet pas de racine carrée. Dans le cas contraire, nous résolvons le problème de Racine Carrée Minimum Étiquetée en utilisant le Lemme 4.7.

Pour conclure la preuve, il reste à analyser le temps d'exécution. Chaque application de la Règle de Réduction de Chemins peut être exécutée en temps $O(n^3 \cdot m)$. Nous pouvons vérifier chaque triplet u_1, u_2, u_3 de sommets adjacents en temps $O(n^3)$. Nous pouvons ensuite construire les ensembles $N_G(u_1) \cap N_G(u_2) \setminus \{u_1, u_2, u_3\}$, $N_G(u_1) \cap N_G(u_2) \setminus \{u_1, u_2, u_3\}$ et vérifier les conditions i)–v) en temps $O(m)$. Observons que nous ne pouvons étendre la séquence $u_1, \ldots, u_i$ pour $i \geq 3$ à la seule condition que $N_G(u_{i-1}) \cap N_G(u_i) \setminus \{u_{i-2}, u_{i-1}, u_i\}$ contienne exactement un élément. Par conséquence, le temps total pour obtenir une séquence maximale, pour u_1, u_2, u_3 donné, est $O(m)$. Egalement vérifier si l'instance est une no-réponse à l'étape 3, et la construction de la nouvelle instance peut être effectué en temps linéaire. Puisque la règle est appliquée au plus n fois, nous déduisons que le temps total pour cette étape est $O(n^5)$. Finalement, il suffit d'observer que l'Algorithme de Bodlaender [23] s'exécute en temps linéaire, et que le problème Racine Carrée Minimum Étiquetée peut être résolu également en temps linéaire sur les graphes de largeur arborescente bornée. □

4.4 Racine Carrée Minimum

Comme présenté dans la Section 4.1.2, nous ne considérons que les graphes connexes. Nous référons à la Section 4.2 pour les résultats structurels sur lesquels nous nous appuierons afin de montrer le Théorème 4.11 suivant.

Théorème 4.11. *Le problème* Racine Carrée Minimum *peut être résolu en temps* $2^{O(k^4)} + O(n^4 m)$ *sur les graphes à* n *sommets et* m *arêtes.*

Nous considérons le problème paramétré Racine Carrée Minimum Étiquetée qui généralise le problème Racine Carrée Minimum.

Racine Carrée Minimum Étiquetée
Version : *Paramétrée*

Entrée : $G = (V, E)$ connexe, et $R, B \subseteq E_G$ avec $R \cap B = \emptyset$
Paramètre : $k \in \mathbb{N}$
Sortie : Existe-t-il un graphe H, tel que $G = H^2$, $|E_H| \leq n - 1 + k$, et qui satisfait : $R \subseteq E_H$ et $B \cap E_H = \emptyset$?

Pour le problème Racine Carrée Minimum Étiquetée, les ensembles R et B représentent respectivement des sous-ensembles donnés d'arêtes *requises*, c'est-à-dire qui doivent appartenir à H, et

d'arêtes *bloquées*, c'est-à-dire qui ne doivent pas appartenir à H. Observons que comme prévu le problème Racine Carrée Minimum Étiquetée généralise le problème Racine Carrée Minimum, en choisissant $R = B = \emptyset$.
Dans la Section 4.4.1, nous réduisons le problème Racine Carrée Minimum à celui de Racine Carrée Minimum Étiquetée avec pour taille de graphe dans l'instance obtenue de $O(k^2)$. En d'autres termes, nous construisons un noyau généralisé quadratique pour le problème Racine Carrée Minimum. Cela signifie que pour résoudre une instance du problème Racine Carrée Minimum, nous pouvons résoudre l'instance obtenue du problème Racine Carrée Minimum Étiquetée par un algorithme brute-force. Nous analyserons le temps d'exécution correspondant en fin de Section 4.4.1, ce qui achèvera la preuve du Théorème 4.11.

Nous décrivons succinctement les étapes majeures de la réduction. Soit G un graphe connexe à n sommets, et soit k un entier positif.

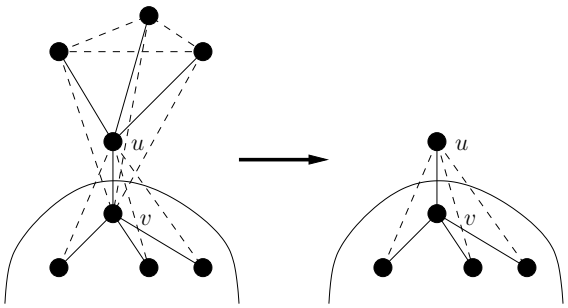


Figure 4.6 – Elagage ; les arêtes de H sont représentées en lignes pleines.

Supposons que H soit une racine carrée de G avec au plus $n + k - 1$ arêtes. Si H contient un sommet u de degré au moins 2 qui a exactement un voisin non-pendant v, alors nous reconnaissons la structure correspondante dans G et nous supprimons ces sommets de G qui sont des sommets pendants de H adjacents à u comme montré dans la Figure 4.6, c'est-à-dire, d'une manière similaire à l'algorithme de Lin et Skiena [145], nous "élaguons" les arêtes pendantes dans une racine carrée potentielle. Puisque la racine carrée que nous recherchons n'est pas un arbre, notre élagage est plus sophistiqué et est basé sur les Lemmes 4.2 et 4.3. Nous montrerons que de cette façon nous obtenons un graphe G' avec n' sommets qui possède les propriétés suivantes : chaque sommet pendant d'une racine carrée quelconque H' de G', avec au plus $n' - 1 + k$ arêtes, est adjacent à un sommet qui possède dans son voisinage au moins deux sommets non-pendants dans H'.
Supposons que H' possède un chemin induit P qui soit suffisamment long, tel que tout sommet interne de P possède dans son voisinage exactement deux sommets non-pendants dans H'. Soit u un sommet interne de P, et soient $x, y \in V_P$ deux voisins non-pendants de u. En utilisant les Lemmes 4.4 et 4.5, nous reconnaissons la structure correspondante dans G' et nous modifions G' comme montré dans la Figure 4.7, c'est-à-dire que nous supprimons u de H' et nous relions x et y par une arête. En réalisant ces opérations récursivement, nous obtenons un graphe G'' avec n'' sommets.
Supposons que H'' soit une racine carrée de G'' avec au plus $n'' + k - 1$ arêtes. Soit H^* le graphe obtenu à partir de H'' en supprimant tous les sommets pendants de H''. Alors H^* ne possède pas de sommet de degré 1, et la longueur de chaque chemin P avec sommets internes de degré 2 dans H^* est bornée par une constante. Cela signifie que la taille de H^* est $O(k)$. Les sommets de $V_{G''} \setminus V_{H^*}$ sont des sommets pendants de H''. Considérons l'ensemble Z des sommets pendants de H'' voisins d'un sommet $u \in V_{H^*}$. Alors les sommets de Z sont simpliciaux dans G''. De plus, ce sont des vrais

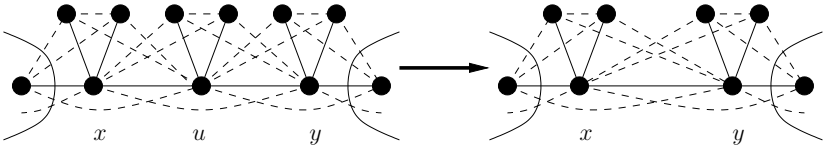


Figure 4.7 – Réduction de chemins ; les arêtes de H' sont représentées en lignes pleines.

jumeaux. Nous utilisons l'Observation 4.1 et le Lemme 4.6 pour montrer que nous pouvons réduire le nombre de vrais jumeaux dans G'' si G'' en possède trop. Cela nous conduit à un graphe G''' avec n''' sommets tel que n''' est $O(k^2)$.
Au cours de la réduction de G en G'' nous étiquetons certaines arêtes, c'est-à-dire, nous ajoutons certaines arêtes aux ensembles R ou B et, par la suite, nous obtenons une instance (G''', k, R, B) du problème Racine Carrée Minimum Etiquetée.

4.4.1 Noyau Généralisé

Dans cette Sous-Section, nous réduisons le problème Racine Carrée Minimum au problème Racine Carrée Minimum Etiquetée de sorte que le graphe de l'instance obtenue ait une taille $O(k^2)$. Avant de donner une description formelle de notre réduction, nous introduisons les définitions suivantes. Une racine carrée H d'un graphe G qui possède au plus $|V_G| - 1 + k$ arêtes pour un certain $k \geq 0$ est appelé une *solution* de l'instance (G, k) du problème Racine Carrée Minimum. Si $R \subseteq E_H$ et $B \cap E_H = \emptyset$ pour deux sous-ensembles disjoints R et B de E_G, alors H est également appelé une *solution* de l'instance (G, k, R, B) du problème Racine Carrée Minimum Etiquetée.
Nous sommes désormais prêts à donner les détails exacts de notre réduction. Soit G un graphe connexe avec n sommets et m arêtes, et soit k un entier positif. Nous vérifions dans un premier temps si G possède une racine carrée qui soit un arbre en utilisant l'algorithme en temps linéaire de Lin et Skiena [145]. Si nous trouvons une telle racine carrée, alors nous nous arrêtons et nous retournons une yes-réponse. À partir d'ici nous supposons que toute racine carrée de G (si elle existe) possède au moins un cycle.
Puisque les graphes connexes qui admettent des racines carrées sont 2-connexes, nous vérifions également si G est 2-connexe. Si ce n'est pas le cas, nous nous arrêtons et nous retournons une no-réponse. Si G est 2-connexe, nous poursuivons comme suit. Nous introduisons deux ensembles d'arêtes R et B. Initialement, nous imposons $R = B = \emptyset$. Ensuite, nous "élaguons" les arêtes pendantes dans les racines carrées potentielles, c'est-à-dire, nous appliquons exhaustivement la règle suivante qui consiste en cinq étapes qui doivent être réalisées dans l'ordre mentionné.

Règle d'Elagage

1. Trouver une paire $S = \{u_1, u_2\}$ de deux sommets voisins tel que une composante connexe de $G - S$ soit composée de $r \geq 3$ sommets $u_3, \ldots, u_r$ qui ensemble avec u_1, u_2 induisent une clique dans G.
2. Si $N_G[u_1] = N_G[u_2]$ alors arrêter et retourner une no-réponse.
3. Si $N_G[u_1] \setminus N_G[u_2] \neq \emptyset$ et $N_G[u_2] \setminus N_G[u_1] \neq \emptyset$, alors arrêter et retourner une no-réponse.
4. Si $N_G[u_1] \setminus N_G[u_2] \neq \emptyset$, alors renommer u_1 en u_2 et u_2 en u_1 (cette étape est uniquement une convention de préférence).
5. Définir les ensembles $R' = \{u_1u_2, \ldots, u_1u_r\}$ et $B' = \{u_iu_j \mid 2 \leq i < j \leq r\} \cup \{u_1x \mid x \in N_G(u_1) \setminus \{u_2, \ldots, u_r\}\}$.

6. Si $R \cap B' \neq \emptyset$ ou $R' \cap B \neq \emptyset$, alors arrêter et retourner une no-réponse. Sinon, modifier $R = R \cup R'$, $B = B \cup B'$, supprimer $u_3, \ldots, u_r$ de G, et supprimer de R et B toutes les arêtes incidentes à $u_3, \ldots, u_r$.

Appliquer cette règle d'élagage de manière récursive conduit à une séquence d'instances $(G_0, k, R_0, B_0), \ldots, (G_\ell, k, R_\ell, B_\ell)$ du problème Racine Carrée Minimum Etiquetée pour un certain $\ell \geq 0$, où $(G_0, k, R_0, B_0) = (G, k, \emptyset, \emptyset)$ et où $(G_\ell, k, R_\ell, B_\ell)$ est soit une instance pour laquelle nous avons retourné une no-réponse (à l'étape 2, 3 ou 6), ou alors une instance pour laquelle il n'existe pas d'ensemble S comme définit à l'étape 1. Pour $0 \leq i \leq \ell - 1$ nous dénotons par R'_i et B'_i les ensembles respectifs R' et B' construits au $(i+1)$-ème appel de la règle d'élagage. Nous avons besoin des lemmes suivants.

Lemme 4.12. *L'instance $(G_\ell, k, B_\ell, R_\ell)$ ne possède pas de solution qui soit un arbre, et G_ℓ est 2-connexe. De plus, $(G_\ell, k, R_\ell, B_\ell)$ admet une solution si et seulement si (G_0, k, R_0, B_0) admet une solution. Si la règle d'élagage a retourné une no-réponse pour $(G_\ell, k, R_\ell, B_\ell)$, alors (G_0, k, R_0, B_0) n'admet pas de solution.*

Démonstration. Pour $0 \leq i \leq \ell$, nous utilisons la récurrence pour montrer que le graphe G_i est 2-connexe et que (G_i, k, B_i, R_i) n'admet pas de solution qui soit un arbre. De plus, pour tout $1 \leq i \leq \ell$, nous montrons que (G_i, k, R_i, B_i) possède une solution si et seulement si $(G_{i-1}, k, R_{i-1}, B_{i-1})$ possède une solution. Finalement, nous montrons que si la règle d'élagage a retourné une no-réponse pour $(G_\ell, k, R_\ell, B_\ell)$, alors (G_0, k, R_0, B_0) n'admet pas de solution.
Si $i = 0$, alors G_i est 2-connexe et (G_0, k, B_0, R_0) n'admet pas de solution qui soit un arbre par hypothèse initiale (puisque nous avons pré-traité G pour ces deux propriétés). Maintenant supposons que $1 \leq i \leq \ell$. Par notre hypothèse de récurrence, nous pouvons supposer que G_{i-1} est 2-connexe et que $(G_{i-1}, k, B_{i-1}, R_{i-1})$ n'admet pas de solution qui soit un arbre.
Puisque la règle d'élagage appliquée à $(G_{i-1}, k, R_{i-1}, B_{i-1})$ a conduit à une nouvelle instance (G_i, k, R_i, B_i), le graphe G_{i-1} possède un couple $S = \{u_1, u_2\}$ de sommets voisins tel que les sommets $u_3, \ldots, u_r$ d'une composante connexe de $G - S$ forment ensemble avec u_1, u_2 une clique dans G_{i-1}. L'étape 6 implique que $G_i = G_{i-1} - \{u_3, \ldots, u_r\}$. Parce que nous n'avons pas retourné de no-réponse pour $(G_{i-1}, k, R_{i-1}, B_{i-1})$, nous déduisons que $N_{G_{i-1}}[u_1] \subset N_{G_{i-1}}[u_2]$. Ainsi, G_{i-1} n'est pas un graphe complet. Parce que G_{i-1} est 2-connexe, alors G_i est 2-connexe. Nous montrons maintenant qu'à toute solution de $(G_{i-1}, k, B_{i-1}, R_{i-1})$ correspond une solution de (G_i, k, B_i, R_i), et réciproquement.
Supposons dans un premier temps que H_{i-1} soit une solution quelconque de $(G_{i-1}, k, B_{i-1}, R_{i-1})$. Soit $N_{G_{i-1}}(u_1) \setminus \{u_2, \ldots, u_r\} = \{x_1, \ldots, x_p\}$. Puisque $N_{G_{i-1}}[u_2] \setminus N_{G_{i-1}}[u_1] \neq \emptyset$, nous avons que $G_{i-1} - \{u_1, u_2\}$ contient au moins deux composantes connexes. G_{i-1} est 2-connexe implique que $\{u_1, u_2\}$ est un $(\{u_3, \ldots, u_r\}, V_{G_{i-1}} \setminus \{u_1, \ldots, u_r\})$-séparateur minimal de G_{i-1}. Nous pouvons alors appliquer le Lemme 4.3 iii), qui stipule que $u_2u_1, \ldots, u_ru_1 \in E_{H_{i-1}}$, $u_3u_2, \ldots, u_ru_2 \notin E_{H_{i-1}}$, et $u_1x_1, \ldots, u_1x_p \notin E_{H_{i-1}}$. Avec $R_i \subseteq R_{i-1} \cup \{u_1u_2\}$ et $B_i \subseteq B \cup \{u_1x_1, \ldots, u_1x_p\}$, nous avons que la graphe obtenu à partir de H_{i-1} par suppression de $u_3, \ldots, u_r$ est une solution de (G_i, k, R_i, B_i) ; en particulier nous observons que $|E_{H_i}| \leq |E_{H_{i-1}}| - (r-3) \leq |V_{G_{i-1}}| - 1 + k - (r-3) = |V_{G_i}| - 1 + k$, ce qui était demandé.
Supposons à présent que H_i soit une solution quelconque de (G_i, k, R_i, B_i). Alors ajouter les arêtes $u_1u_3, \ldots, u_1u_r$ à H_i conduit à un graphe H qui est une racine carrée de G_{i-1}. Les arêtes $u_1u_3, \ldots, u_1u_r$ ne sont pas dans B_{i-1}, puisqu'elles sont dans l'ensemble R'_{i-1} construit à l'étape 5 et $R'_{i-1} \cap B_{i-1} = \emptyset$ (autrement la règle d'élagage se serait arrêté en traitant $(G_{i-1}, k, R_{i-1}, B_{i-1})$ dans l'étape 6). Maintenant supposons que R_{i-1} contienne une arête n'appartenant pas à H. Par définition de R_i, cette arête doit relier deux sommets u_s et u_t, pour $3 \leq s < t \leq r$. Alors u_su_t appartient à R_i, parce qu'elle avait été placée dans l'ensemble R_h pour un certain $h \leq i-1$. A l'étape

4 de l'appel correspondant de la règle d'élagage, une des arêtes u_su_1 or u_tu_1 avait également été placée dans B_h. Ainsi l'arête u_su_1 ou l'arête u_tu_1 appartient à B_{i-1}. Cela conduit à une contraction puisque u_su_1 et u_tu_1 appartiennent ensemble à R'_{i-1} et $R'_{i-1} \cap B_{i-1} = \emptyset$ (sans cela la règle d'élagage se serait arrêté lors de la construction de $(G_{i-1}, k, R_{i-1}, B_{i-1})$ à l'étape 6). Ainsi, après avoir observé que $|E_H| = |E_{H_i}| + (r-3) \leq |V_{G_i}| - 1 + k + (r-3) = |V_{G_{i-1}}| - 1 + k$, nous pouvons conclure que H est une solution de $(G_{i-1}, k, R_{i-1}, B_{i-1})$. Nous observons que H_i ne peut être un arbre, autrement cela impliquerait que H est un arbre, ce qui est impossible puisque $(G_{i-1}, k, R_{i-1}, B_{i-1})$ ne possède pas une telle solution.
Il ne nous reste plus qu'à montrer que si la règle d'élagage a retourné une no-réponse pour $(G_\ell, k, R_\ell, B_\ell)$, alors (G_0, k, R_0, B_0) ne possède pas de solution. Avec ce que nous avons montré précédemment, cela revient désormais à montrer que $(G_\ell, k, R_\ell, B_\ell)$ ne possède pas de solution.
Supposons que la règle d'élagage a retourné une no-réponse pour $(G_\ell, k, R_\ell, B_\ell)$. Alors cela a dû se produire lors de l'étape 2, 3 ou 6, en particulier après l'étape 1. Ainsi, il existe un couple $S = \{u_1, u_2\}$ de sommets adjacents dans G_ℓ, tel que une composante connexe de $G_\ell - S$ d'ensemble de sommets $\{u_3, \ldots, u_r\}$ forme une clique avec $\{u_1, u_2\}$.
Supposons d'abord que S ne soit pas un séparateur de G_ℓ, c'est-à-dire, G_ℓ est un graphe complet avec ensemble de sommets $\{u_1, \ldots, u_r\}$. Alors $N_G[u_1] = N_G[u_2]$ (et la no-réponse a été donnée à l'étape 2 de la règle d'élagage). Pour obtenir une contradiction, supposons que $(G_\ell, k, R_\ell, B_\ell)$ possède une solution H. Toute étoile avec $|V_{G_\ell}|$ sommets est une racine carrée de G_ℓ avec au plus $|V_{G_\ell}| - 1 + k$ arêtes. Cependant, H ne peut pas être une telle étoile, puisque $(G_\ell, k, R_\ell, B_\ell)$ ne possède pas de solution qui soit un arbre. En conséquence, $R_\ell \neq \emptyset$ ou $B_\ell \neq \emptyset$. Rappelons que $B_0 = R_0 = \emptyset$. Ainsi, $\ell \geq 1$, et le fait que R_ℓ ou B_ℓ ne soit pas vide doit avoir été obtenu lors d'un précédent appel de la règle d'élagage, que nous pouvons appeler le $(h+1)$-ème appel, pour un certain $0 \leq h \leq \ell - 1$. Par définition des étapes 5 et 6, nous avons que $B_h \neq \emptyset$ implique que $R_h \neq \emptyset$. Nous avons alors que $R_h \neq \emptyset$. Soit l'arête $u_iu_j \in R_h$. Par les étapes 5 et 6, cette arête dispose d'un sommet terminal, u_i, tel que $u_iu_s \in B_\ell$ pour tout $s \in \{1, \ldots, r\} \setminus \{i, j\}$. En conséquence, $u_ju_s \in E_H$ pour tout $s \in \{1, \ldots, r\} \setminus \{j\}$. Puisque l'étoile avec sommet central u_j et feuilles $V_{G_\ell} \setminus \{u_j\}$ n'est pas une solution de $(G_\ell, k, R_\ell, B_\ell)$, il doit y avoir une arête $u_su_t \in R_\ell$ avec $s, t \in \{1, \ldots, r\} \setminus \{j\}$. Cependant alors, par les étapes 5 et 6, $u_ju_s \in B_\ell$ ou $u_ju_t \in B_\ell$, c'est-à-dire, au moins une de ces arêtes ne peut être contenue dans H ; une contradiction.
Supposons maintenant que S est un séparateur de G_ℓ. Puisque G_ℓ est 2-connexe, u_1 et u_2 ont tous les deux au moins un voisin dans $V_{G_\ell} \setminus \{u_1, \ldots, u_r\}$. Alors $\{u_1, u_2\}$ est un séparateur minimal (et nous pouvons appliquer le Lemme 4.3 par la suite). Rappelons que la règle d'élagage retourne une no-réponse uniquement aux étapes 2, 3, ou 6. Nous considérons donc ces trois cas séparément.

Cas 1. La no-réponse est donnée dans l'étape 2. Alors $N_G[u_1] = N_G[u_2]$. Par le Lemme 4.3 i) et ii), G_ℓ est l'union de deux cliques $\{u_1, \ldots, u_r\}$ et $\{u_1, u_2, x_1, \ldots x_p\}$ où $\{x_1, \ldots, x_p\} = N_G(u_1) \setminus \{u_2, \ldots, u_r\}$. Pour obtenir une contradiction, nous supposons que $(G_\ell, k, R_\ell, B_\ell)$ n'a pas de solution H. Par le Lemme 4.3 i) et ii), nous pouvons supposer sans perte de généralité que $u_1u_2, \ldots, u_1u_r \in E_H$, $u_2u_3, \ldots, u_2u_r \notin E_H$, $u_1x_1, \ldots, u_1x_p \notin E_H$ et $u_2x_1, \ldots, u_2x_p \in E_H$. Rappelons que $(G_\ell, k, R_\ell, B_\ell)$ ne possède pas de solution qui soit un arbre. Ainsi, il existe une arête $u_iu_j \in R_\ell$ pour $i, j \in \{2, \ldots, r\}$ ou il existe une arête $x_ix_j \in R_\ell$ pour $i, j \in \{x_1, \ldots, x_p\}$. Par symétrie, nous considérons uniquement le premier cas $u_iu_j \in R_\ell$. Cette arête a été placée dans R_ℓ dans un certain appel précédent de la règle d'élagage. Cependant, à cause des étapes 5 et 6 exécutée lors de cet appel, nous avons que $u_iu_1 \in B_\ell$ ou $u_ju_1 \in B_\ell$, c'est-à-dire, au moins une de ces deux arêtes ne peut apparaître dans H ; une contradiction.

Cas 2. La no-réponse est donnée à l'étape 3. Alors nous avons $N_G[u_1] \setminus N_G[u_2] \neq \emptyset$ et $N_G[u_2] \setminus$

$N_G[u_1] \neq \emptyset$. Par le Lemme 4.3 i) et iii), $(G_\ell, k, R_\ell, B_\ell)$ n'a pas de solution.

Cas 3. La no-réponse est donnée à l'étape 6. Alors $R_\ell \cap B'_\ell \neq \emptyset$ ou $R'_\ell \cap B_\ell \neq \emptyset$. Par l'étape 4, nous pouvons supposer que $N_G[u_1] \setminus N_G[u_2] = \emptyset$ et que $N_G[u_2] \setminus N_G[u_1] \neq \emptyset$. Pour obtenir une contradiction, nous supposons que $(G_\ell, k, R_\ell, B_\ell)$ possède une solution H. Par le Lemme 4.3 iii), $R'_\ell = \{u_2u_1, \ldots, u_ru_1\} \subseteq E_H$. D'où $R'_\ell \cap B_\ell = \emptyset$, ce qui signifie que $R_\ell \cap B'_\ell \neq \emptyset$.
Soit $\{x_1, \ldots, x_p\} = N_G(u_1) \setminus \{u_1, \ldots, u_r\}$. Nous avons alors que $B'_\ell = \{u_iu_j \mid 2 \leq i < j \leq r\} \cup \{u_1x_1, \ldots, u_1x_p\}$. Par les mêmes arguments que ceux utilisés dans le Cas 1, nous avons que $u_iu_j \notin R_\ell$ pour tout $2 \leq i < j \leq r$. Par le Lemme 4.3 iii), nous avons que E_H, et donc que R_ℓ, ne contient pas les arêtes $u_1x_1, \ldots, u_1x_p$. Nous pouvons conclure que $R_\ell \cap B'_\ell = \emptyset$; une contradiction. □

Lemme 4.12 montre que la règle d'élagage est correcte, c'est-à-dire, nous trouvons soit que $(G, k, \emptyset, \emptyset)$ ne possède pas de solution, ou alors nous pouvons continuer avec l'instance $(G_\ell, k, R_\ell, B_\ell)$ à la place. Supposons que l'on soit dans le dernier cas. Rappelons que $(G_\ell, k, R_\ell, B_\ell)$ n'a pas d'ensemble S comme spécifié à l'étape 1, autrement la règle d'élagage aurait été appliquée une fois de plus.
Pour simplifier la notation, nous écrivons $(G, k, R, B) = (G_\ell, k, R_\ell, B_\ell)$. Nous avons besoin des propriétés suivantes qui sont satisfaites pour toute solution de (G, k, R, B) (si elle existe).

Lemme 4.13. *Toute solution H de (G, k, R, B) satisfait les propriétés suivantes :*
i) le voisin de tout sommet pendant de H possède dans H au moins deux sommets non-pendants dans son voisinage ;
ii) seules les arêtes de G incidentes aux sommets pendants de H peuvent être dans R ou B ;
iii) si un sommet pendant v de H est incident à une arête de R dans G, alors toutes les autres arêtes de G incidentes à v sont dans B.

Démonstration. Pour montrer i), supposons que H soit solution d'une instance (G, k, R, B), tel que H contienne un sommet pendant u adjacent à un sommet v. Si $d_H(v) = 1$, alors H est isomorphe à K_2, ce qui n'est pas possible car (G, k, R, B) n'a pas de solution qui soit un arbre. Ainsi $d_H(v) \geq 2$ et v a au moins un voisin différent de u. Si tous les voisins de v sont pendants, alors H est un arbre ; contradiction. Ainsi, v a au moins un sommet non-pendant. Si v possède un voisin unique non-pendant w, alors par le Lemme 4.2, $G - \{v, w\}$ contient une composante connexe induite par les voisins pendants de v dont les sommets forment avec v et w une clique dans G. En conséquence, nous pouvons appliquer la règle d'élagage sur $S = \{v, w\}$, qui est une contradiction. Les propriétés ii) et iii) découlent de la construction de R et B au cours des étapes 4 et 5 de la règle d'élagage. □

Nous appliquons maintenant de manière exhaustive la règle suivante sur (G, k, R, B). Cette règle est constituée de quatre étapes qui doivent être réalisées dans l'ordre indiqué.

Règle de Réduction de Chemins

1. Trouver un F-triplet $S = \{u_1, u_2, u_3\}$.
2. Définir $R' = \{u_2u_1, u_2u_3, \ldots, u_2u_r\}$
   et $B' = \{x_1u_2, \ldots, x_pu_2\} \cup \{y_1u_2, .., y_qu_2\} \cup \{u_1u_3, \ldots, u_1u_r\} \cup \{u_3u_4, \ldots, u_3u_r\}$
   (observons que l'ensemble $\{u_3u_4, \ldots, u_3u_r\} = \emptyset$ si $r = 3$).
3. Si $R \cap B' \neq \emptyset$ ou $R' \cap B \neq \emptyset$, alors stopper et retourner une no-réponse.
4. Supprimer les sommets $u_2, u_4, \ldots, u_r$ de G. Supprimer toutes les arêtes incidentes à $u_2, u_4, \ldots, u_r$ de R et B. Si $u_1u_3 \in B$, alors supprimer u_1u_3 de B. Ajouter u_1u_3 à R. Ajouter $x_1u_3, \ldots, x_pu_3$ et $y_1u_1, \ldots, y_qu_1$ in G. Mettre ces arêtes dans B.

En appliquant récursivement la règle de réduction de chemin, nous obtenons une séquence d'instances $(G_0, k, R_0, B_0), \ldots, (G_\ell, k, R_\ell, B_\ell)$ du problème Racine Carrée Minimum Etiquetée pour un certain entier $\ell \geq 0$, où $(G_0, k, R_0, B_0) = (G, k, R, B)$ et où $(G_\ell, k, R_\ell, B_\ell)$ est une instance pour laquelle nous avons soit retourné une no-réponse, ou alors pour laquelle il n'existe pas de F-triplet S. Pour $0 \leq i \leq \ell$ nous dénotons les ensembles R' et B' construits lors du $(i+1)$-ème appel à la règle de réduction de chemin à partir des ensembles respectifs R'_i et B'_i.
Nous avons besoin du lemme suivant, que nous utiliserons plus tard à différents endroits.

Lemme 4.14. *Soient $1 \leq i \leq \ell$ et $\{u_1, u_2, u_3\}$ le F-triplet qui a conduit à l'instance (G_i, k, R_i, B_i). Si H_i est une solution de (G_i, k, R_i, B_i), alors $u_1u_3 \in E_{H_i}$ et le graphe H_{i-1}, obtenu à partir de H_i par suppression de l'arête u_1u_3 et par l'adjonction de u_2 et des sommets $u_4, \ldots, u_r$ (si $r \geq 4$) ensembles avec les arêtes $u_2u_1, u_2u_3, \ldots, u_2u_r$, est une solution de $(G_{i-1}, k, R_{i-1}, B_{i-1})$.*

Démonstration. u_1u_3 est une arête de H_i, puisque $u_1u_3 \in R_i$ de par l'étape 4 du dernier appel à la règle de réduction de chemins. Le graphe H_{i-1} n'est pas seulement une racine carrée de G_{i-1} mais aussi une solution de $(G_{i-1}, k, R_{i-1}, B_{i-1})$ pour les raisons suivantes. D'abord, H_{i-1} possède au plus $|V_{G_{i-1}}| - 1 + k$ arêtes. Ensuite, H_{i-1} ne contient aucune arête de B_{i-1} puisque les arêtes ajoutées $u_2u_1, u_2u_3, \ldots, u_2u_r$ sont toutes dans R'_{i-1} et $R'_{i-1} \cap B_{i-1} = \emptyset$. Troisièmement, H_{i-1} contient toutes les arêtes de R_{i-1}, ce qui peut être vu comme suit. Supposons que H_{i-1} ne contienne pas une arête de R_{i-1}. Alors cette arête doit être dans l'ensemble $\{x_1u_2, \ldots, x_pu_2\} \cup \{y_1u_2, .., y_qu_2\} \cup \{u_1u_3, \ldots, u_1u_r\} \cup \{u_3u_4, \ldots, u_3u_r\}$. Cependant, cet ensemble est égal à B'_{i-1} et $R_{i-1} \cap B'_{i-1} = \emptyset$. □

Nous établissons le Lemme 4.15 concernant les vrais jumeaux dans $G_0, \ldots, G_\ell$. Le Lemme 4.16 quant à lui est analogue au Lemme 4.12 pour la règle de réduction de chemins.

Lemme 4.15. *Soit $1 \leq i \leq \ell$ et soit $\{u_1, u_2, u_3\}$ le F-triplet qui a conduit à l'instance (G_i, k, R_i, B_i). Alors tous vrais jumeaux $v, w \in V_{G_i} \setminus \{u_1, u_3\}$ dans G_i sont vrais jumeaux dans G_{i-1}.*

Démonstration. Supposososns que G_i possède deux vrais jumeaux $v, w \in V_{G_i} \setminus \{u_1, u_3\}$ qui ne soient pas des vrais jumeaux dans G_{i-1}. Considérons le F-graphe correspondant qui a conduit à l'instance (G_i, k, R_i, B_i). Puisque v, w sont des vrais jumeaux dans G_{i-1}, le voisinage de v ou w est modifié par la règle de réduction de chemin. Nous pouvons supposer sans perte de généralité que c'est le voisinage de v qui est modifié. Observons que $v \neq u_2$ et que $v \notin \{u_4, \ldots, u_r\}$ si $r \geq 3$, puisque ces sommets ont été supprimés par l'étape 4 de la règle de réduction de chemins lorsque G_i a été construit. De $v \notin \{u_1, u_3\}$, nous avons que $v \in \{x_1, \ldots, x_p\} \cup \{y_1, \ldots, y_q\}$. Par symétrie nous pouvons supposer que $v \in \{x_1, \ldots, x_p\}$. Observons que v est adjacent à u_1 ainsi qu'à u_3 dans G_i. Du fait que le voisinage de chaque x_i est modifié de la même façon (à savoir la suppression de u_2 et l'ajout de u_3), nous avons que $w \notin \{x_1, \ldots, x_p\}$. u et v sont de vrais jumeaux, ils sont donc en particulier voisins. Parce que deux sommets x_i et y_j ne sont pas voisins dans G_i, nous obtenons que $w \notin \{y_1, \ldots, y_q\}$. Nous concluons alors que le voisinage de w n'est pas modifié lorsque la règle de réduction de chemins s'applique. v est voisin de u_1 et u_3 dans G_i et v, w sont de vrais jumeaux dans G_i, implique que w est adjacent à u_1 et u_3 dans G_{i-1}. Cependant, par définition d'un F-graphe, $N_{G_{i-1}}(u_1) \cup N_{G_{i-1}}(u_3) = \{u_2, u_4, \ldots, u_r\}$, et les sommets $u_2, u_4, \ldots, u_r$ ne sont pas dans G_i vu qu'ils ont été supprimés par la règle de réduction de chemins ; une contradiction. □

Lemme 4.16. *L'instance $(G_\ell, k, B_\ell, R_\ell)$ n'admet pas de solution qui soit un arbre, et G_ℓ est 2-connexe. De plus, $(G_\ell, k, R_\ell, B_\ell)$ admet une solution si et seulement si (G_0, k, R_0, B_0) admet une solution. Si la règle de réduction de chemins a retourné une no-réponse pour $(G_\ell, k, R_\ell, B_\ell)$, alors (G_0, k, R_0, B_0) n'a pas de solution.*

Démonstration. Pour $0 \leq i \leq \ell$, nous utilisons la récurrence pour montrer que le graphe G_i est 2-connexe et que (G_i, k, B_i, R_i) n'admet pas de solution qui soit un arbre. De plus, pour tout $1 \leq i \leq \ell$, nous montrons que (G_i, k, R_i, B_i) admet une solution si et seulement si $(G_{i-1}, k, R_{i-1}, B_{i-1})$ admet une solution. Finalement, nous prouvons que si la règle de réduction de chemins a retourné une no-réponse pour $(G_\ell, k, R_\ell, B_\ell)$, alors (G_0, k, R_0, B_0) n'a pas de solution.
Si $i = 0$, alors G_i est 2-connexe et (G_0, k, B_0, R_0) n'a pas de solution qui soit un arbre par le Lemme 4.12. Maintenant supposons que $1 \leq i \leq \ell$. Par notre hypothèse de récurrence, nous pouvons supposer que G_{i-1} est 2-connexe et que $(G_{i-1}, k, B_{i-1}, R_{i-1})$ n'admet pas de solution qui soit un arbre.
Puisque la règle de réduction de chemins s'est appliquée sur $(G_{i-1}, k, R_{i-1}, B_{i-1})$ et qu'elle a conduit à une nouvelle instance (G_i, k, R_i, B_i), le graphe G_{i-1} possède un F-triplet $S = \{u_1, u_2, u_3\}$. Puisque G_{i-1} est 2-connexe, G_i est 2-connexe ; en particulier notons que $p \geq 1$ et $q \geq 1$ par définition d'un F-triplet.
Supposons que H_{i-1} soit une solution pour $(G_{i-1}, k, R_{i-1}, B_{i-1})$. Nous affirmons que H_{i-1} ne contient aucune arête $u_s u_t \in R_{i-1}$ avec $4 \leq s < t \leq r$. Nous montrons cette affirmation par contradiction : soit $u_s u_t \in E_{H_{i-1}} \cap R_{i-1}$ pour $4 \leq s < t \leq r$.
Supposons que $u_s u_t \in R_0$. Nous pouvons appliquer le Lemme 4.13 vu que (G_0, k, R_0, B_0) admet une solution H_0 ; si $i \geq 1$ cela se déduit de l'hypothèse de récurrence. Par le Lemme 4.13 nous avons soit que u_s est un sommet pendant dans H_0 avec u_t comme (unique) voisin, ou alors nous avons l'inverse. Nous pouvons supposer sans perte de généralité que nous sommes en présence du premier cas, c'est-à-dire, u_s est pendant dans H_0 et u_t est son voisin. Notons que $N_{G_0}[u_s] \subseteq N_{G_0}[u_t]$. Nous affirmons que $N_{G_h}[u_s] \subseteq N_{G_h}[u_t]$ pour tout $0 \leq h \leq i-1$. Pour obtenir une contradiction, nous supposons le contraire. Donc à un certain moment u_s se retrouver voisin d'un sommet v non adjacent à u_t pour la première fois dans l'étape 4 d'un certain appel de la règle de réduction de chemins. Soit $S = \{u'_1, u'_2, u'_3\}$ le F-triplet correspondant. Alors nous pouvons supposer sans perte de généralité que soit $u_s \notin \{u'_1, u'_2, u'_3\}$ est adjacent à u'_1 et u'_2 mais pas à $u'_3 = v$, ou alors que $v \notin \{u'_1, u'_2, u'_3\}$ est adjacent à u'_1, u'_2 mais pas à $u'_3 = u_s$. Dans le premier cas, u_t n'est pas dans $\{u'_1, u'_2\}$, mais doit leur être voisin par hypothèse, et ainsi, l'arête $u_t u'_3 = u_t v$ sera ajoutée dans la même étape ; une contradiction. Dans le deuxième cas, puisque u_s est adjacent à u'_1 et u'_2, u_t est également adjacent à u'_1 et u'_2 (à nouveau par hypothèse). Puisque u_t n'est pas supprimé dans cette étape (vu que u_t appartient à G_{i-1}), cela enfreint la définition d'un F-triplet. Nous pouvons conclure que $N_{G_h}[u_s] \subseteq N_{G_h}[u_t]$ pour tout $0 \leq h \leq i-1$.
Nous supposons d'abord que $u_s u_2$ est une arête dans G_0. L'étape 4 de la règle de réduction de chemins ne fait que déplacer une arête $u'_1 u'_3$ d'un ensemble B vers un ensemble R si u'_1 et u'_3 sont des sommets sortants d'un F-triplet. Dans ce cas leurs voisins communs seront supprimés du graphe par définition d'un F-triplet. Du fait que $N_{G_h}[u_s] \subseteq N_{G_h}[u_t]$ pour tout $0 \leq h \leq i-1$, nous obtenons que u_t est un voisin commum de u_2 et de u_s dans G_h pour tout $0 \leq h \leq i-1$; en particulier u_t appartient à G_{i-1}. Ainsi, l'arête $u_s u_2$ ne sera jamais déplacée de B_h vers R_h dans l'étape 4 du $(h+1)$-ème appel de la règle de réduction de chemins pour un certain $0 \leq h \leq i-1$. Si $u_s u_2$ n'est pas une arête dans G_0, alors à un certain point elle le deviendra à cause de l'étape 4 d'un certain appel de la règle de réduction de chemins, que nous pouvons appeler le (h^*+1)-ème appel, pour un certain $0 \leq h^* \leq i-1$. Dans la même étape, $u_s u_2$ sera placé dans l'ensemble B_{h^*}. Alors, à nouveau parce que $N_{G_h}[u_s] \subseteq N_{G_h}[u_t]$ pour tout $0 \leq h \leq i-1$, l'arête $u_s u_2$ ne sera jamais déplacée de l'ensemble B_{h^*} vers un ensemble R_h pour un certain $h^* < h \leq i-1$. Ainsi, dans les deux cas, nous avons que $u_s u_2 \in B_{i-1}$ même si $i \geq 1$. De $u_s u_2 \in R'_{i-1}$ (de l'étape 2 du i-ème appel), nous avons que $R'_{i-1} \cap B_{i-1} \neq \emptyset$. En conséquence, la règle de réduction en chemins ne retournera jamais une no-réponse pour $(G_{i-1}, k, R_{i-1}, B_{i-1})$ dans l'étape 3, et en conséquence l'instance (G_i, k, R_i, B_i) n'existera pas ; une contradiction.

Maintenant supposons que $u_s u_t$ a été placé dans un certain ensemble R_h pour un certain $1 \leq h \leq i-1$. Les propriétés ii) et iii) d'un F-graphe ensembles avec l'étape 4 de la règle de réduction de chemins conduisent à l'observation suivante : si u_s et u_t forment avec un sommet z un triangle, alors $u_s z \in B_h$ ou $u_t z \in B_h$. De plus, dans le cas où $z \in V_{G_{i-1}}$, cette propriété ne peut être invalidée par un quelconque appel ultérieur à la règle de réduction de chemins. Ainsi, si $u_s u_t \in R_{i-1}$, alors $u_s u_2 \in B_{i-1}$ ou $u_t u_2 \in B_{i-1}$, et de même vu que $\{u_s u_2, u_t u_2\} \subseteq R'_{i-1}$, nous obtenons la même contradiction qu'avant. Nous concluons que H_{i-1} ne contient aucune arête $u_s u_t \in R_{i-1}$ pour $4 \leq s < t \leq r$. De plus, par le Lemme 4.5, nous pouvons supposer sans perte de généralité que H_{i-1} ne contient pas d'arête $u_s u_t \notin R_{i-1}$ avec $4 \leq s < t \leq r$; autrement nous pourrions retirer de H_{i-1} une telle arête, et le graphe résultant resterait une solution de $(G_{i-1}, k, R_{i-1}, B_{i-1})$. En conséquence, $u_4, \ldots, u_r$ sont des sommets pendants de H_{i-1}. Cela signifie que le graphe H obtenu à partir de H_{i-1} par suppression des sommets $u_2, u_4, \ldots, u_r$ et par l'ajout des arêtes $u_1 u_3$ est non seulement une racine carrée de G_i avec au plus $|V_{G_i}| - 1 + k$ arêtes, mais est en plus une solution de (G_i, k, R_i, B_i).
Maintenant supposons que H_i soit une solution de (G_i, k, R_i, B_i). Par le Lemme 4.14, le graphe H obtenu à partir de H_i par suppression de l'arête $u_1 u_3$ et par l'ajout de u_2 et des sommets $u_4, \ldots, u_r$ (si $r \geq 4$) ainsi que des arêtes $u_2 u_1, u_2 u_3, \ldots, u_2 u_r$ est une solution de $(G_{i-1}, k, R_{i-1}, B_{i-1})$. Nous observons que H_i ne peut être un arbre, autrement H serait un arbre, ce qui n'est pas possible puisque $(G_{i-1}, k, R_{i-1}, B_{i-1})$ n'admet pas une telle solution par hypothèse de récurrence.
Finalement, supposons que la règle de réduction de chemins a retourné une no-réponse pour $(G_\ell, k, R_\ell, B_\ell)$. Nous devons montrer que (G_0, k, R_0, B_0) n'admet pas de solution. De par les observations précédentes, cela revient à montrer que $(G_\ell, k, R_\ell, B_\ell)$ n'admet pas de solution. La seule étape pour laquelle la règle de réduction de chemins peut retourner une no-réponse est l'étape 3, ce qui signifie que G_ℓ contient un F-triplet $S = \{u_1, u_2, u_3\}$ tel que $R_\ell \cap B'_\ell \neq \emptyset$ ou $R'_\ell \cap B_\ell \neq \emptyset$.
Pour obtenir une contradiction, supposons que $(G_\ell, k, R_\ell, B_\ell)$ admette une solution H. Par le Lemme 4.5, le graphe F présenté dans la Figure 4.4 est un sous-graphe de H tel que $d_H(u_2) = r - 1$, $\{x_1, \ldots, x_p\} = N_H(u_1) \setminus \{u_2\}$ et $\{y_1, \ldots, y_q\} = N_H(u_3) \setminus \{u_2\}$. En conséquence, $R'_\ell = \{u_2 u_1, u_2 u_3, \ldots, u_2 u_r\} \subseteq E_H$, et hence $R'_\ell \cap B_\ell = \emptyset$, et de plus, $E_H \cap B'_\ell = E_H \cap (\{x_1 u_2, \ldots, x_p u_2\} \cup \{y_1 u_2, .., y_q u_2\} \cup \{u_1 u_3, \ldots, u_1 u_r\} \cup \{u_3 u_4, \ldots, u_3 u_r\}) = \emptyset$, qui implique que $R_\ell \cap B'_\ell = \emptyset$; une contradiction. □

Le Lemme 4.16 montre que la règle de réductionde chemins est valide, c'est-à-dire, soit nous trouvons que (G_0, k, R_0, B_0) n'admet pas de solution, ou alors que nous pouvons continuer avec l'instance $(G_\ell, k, R_\ell, B_\ell)$. Supposons que le dernier cas se présente. Rappelons que $(G_\ell, k, R_\ell, B_\ell)$ ne contient pas de F-triplet, autrement nous aurions appliqué la règle de réduction de chemins une fois encore. Rappelons également que R_0 est l'ensemble de sommets de R directement issu après application récursive de la règle d'élagage. Nous écrivon $R^1 = R_0 \cap R_\ell$ et $R^2 = R_\ell \setminus R_0$. À partir de maintenant, pour simplifier la notation, nous écrivons également $(G, k, R, B) = (G_\ell, k, R_\ell, B_\ell)$; notons que $R = R^1 \cup R^2$. Nous avons besoin des propriétés suivantes qui sont satisfaites pour toutes les solutions de (G, k, R, B) (si une existe).
Nous appelons cycle induit C dans un graphe H *semi-pendant* si tous les sommets de C, sauf peut-être un, ne sont voisins que de sommets pendants de H, à l'exception de leurs voisins dans C. De manière similaire, nous appelons chemin induit P dans un graphe H *semi-pendant* si tous les sommets internes de P ne sont adjacents sont adjacents qu'à des sommets pendants de H, à l'exception de leurs voisins dans P.

Lemme 4.17. *Toute solution H de (G, k, R, B) satisfait les propriétés suivantes :*
i) le voisinage de chaque sommet pendant de H possède au moins deux sommets non pendant dans H ;

ii) seules les arêtes de G incidentes à des sommets pendants de H peuvent appartenir à R^1, et si un sommet pendant v de H est incident à une arête de R, alors toutes les autres arêtes de G incidentes à v sont dans B ;
iii) aucune arête de R^2 n'est incidente à un sommet pendant de H ;
iv) la longueur de tout chemin semi-pendant dans H est au plus 5 ;
v) la longueur de tout cycle semi-pendant dans H est au plus 6.

Démonstration. Nous montrons la propriété i) par contradiction. Supposons que H contienne un sommet v qui soit le voisin (unique) d'un sommet pendant u, tel que v a au plus un voisin non pendant dans H. Si tous les voisins de v dans H sont pendants, alors H est un arbre. Cependant, cela contredirait le Lemme 4.16. Supposons alors que v a un unique voisin non-pendant dans H. Rappelons que H est solution de $(G_\ell, k, R_\ell, B_\ell)$. Notons que si v est un sommet sortant du F-triplet correspondant, alors le Lemme 4.14 stipule que $(G_{\ell-1}, k, R_{\ell-1}, B_{\ell-1})$ admet une solution $H_{\ell-1}$ pour laquelle v est un sommet non-pendant qui a au moins un voisin pendant et un unique voisin non pendant. En conséquence, en appliquant le Lemme 4.14 récursivement, nous obtenons que (G_0, k, R_0, B_0) admet une solution H_0 contenant un sommet avec exactement la même propriété. Cela contredit le Lemme 4.13 i). Nous concluons que la propriété i) est satisfaite.
Nous montrons à présent la propriété ii). Par le Lemme 4.13, toute arête de G_0 qui est dans R_0 est incidente à un sommet pendant u de toute solution de (G_0, k, R_0, B_0), tel que les autres arêtes incidentes à u appartiennet à B_0. Nous observons qu'en appliquant la règle de réduction de chemins, u ne sera jamais dans un F-triplet pas plus qu'il ne sera supprimé du graphe, mais u peut être un sommet de type x ou de type y. Ainsi, la règle de réduction de chemins peut modifier les voisins de u mais dans ce cas les nouvelles arêtes incidentes à u seront placées dans B (et y resteront par la suite). En conséquence, u doit également être un sommet pendant de toute solution de $(G, k, R, B)(= (G_\ell, k, R_\ell, B_\ell))$. La propriété ii) est donc satisfaite.
Nous montrons maintenant la propriété iii). Rappelons que nous avons appliqué la règle de réduction de chemins uniquement après avoir appliqué la règle d'élagage de manière exhaustive. Lorsque nous appliquons la règle de réduction de chemins sur un F-triplet $\{u_1, u_2, u_3\}$, alors par la suite u_1 et u_3 ont degré au moins 2 dans toute solution de l'instance résultant, qui peut considéré comme suit. L'arête u_1u_3 est ajoutée à $R^2 \subseteq R$, et ainsi appartient à toute solution. Nous observons également que u_1 est adjacent à x_1 dans G, alors que l'arête u_3x_1 appartient à B. Cela signifie que u_1 ne peut être rendu adjacent à x_1 via un chemin $u_1u_3x_1$ dans H, et en tant que tel u_1 doit avoir au moins un autre voisin dans H. Pour les mêmes raisons, u_3, qui est adjacent à y_1 dans G tandis que $u_1y_1 \in B$, doit avoir un autre voisin dans H autre que u_1. En conséquence, toute arête de R^2 ne peut être incidente à un sommet pendant de H. Nous avons donc montré la propriété iii).
Nous montrons la propriété iv). Soit P un chemin semi-pendant de longueur au moins 6 dans H. Par définition, P est un chemin induit. Ainsi, nous pouvons choisir n'importe quel triplet de sommets consécutifs de P avec la relation des trois sommets u_1, u_2, u_3 du Lemme 4.4. En appliquant ce lemme, nous voyons que G est un F-graphe et alors que nous pouvons appliquer la règle de réduction de chemins une fois de plus ; contradiction. La propriété v) peut être démontrée en utilisant les mêmes arguments. □

Nous aurons besoin du Lemme 4.18 suivant qui vaut dans le cas où une solution de (G, k, R, B) existe.

Lemme 4.18. *Dans toute solution de (G, k, R, B), le nombre de sommets non-pendants est au plus $15k - 14$.*

Démonstration. Supposons que (G, k, R, B) admet une solution H. Soit Z l'ensemble des sommets pendants de H, et soit $H^* = H - Z$. Nous devons montrer que V_{H^*} contient au plus $15k - 14$

sommets. Soit V' l'ensemble des sommets de degré au moins 3 dans H^*, et soit V'' l'ensemble des sommets de degré 2 dans H^*. Par le Lemme 4.17 i) tout sommet de H qui est voisin d'un sommet pendant de H est de degré au moins 2 dans H^*. Ainsi, H^* n'a pas de sommet de degré au plus 1, c'est-à-dire, $V_{H^*} = V' \cup V''$. Puisque H est une solution de (G, k, R, B), nous avons que $|E_H| \leq |V_G| - 1 + k = |V_H| - 1 + k$. Cela signifie que

$$\begin{aligned} |V'| + |V''| - 1 + k &= |V_H| - |Z| - 1 + k \\ &\geq |E_H| - |Z| \\ &= |E_{H^*}| \\ &= \tfrac{1}{2} \textstyle\sum_v d_{H^*}(v) \\ &\geq \tfrac{1}{2}(3|V'| + 2|V''|). \end{aligned}$$

Ainsi, $|V'| \leq 2k-2$. Soit α le nombre de chemins dans H^* qui ont uniquement des sommets internes de degré 2 ; notons que la longueur de tels chemins est au plus 5 d'après le Lemme 4.17 iv). Soit β le nombre de cycles dans H^* qui ont exactement un sommet de degré au moins 3 ; notons que d'après le Lemme 4.17 v) la longueur de tels cycles est au plus 6. Puisque $|E_{H^*}| \leq |V'| + |V''| - 1 + k$, nous avons que $\alpha + \beta \leq 2k - 2 - 1 + k = 3k - 3$ et que $\beta \leq k$. Ainsi, $|V''| \leq 5k + 4((3k-3) - k) = 13k - 12$. En conséquence, H^* a au plus $2k - 2 + 13k - 12 = 15k - 14$ sommets. □

Nous sommes désormais prêts pour notre règle de réduction finale. L'objectif de cette règle est d'être appliquee une fois pour déduire si (G, k, R, B) n'admet pas de solution ou pour dériver une nouvelle instance de taille bornée. Une *partition de vrais jumeaux* d'un ensemble de sommets S d'un graphe G est une partition $S_1, \ldots, S_t$ of S telle que pour tout $u, v \in S$ et tout $1 \leq i \leq t$, u et v sont dans S_i si et seulement si u et v sont de vrais jumeaux dans G. Observons que si S est uniquement constituée de sommets simpliciaux alors il n'y a pas d'arête entre deux sommets qui appartiennent à différents ensembles S_i et S_j.

Règle de Réduction de Sommets Simpliciaux

1. Déterminer l'ensemble S de tous les sommets simpliciaux de G incidents aux arêtes de R^2, et qui ont en plus toutes leurs arêtes incidentes sauf une dans B (à condition d'être incident à une arête de R^1).
2. Si $|V_G \setminus S| > 15k - 14$, alors arrêter et retourner une no-réponse.
3. Construire la partition de vrais jumeaux $S_1, \ldots, S_t$ de S. Soit $X_1, \ldots, X_t$ l'ensemble des sommets respectifs de $S_1, \ldots, S_t$ incidents à une arête de R^1.
4. Si $t > 15k - 14$, alors arrêter et retourner une no-réponse.
5. S'il existe un ensemble X_i tel que les arêtes de R^1 incidentes à un sommet de X_i ont un sommet communs, alors arrêter et retourner une no-réponse.
6. S'il existe un ensemble S_i tel $|S_i \setminus X_i| \geq 15k - 13$ et tel qu'il y ait trois sommets $u \in X_i$, $v \in N_G(u)$ et $x \in S_i \setminus X_i$ avec $uv \in R^1$ et $xv \in B$, alors arrêter et retourner une no-réponse.
7. Pour $i = 1, \ldots, t$, si $|X_i| > 1$, alors prendre arbitrairement $|X_i| - 1$ des sommets de X_i et les supprimer de G ainsi que de S_i. Supprimer également de R et B les arêtes incidentes à ces sommets.
8. Pour $i = 1, \ldots, t$, si $|S_i| > 15k - 13$, alors supprimer de G arbitrairement $|S_i| - 15k + 13$ sommets de $S_i \setminus X_i$, et supprimer également de R et B les arêtes incidentes à ces sommets.

Appliquer la règle de réduction de sommets simpliciaux sur (G, k, R, B) soit conduit à une non-réponse (dans l'étape 2, 4, 5 ou 6) ou à une nouvelle instance $(\hat{G}, k, \hat{R}, \hat{B})$ du problème Racine Carrée Minimum Etiquetée. Nous montrons à présent que si $\hat{G}$ existe, alors sa taille est bornée par une fonction quadratique en k. Nous avons besoin dans un premier temps des deux lemmes suivants.

Lemme 4.19. *Pour $i = 1, \ldots, t$, aucun sommet de $S_i \setminus X_i$ n'est incident à une arête de R.*

Démonstration. Par définition de S_i, aucun sommet de S_i, et donc aucun sommet de $S_i \setminus X_i$, n'est pas incident à une arête de R^2. Par définition de X_i, aucun sommet de $S_i \setminus X_i$ n'est incident à une arête de R^1. Et puisque $R = R^1 \cup R^2$, nous avons montré le Lemme 4.19. □

Pour $x \in V_G$, nous définissons $B(x)$ l'ensemble des arêtes de B incidentes à x.

Lemme 4.20. *$B(x) = B(y)$ pour tout $x, y \in S_i \setminus X_i$.*

Démonstration. Soient $x, y \in S_i \setminus X_i$ et soit $xz \in B$, pour $z \in V_G$. Nous montrons d'abord que $y \neq z$ et nous montrerons ensuite que $yz \in B$.
Pour obtenir une contradiction, supposons que $y = z$. Alors l'arête xy était incluse dans B soit par une application de la règle d'élagage ou alors par une application de la règle de réduction de chemins. Dans les deux cas, xy a également été rendue adjacente à une arête de R. Cette arête pourrait être supprimée par la suite. La suppression d'une arête e de R se produit soit dans l'étape 6 de la règle d'élagage ou alors dans l'étape 4 de la règle de réduction de chemins. Cependant, ces deux règles ajoutent une nouvelle arête e' dans R qui est adjacente à toutes les arêtes qui étaient précédemment adjacentes à e et qui n'étaient pas supprimées par les deux règles. Ainsi, xy est toujours adjacente à une arête de R dans G. En d'autres termes, x ou y est incident à une arête de R dans G. Puisque x et y appartiennent à $S_i \setminus X_i$, cette situation n'est pas possible par le Lemme 4.19. D'où $y \neq z$.
Pour montrer que l'arête $yz \in B$, nous utilisons à nouveau l'observation que lorsque la règle de l'élagage ou la règle de réduction de chemins supprime une arête $e \in R$, la règle ajoute une nouvelle arête e' dans R telle que e' est adjacente à toutes les arêtes uv qui étaient auparavant adjacentes à e et qui n'étaient pas supprimées par la règle. Dans ce cas nous pouvons faire l'observation que si un sommet u est une extrémité de e qui n'est pas supprimée par la règle, alors u est un sommet terminal de e'. Puisque les sommets appartenant à $S_i \setminus X_i$ ne sont incidents à aucune arête de R par le Lemme 4.19, nous avons que z était incident à une arête de R après application de la règle d'élagage ou la règle de réduction de chemins qui avait ajouté l'arête xz dans B. Nous observons également qu'une arête de B n'est supprimée de B que si une de ses extrémités est supprimée à moins qu'elle ne soit ajoutée à R par la règle de réduction de chemins. Cela signifie que nous pouvons affirmer ce qui suit.
Supposons premièrement que l'arête xz ait été ajoutée à B par une application de la règle d'élagage. Si y était adjacent à z lorsque la règle s'est appliquée, alors l'arête yz appartenait également à B par la définition de cette règle. Si l'adjacence de y à z a été réalisée par la suite par la règle de réduction de chemins, alors $yz \in B$ par la définition de la règle de réduction de chemins.
Supposons désormais que xz ait été ajoutée à B par une application de la règle de réduction de chemins. Par définition de cette règle, x et z n'étaient pas voisins jusqu'à présent. Supposons que $yz \notin B$. Alors xy, yz sont des arêtes du graphe d'entrée original du problème Racine Carrée Minimum. Du fait que xz n'était pas une telle arête, x et y sont devenus vrais jumeaux uniquement à cause d'une application de la règle de réduction de chemins. Alors, d'après le Lemme 4.15, x ou y doit être un sommet sortant d'un certain F-triplet, c'est-à-dire, au moins un de ces deux sommets doit être incident à une arête de R. Alors il y a une arête de R incidente à au moins un de ces deux sommets après application exhaustive de la règle de réduction de chemins. x et y sont donc

dans $S_i \setminus X_i$, ce qui contredit le Lemme 4.19. En conséquence, $yz \in B$, ce qui conclut la preuve du Lemme 4.20. □

Nous prouvons notre dernier lemme ; en particulier notons que si $\hat{G}$ existe alors sa taille est bornée par une fonction quadratique en k.

Lemme 4.21. *Si la règle de réduction de sommets simpliciaux retourne une no-réponse pour (G, k, R, B), alors (G, k, R, B) n'admet pas de solution. Dans le cas contraire, la nouvelle instance $(\hat{G}, k, \hat{R}, \hat{B})$ admet une solution si et seulement si (G, k, R, B) admet une solution. De plus, $\hat{G}$ possède au plus $(15k - 14)(15k - 12)$ sommets.*

Démonstration. Nous commençons par montrer que (G, k, R, B) n'admet pas de solution si la règle de réduction de sommets simpliciaux retourne une no-réponse pour (G, k, R, B). Cette no-réponse peut être retournée dans l'étape 2, 4, 5 ou 6. Nous discutons chaque étape dans des cas différents.

Cas 1. La no-réponse est donnée à l'étape 2. Supposons que (G, k, R, B) admette une solution H. Nous allons montrer que $|V_G \setminus S| \leq 15k - 14$, ce qui signifie que la no-réponse retournée est correcte si $|V_G \setminus S| > 15k - 14$.
Soit Z l'ensemble des sommets pendants de H, et soit $H^* = H - Z$. Par l'Observation 4.1 i), les sommets de Z sont simpliciaux dans G. Alors, d'après le Lemme 4.17 ii) et iii), nous avons que $Z \subseteq S$. Ainsi, $|V_G \setminus S| = |V_G| - |S| = |V_H| - |S| \leq |V_H| - |Z| = |V_{H^*}| \leq 15k - 14$, la dernière inégalité provenant du Lemme 4.18.

Cas 2. La no-réponse est donnée à l'étape 4. Supposons que (G, k, R, B) admette une solution H. Nous allons montrer que $t \leq 15k - 14$, ce qui signifie que la no-réponse est correcte si $t > 15k - 14$. Soit H^* le graphe obtenu à partir de H par suppression de tous les sommets pendants de H. Alors d'après le Lemme 4.18, $|V_{H^*}| \leq 15k - 14$. Si un ensemble S_i contient un sommet pendant u de H, alors u est adjacent à un sommet v de H^*. Alors, par l'Observation 4.1 ii), v n'est pas voisin de sommets pendants de H dans aucun S_j, $j \neq i$. Dans le cas contraire S_i est constitué de sommets non-pendants de H, c'est-à-dire, des sommets de H^* ; étant non vide S_i contient au moins un sommet de H^*. Nous concluons qu'à chaque ensemble de la partition de vrais jumeaux de S correspond au moins un sommet unique de H^*. Si leur nombre total t satisfait $t > 15k - 14$, cela implique que $|V_{H^*}| > 15k - 14$; une contradiction. Ainsi, $t \leq 15k - 14$, ce' que nous devions montrer.

Cas 3. La no-réponse est donné à l'étape 5. Supposons que (G, k, R, B) admette une solution H. Nous allons montrer que les arêtes de R^1 incidentes à un ensemble X_i ont une extrémité commune, pour $i = 1, \ldots, t$, ce qui signifie que retourner une no-réponse est correct si ce n'est pas le cas.
Pour obtenir une contradiction, supposons qu'un certain ensemble X_i contienne deux sommets u et v qui soient incidents aux arêtes $uu', vv' \in R^1$ avec $u' \neq v'$. D'après le Lemme 4.17 ii), nous avons que les arêtes uu' et vv' sont incidentes à des sommets pendants de H. Par l'Observation 4.1 iii), ces sommets pendants ne sont pas voisins dans G. Cependant, d'après la définition de S_i nous pouvons déduire que u, v, u', v' sont tous adjacents deux à deux (vrais jumeaux) ; une contradiction. Cela complète le Cas 3.

Cas 4. La no-réponse est donnée à l'étape 6. Il existe alors un ensemble S_i tel que $|S_i \setminus X_i| \geq 15k - 13$ et tel qu'il y ait trois sommets $u \in X_i$, $v \in N_G(u)$ et $x \in S_i \setminus X_i$ avec $uv \in R^1$ et $xv \in B$. Pour obtenir une contradiction, supposons que (G, k, R, B) admette une solution H.
Par le Lemme 4.18, H contient au plus $15k - 14$ sommets non-pendants. Puisque $|S_i \setminus X_i| \geq 15k - 13$, cela signifie qu'au moins un sommet $y \in S_i \setminus X_i$ est un sommet pendant de H. Egalement, $u \in X_i$

est un sommet pendant de H avec v comme unique voisin, car l'arête uv appartient à R^1 et toutes les arêtes incidentes à u appartiennent à B par définition de S. Si $y = x$, alors v n'est pas voisin de y dans H, car $xv \in B$. Si $y \neq x$, alors v n'est pas voisin de y dans H, parce que $xv \in B$ et $B(x) = B(y)$ (du au Lemme 4.20) implique $yv \in B$. Nous pouvons conclure que u et y sont des sommets pendants dans H adjacents à des sommets différents. Cependant, par l'Observation 4.1 iii) nous avons que u et y ne sont pas adjacents dans G. C'est une contradiction, parce que u et y sont de vrais jumeaux dans G par définition de S_i. Cela complète le Cas 4.

À partir de maintenant supposons que la règle de réduction de sommets simpliciaux n'a pas retourné de no-réponse après application de l'étape 6. Soit (G', k, R', B') l'instance créée après avoir appliqué l'étape 7 à un certain ensemble $X_i = \{x_1, \ldots, x_\ell\}$ avec $\ell \geq 2$, c'est-à-dire, G' est le graphe obtenu à partir de G par suppression des sommets $x_2, \ldots, x_\ell$, tandis que R' et B' sont les ensembles obtenus respectivement à partir de R et B par suppression de leurs arêtes incidentes à $x_2, \ldots, x_\ell$. Nous affirmons que (G', k, R', B') admet une solution si et seulement si (G, k, R, B) admet une solution. Avant de montrer cette affirmation, observons d'abord que dans toute solution H de (G, k, R, B) les sommets $x_1, \ldots, x_\ell$ sont des sommets pendants dans H. En effet, $x_1, \ldots, x_\ell$ sont incidents à une arête unique de R^1, alors que toutes les autres arêtes qui leur sont incidentes appartiennent à B. De plus, $x_1, \ldots, x_\ell$ ont un voisin commun (unique) dans H, puisque dans le cas contraire l'étape 5 aurait retourné une no-réponse. Dénotons v ce voisin commun. De manière similaire, x_1 est un sommet pendant qui possède v comme voisin (unique) dans toute solution H' de (G', k, R', B').
Supposons dans un premier temps que (G', k, R', B') admette une solution H'. Alors le graphe obtenu à partir de H' par adjonction des sommets $x_2, \ldots, x_\ell$ et des arêtes $x_2v, \ldots, x_\ell v$ est une racine carrée de G par le Lemme 4.6 i). Par définition de R', B' et l'ensemble X_i (dont tous les sommets sont incidents à une arête de $R^1 \subseteq R$ et à des arêtes de B), H' est également une solution de (G, k, R, B).
Supposons à présent que (G, k, R, B) admette une solution H. Alors le graphe obtenu à partir de H par suppression des sommets $x_2, \ldots, x_\ell$ est une racine carrée de G' par le Lemme 4.6 ii). Par définition de R' et B', ce graphe est alors également une solution de (G', k, R', B').

Nous dénotons à nouveau (G, k, R, B) l'instance résultant de l'étape 7 et nous observons que tout ensemble X_i contient à présent au plus un sommet. Il reste maintenant à considérer l'étape 8. Soit (G', k, R', B') l'instance créée après application de l'étape 8 sur un certain ensemble S_i avec $|S_i| > 15k - 13$, c'est-à-dire, G' est le graphe obtenu à partir de G après suppression de $S_i \setminus X_i$ d'un ensemble T de $|S_i| - 15k + 13 \geq 1$ sommets choisis arbitrairement (notons que ce cas est possible puisque $|X_i| \leq 1$), tandis que R' et B' sont les ensembles obtenus respectivement à partir de R et B par suppression des arêtes incidentes aux sommets de T. Nous affirmons que (G', k, R', B') admet une solution si et seulement si (G, k, R, B) admet une solution.
Supposons premièrement que (G', k, R', B') admette une solution H'. Puisque nous ne pouvions appliquer ni la règle d'élagage ni la règle de réduction de chemins pour (G, k, R, B), nous ne pouvons pas non plus appliquer ces règles pour (G', k, R', B'). En utilisant les mêmes arguments que nous avons appliqués pour (G, k, R, B) dans la preuve du Lemme 4.18, nous avons que H' contient au plus $15k - 14$ sommets non-pendants. Notons que H' contient au moins $15k - 13$ sommets, qui sont tous dans S_i. Ainsi, H' possède au moins un sommet pendant x qui appartient à S_i. Soit v le voisin (unique) de x dans H'. Alors d'après le Lemme 4.6 i), le graphe H obtenu à partir de H' par adjonction des sommets de T et de leurs arêtes incidentes à v est une racine carrée de G. Nous affirmons que H est également une solution de (G, k, R, B). Puisque les sommets de $T \subseteq S_i \setminus X_i$ ne sont pas incidents aux arêtes de R d'après le Lemme 4.19, nous devons montrer qu'aucune des $|T|$ arêtes que nous avons ajouté pour obtenir H n'appartient à B. Si $x \in S_i \setminus X_i$, alors $xv \notin B$ et

parce que $B(x) = B(y)$ pour tout $y \in S_i \setminus X_i$, nous avons que l'arête yv n'appartient pas à B pour tout $y \in T$. Supposons que $x \in X_i$. Rappelons que $|X_i| \leq 1$ après l'étape 7. $|S_i| > 15k - 13$ après l'étape 7, implique que $|S_i \setminus X_i| \geq 15k - 13$. Alors $yv \notin B$ pour tout $y \in S_i \setminus X_i$ vu que dans le cas contraire l'algorithme aurait retourné une no-réponse à l'étape 6.
Supposons désormais que (G, k, R, B) admette une solution H. D'après le Lemme 4.18, le graphe H contient au plus $15k - 14$ sommets non-pendants. Ainsi, H possède au moins $|S_i| - 15k + 14 \geq 15k - 12 - 15k + 14 = 2$ sommets pendants. Nous savons que les sommets appartenant à $S_i \setminus X_i$ sont de vrais jumeaux qui ne sont pas incidents aux arêtes de R, et que $B(x) = B(y)$ pour tout $x, y \in S_i \setminus X_i$. Nous pouvons supposer en conséquence, sans perte de généralité, que les sommets de T font parti des sommets pendants de H. Si $X_i = \{x\} \neq \emptyset$, alors x est un sommet pendant dans H incident à une unique arête $xv \in R^1$. Par l'Observation 4.1, tous les sommets pendants de H qui sont dans S_i sont voisins de v dans H. Alors d'après le Lemme 4.6 ii), le graphe obtenu à partir de H par suppression des sommets de T est une racine carrée de G'. Par définition de R' et B', ce graphe obtenu est également une solution de (G', k, R', B'). Si $X_i = \emptyset$, alors tous les sommets pendants de H qui sont dans S_i sont voisins d'un certain sommet v dans H par l'Observation 4.1. Alors d'après le Lemme 4.6 (ii), le graphe obtenu à partir de H par suppression des sommets de T est une racine carrée de G'. Par définition de R' et B', ce graphe est également solution de (G', k, R', B').

D'après les observations précédentes il vient que l'instance $(\hat{G}, k, \hat{R}, \hat{B})$ obtenue après l'étape 8 admet une solution si et seulement si (G, k, R, B) admet une solution. Pour achever cette preuve, nous devons montrer que $\hat{G}$ possède au plus $(15k-14)(15k-12)$ sommets. Chaque ensemble S_i possède au plus $15k-13$ sommet du fait de l'étape 8, et nous avons également que $t \leq 15k-14$ du fait de l'étape 4. Ainsi $|S| \leq (15k-14)(15k-13)$. Puisque le nombre de sommets dans $V_G \setminus S$ est au plus $15k-14$ du fait de l'étape 2, nous obtenons que $|V_{\hat{G}}| \leq (15k-14)(15k-13)+15k-14 = (15k-14)(15k-12)$, ce qu'il fallait démontrer. □

Résolution de la Version Etiquetée

Soit n et m les nombres respectifs de sommets et d'arêtes du graphe G de l'instance originelle (G, k) du problème Racine Carrée Minimum. Pour compléter la preuve du Théorème 4.11, nous notons d'abord que les règles d'élagage et de réduction de chemins sont appliquées au plus n fois pour construire l'instance $(\hat{G}, k, \hat{R}, \hat{B})$. Chaque application de la règle d'élagage peut être exécutée en temps $O(n^2m)$ et chaque application de la règle de réduction de chemins peut être exécutée en temps $O(n^3m)$. Finalement, la règle de réduction de sommets simpliciaux peut être exécutée en temps $O(nm)$. Ainsi, notre algorithme de réduction de noyau s'exécute en temps $O(n^4m)$, et il reste à résoudre l'instance réduite obtenue $(\hat{G}, k, \hat{R}, \hat{B})$. $\hat{G}$ possède au plus $(15k - 14)(15k - 12)$ sommets, donc $\hat{G}$ possède au plus $\frac{1}{2}(15k - 14)(15k - 12)((15k - 14)(15k - 12) - 1) = O(k^4)$ arêtes. En conséquence, nous pouvons résoudre le problème Racine Carrée Minimum Étiquetée pour l'instance $(\hat{G}, k, \hat{R}, \hat{B})$ en temps $2^{O(k^4)}$; il suffit de considérer tous les sous-ensembles d'arêtes $\hat{G}$ qui ont une taille au plus $|V_{\hat{G}}| - 1 + k$ et d'utiliser la méthode brute force. Finalement, nous concluons que le temps d'exécution total de notre algorithme est $2^{O(k^4)} + O(n^4m)$.

4.5 Racine Carrée Maximum

Rappelons que le problème Racine Carrée Maximum est de tester si un graphe donné G avec m arêtes admet une racine carrée avec au moins $m - k$ arêtes, pour un certain entier $k \in \mathbb{N}$ donné. Rappelons que ce problème est équivalent à décider si un graphe G admet une racine carrée qui

peut être obtenue de lui-même par suppression d'au plus k arêtes. Nous montrons que ce problème est soluble à paramètre fixé k. Nous présentons également un algorithme exact exponentiel pour résoudre le problème Racine Carrée Maximum.
Les deux algorithmes sont basés sur l'observation que pour construire une racine carrée H à partir d'un graphe donné G, nous devons au minimum supprimer une arête de toute paire d'arêtes incidentes qui n'appartiennent pas à un triangle dans G. Nous construisons en conséquence un graphe auxiliaire $\mathcal{P}(G)$ avec ensemble de sommets E_G et une arête entre deux sommets e_1 et e_2 sont incidentes dans G. Observons que $\mathcal{P}(G)$ est un sous-graphe couvrant du *graphe adjoint* de G, traduction de l'anglais *line graphe*. Nous montrons le lemme suivant.

Lemme 4.22. *Soit H un sous-graphe couvrant de G. Alors H est une racine carrée de G si et seulement si E_H est un ensemble stable de $\mathcal{P}(G)$, et toute paire de sommets voisins de G sont à distance au plus 2 dans H.*

Démonstration. Supposons dans un premier temps que H est une racine carrée de G. Par définition, toute paire de sommets voisins dans G sont à distance au plus 2 dans H. Pour montrer que E_H est un ensemble stable dans $\mathcal{P}(G)$, supposons que deux arêtes $e_1, e_2 \in E_H$ soient des sommets adjacents dans $\mathcal{P}(G)$. Alors $e_1 = xy$ et $e_2 = yz$ pour trois sommets distincts $x, y, z \in V_G$, avec $xz \notin E_G$. Cela signifie que x et z sont à distance 2 dans H, impliquant que $xz \in E_G$, qui est une contradiction.
Réciproquement, supposons que E_H soit un ensemble stable de $\mathcal{P}(G)$ et que chaque paire de deux sommets voisins dans G soient à distance au plus 2 dans H. Pour montrer que H est une racine carrée de G, il suffit de montrer que toute paire de sommets non-adjacents dans G sont à distance au moins 3 dans H. Soient u et v deux sommets non-adjacents dans G qui soient à distance au plus 2 dans H. Alors il existe un sommet $z \notin \{u, v\}$ tel que $uz, vz \in E_H$. Alors $e_1 = uz$ et $e_2 = vz$ sont voisins dans $\mathcal{P}(G)$, ce qui contredit que E_H est stable dans $\mathcal{P}(G)$. □

Nous utilisons le Lemme 4.22 pour montrer les Propositions 4.23 et 4.24. Ici, nous utilisons la notation O^* pour supprimer tout facteurs polynomiaux en n et m. Une *couverture de sommets* est un sous-ensemble $U \subseteq V$ tel que toute arête soit incidente à au moins un sommet de U. Le problème Couverture de Sommets consiste à décider si un graphe donné possède une couverture de sommets de taille au plus p, pour un certain entier p donné. Dans la Proposition 4.23 nous montrons l'existence d'un algorithme qui s'exécute en temps $O^*(2^k)$ pour décider si un graphe donné G admet une racine carrée H telle que $|E_G \setminus E_H| \leq k$.

Proposition 4.23. *Le problème* Racine Carrée Maximum *peut être résolu en temps $O^*(2^k)$.*

Démonstration. Soit G un graphe avec n sommets et m arêtes, et soit $k \geq 0$ un entier. D'après le Lemme 4.22, il suffit de vérifier si $\mathcal{P}(G)$ possède une couverture de sommets U de taille au plus k telle que $H_U = (V_G, E_G \setminus U)$ soit une racine carrée G. Toutes les couvertures de sommets de taille au plus k d'un graphe peuvent être énumérée en adaptant un algorithme de branchement standard pour le problème Couverture de Sommets qui s'exécute en temps $O^*(2^k)$ [65]. $\mathcal{P}(G)$ peut être calculé en temps $O(m^2)$, et vérifier si le graphe H_U est une racine carrée de G peut être réalisé en temps $O(nm)$. En conséquence, notre algorithme s'exécute en totalité en temps $O^*(2^k)$. □

Observons que le problème Racine Carrée Maximum admet un noyau linéaire pour les graphes connexes. Ce résultat est une conséquence immédiate des résultats de Aingworth, Motwani et Harary [3], qui ont montré que si H est une racine carrée d'un graphe connexe à n sommets $G \neq K_n$, alors $|E_G \setminus E_H| \geq n-2$. En conséquence, $n \leq k+2$ pour toute yes-instance (G, k) du problème Racine Carrée Maximum avec $G \neq K_n$ (trivialement, K_n est lui-même une de ses racines carrées). Notons que ce noyau ne conduit pas à un temps d'exécution plus rapide que $O^*(2^k)$.

Dans la Proposition 4.24 nous présentons notre algorithme exact, qui non seulement résoud le problème de décision mais calcule si elle existe une racine carrée d'un graphe donné qui contient un nombre maximum d'arêtes.

Proposition 4.24. *Le problème* Racine Carrée Maximum *peut être résolu en temps* $O^*(3^{m/3})$ *sur les graphes à* m *arêtes.*

Démonstration. Soit G un graphe à n sommets et m arêtes, et soit un entier $k \geq 0$. Nous calculons le graphe auxiliaire $\mathcal{P}(G)$, nous énumérons tous les ensembles stables maximaux I de $\mathcal{P}(G)$, et nous vérifions pour chacun d'entre eux $I \subseteq E_G$ si G est le carré du graphe $H_I = (V_G, I)$. Parmi les graphes H_I qui sont racines carrées de G, nous retournons celui qui contient le plus grand nombre d'arêtes ; si aucun tel graphe H_I n'a été trouvé, alors G n'admet pas de racine carrée. La validité de l'algorithme découle immédiatement du Lemme 4.22. Rappelons que $\mathcal{P}(G)$ peut être calculé en temps $O(m^2)$. Tous les ensembles stables maximaux d'un graphe à m sommets $\mathcal{P}(G)$ peuvent être énumérés en temps $O^*(3^{m/3})$ en utilisant l'algorithme à délai polynomial de Tsukiyama et al. [182], puisque $\mathcal{P}(G)$ contient au plus $3^{m/3}$ ensemble stables maximaux [159]. Finalement, nous rappelons que pour chaque ensemble stable maximal I, nous pouvons vérifier en temps $O(nm)$ si $(H_I)^2 = G$. En conséquence, le temps d'exécution total de notre algorithme est de $O^*(3^{m/3})$. □

4.6 Conclusion

Dans ce Chapitre, nous nous sommes intéressés à la résolution paramétrée des problèmes Racine Carrée Minimum et Racine Carrée Maximum. Nous avons montré en Section 4.4 et 4.5 que pour les paramètres définis, ces deux problèmes sont dans la classe **FPT**.
Pour montrer notre premier résultat paramétré, nous avons construit un noyau généralisé quadratique pour le problème Racine Carrée Minimum. Nous pensons qu'il existe également un noyau quadratique pour ce problème en utilisant la même réduction. Cependant, montrer cela semble être encore plus technique et amène également à un graphe qui contient plus que $(15k - 14)(15k - 12)$ sommets.

Peut-on montrer l'existence d'un noyau quadratique pour le problème Racine Carrée Minimum *?*

Également, il est à noter que notre noyau généralisé pour le problème Racine Carrée Minimum n'implique pas un noyau pour le problème Racine Carrée Minimum Étiquetée, parce que nos règles de réduction requièrent que l'instance originelle ne soit pas étiquetée. Aussi, nous soulevons une question sur la résolution à paramètre fixe du problème plus général de Racine Carrée Minimum Étiquetée.

Le problème Racine Carrée Minimum Étiquetée *est-il un problème* FPT *?*

Les paramètres que nous avons choisi pour l'étude de la résolution à paramètre fixe du problème Racine Carrée mesurent l'éloignement en nombre d'arêtes d'une racine carrée à un arbre ou au graphe de l'entrée. Nous posons alors la question du choix d'autres paramètres.

Y a-t-il d'autres paramètres pour lesquels une autre variante du problème Racine Carrée *soit* FPT *?*

Nous avons construit en Section 4.3 un algorithme pour décider si un graphe de degré maximum $\Delta(G) \leq 6$ admet une racine carrée. Si une racine carrée existe, alors notre algorithme détermine une racine carrée minimum. Nous proposons en Section 4.5 un algorithme exact exponentiel en temps $O^*(3^{m/3})$ pour déterminer une racine carrée maximum d'un graphe. Aussi nos prochaines questions relèvent de la résolution exacte de nos problèmes.

Est-il possible de construire un algorithme exact pour le problème Racine Carrée Minimum *qui soit meilleur que l'algorithme trivial ?*

Peut-on construire un algorithme en temps $O^*(c^n)$, $c \in \mathbb{R}$ *pour résoudre la version décisionnelle du problème* Racine Carrée *sur les graphes en général ?*

Nous posons aussi la question de l'existence d'un algorithme en temps sous-exponentiel en m pour le problème Racine Carrée Minimum.

Peut-on construire un algorithme en temps $O^*(c^{o(m)})$, $c \in \mathbb{R}$ *pour résoudre la version optimisation du problème* Racine Carrée Minimum *?*

Nous avons construit un algorithme en temps polynomial pour décider l'existence d'une racine carrée d'un graphe de degré maximum $\Delta(G) \leq 6$. Cependant, nous ne savons pas si le problème Racine Carrée demeure dans $\mathbf{P}$ si $\Delta(G) \leq 7$.

Existe-t-il une constante $c \in \mathbb{N}_{\geq 7}$ *pour laquelle le problème* Racine Carrée *devient* NP*-complet pour un graphe* G *satisfaisant* $\Delta(G) \leq c$*.*

La question précédente relève de la complexité du problème Racine Carrée restreint à certaines classes de graphes. Il semble raisonnable de penser qu'il existe un algorithme en temps $O^*(n^{\log n})$ pour déterminer une racine carrée d'un graphe *split*. Cependant, nous ne savons pas si ce problème demeure NP-difficile lorsqu'on se restreint à cette classe de graphes.

Quelle est la complexité du problème Racine Carrée *lorsqu'on se restreint aux graphes splits ?*

Chapitre 5

Algorithmes pour d'Autres Problèmes de Graphes

Sommaire

5.1 Introduction

Les problèmes sur des graphes préalablement coloriés sont déjà bien étudiés. Un des plus célèbres problèmes de cette catégorie est le problème de Motif de Graphe, traduit de l'anglais *graph motif*, introduit en 1994 par McMorris et al. [153].

> Motif de Graphe
> *Version* : *Décision*
>
> **Entrée** : $G = (V, E)$, φ une coloration de V, et M un multi-ensemble de couleurs : chaque couleur pouvant apparaître plusieurs fois dans M
> **Sortie** : M est-il un motif connexe de G, c'est-à-dire existe-t-il un sous-ensemble $S \subseteq V$ connexe de sommets dont les couleurs soient en bijection avec M ?

MOTIF DE GRAPHE
VERSION : *DÉCISION*

Entrée : $G = (V, E)$, $\varphi : V \to \{1, \ldots, k\}$, et $\vec{M} \in (\mathbb{N}_{\leq n})^k$ un vecteur de multiplicité des couleurs
Sortie : Existe-t-il un sous-ensemble $S \subseteq V$ connexe tel que chaque couleur de φ apparaît dans S autant de fois que sa multiplicité donnée dans $\vec{M}$?

Fellows et al. ont montré que le problème de MOTIF DE GRAPHE est NP-complet, même lorsque M est un ensemble simple et si G est un arbre avec $\Delta(G) \leq 3$ [79]. Le cas particulier lorsque M est un ensemble simple, donc si $\vec{M} \in (\{0, 1\})^k$, est un problème appelé MOTIF MULTICOLOR, traduit de l'anglais *Colorful Motif* [7]. Fellows et al. ont également montré que le problème MOTIF DE GRAPHE demeure NP-complet, même lorsque M ne contient que deux couleurs et que G est biparti avec $\Delta(G) \leq 4$ [79]. De nombreuses variantes de ce problème ont été étudiées. Typiquement, les contributions pour ces problèmes sont des classifications de complexité ou des algorithmes paramétrés [19, 79, 80, 103, 133, 168].

À notre connaissance, les seuls résultats en faveur de l'ALGORITHMIQUE EXACTE EXPONENTIELLE qui concernent des variantes du problème de MOTIF DE GRAPHE sont ceux proposés par Dondi, Fertin et Vialette en 2011 [62]. Ces auteurs proposent deux algorithmes pour résoudre le problème MOTIF MAXIMUM sur les arbres. Le premier algorithme s'exécute en temps $O^*(1.6181^n)$, tandis que le deuxième s'exécute en temps $O^*(1.3248^n)$ à la condition que M soit un ensemble simple.

MOTIF MAXIMUM
VERSION : *OPTIMISATION*

Entrée : $G = (V, E)$, φ coloration de V, M un multi-ensemble de couleurs
Sortie : Déterminer le plus grand sous-multi-ensemble $M' \subseteq M$ de couleurs pour lequel M' est un motif connexe dans G

5.1.1 Nos Résultats

Dans le chapitre 2, nous nous sommes intéressés aux problèmes de coloration. Pour un graphe G, l'objectif était alors de colorier les sommets du graphe, c'est-à-dire d'attribuer à chaque sommet $v \in V$ une couleur particulière. Dans les Sections 5.2 et 5.3 de ce Chapitre, nous étudions des problèmes différents, mais qui ont toujours trait aux colorations de graphes. Notre objectif ne sera pas de colorier le graphe, puisque pour ces problèmes le graphe d'entrée G est déjà colorié, c'est-à-dire qu'une coloration φ des sommets V du graphe fait déjà partie de l'entrée. Ainsi donc pour nos problèmes, une couleur est déjà attribuée à chaque sommet du graphe. Il serait tout à fait intéressant d'étudier ces problèmes en ayant connaissance de certaines propriétés de la coloration φ de l'entrée. Par exemple, si nous savions préalablement que les classes de couleurs sont toutes des ensembles stables. Toutefois, pour nos problèmes considérés, notre étude porte sur toutes les colorations possibles de l'entrée, il n'y a donc aucune restriction à la coloration φ de l'entrée.
Nous avons vu tout au fil de cette Thèse que l'efficacité d'un algorithme dépend de son temps d'exécution. Un autre paramètre essentiel à l'efficacité d'un algorithme sont les contraintes d'espace mémoire disponible. Dans la Section 5.4, nous nous intéresserons à un problème relié au problème de domination dans un graphe. Pour ce problème, nous construirons deux algorithmes de résolution

différents pour le même problème étudié. La différence du temps d'exécution de ces algorithmes réside dans la quantité polynomiale ou exponentielle de mémoire disponible.

Dans ce Chapitre, nous établissons entre autres les résultats principaux suivants :

- ◇ Nous étudions dans la Section 5.2 la résolution exacte d'une variante du problème de Motif de Graphe. Le problème que nous étudions est le problème d'Ensemble Tropical Connexe. Nous montrons dans un premier temps que ce problème n'est pas dans la classe **SUBEXP** lorsque le graphe d'entrée est un arbre. Puis nous proposons trois algorithmes de résolution exacte pour ce problème. En utilisant la technique de *Balancement d'Algorithmes*, nous construisons un algorithme en temps $O^*(1.5359^n)$ pour le problème Ensemble Tropical Connexe.
- ◇ Dans la Section 5.3, nous étudions la restriction aux arbres du problème Ensemble Tropical Connexe. Lorsqu'un problème n'est pas soluble en temps sous-exponentiel, la question se pose alors de la complexité algorithmique que l'on est en droit d'attendre. Pour un tel problème, existe-t-il un algorithme de résolution en temps $O^*(c^n)$, pour une certaine constante $1 < c < 2$? Dans un premier temps, nous construisons un algorithme qui s'exécute en temps optimal $O^*(3^{n/3})$ pour énumérer les ensembles tropicaux connexes minimaux d'un arbre. Cet algorithme pourrait facilement être appliqué pour le problème d'optimisation d'Ensemble Tropical Connexe. Toutefois, nous améliorons ce résultat pour ce problème en construisant un algorithme de résolution exacte en temps $O^*(1.2721^n)$. Les Sections 5.2 et 5.3 sont l'objet d'une publication acceptée et à paraître à la conférence IPEC 2014 (*9th International Symposium on Parameterized and Exact Computation*).
- ◇ Finalement, dans la Section 5.4, nous étudions le problème de Domination Romaine Faible. Pour ce problème, nous proposons deux algorithmes de résolution qui s'exécutent respectivement en temps $O^*(2^n)$ avec espace exponentiel et en temps $O^*(2.23^n)$ avec espace polynomial. Les résultats présentés dans cette Section ont fait l'objet d'une publication en 2013 [33].

5.2 Ensemble Tropical Connexe

5.2.1 Introduction

Ensemble Tropical Connexe. Soit $G = (V, E)$ un graphe, φ une coloration de V, et C l'ensemble des couleurs de φ. Un sous-ensemble $S \subseteq V$ est dit *Tropical* si toutes les couleurs de C apparaissent parmi les sommets de S, c'est-à-dire $\forall\, c \in C, \exists\, x \in S : \varphi(x) = c$. Dans cette Section, nous étudions la résolution exacte du problème d'optimisation Ensemble Tropical Connexe.

> Ensemble Tropical Connexe
> *Version* : *Optimisation*
> **Entrée** : $G = (V, E)$ un graphe, $C \subseteq \mathbb{N}^*$ un ensemble de couleurs, et $\varphi : V \to C$ une coloration de V
> **Sortie** : Déterminer un sous-ensemble minimum $S \subseteq V$ de sommets (au sens de la Section 1.1), tel que $G[S]$ soit connexe et tel que S contienne au moins un sommet de chaque couleur de C

Le problème Ensemble Tropical Connexe est un problème NP-difficile, même lorsqu'on se restreint aux arbres de hauteur au plus 3 [8]. Un algorithme trivial pour résoudre ce problème s'exécute en temps $O^*(2^n)$. ♣

Beaucoup de problèmes NP-difficiles sont solubles en temps polynomial lorsque le graphe d'entrée est un arbre. Ces problèmes sont de plus souvent FPT de paramètre largeur arborescente du graphe. Ces résultats sont souvent justifiés par le Théorème de Courcelle [49]. Si certains problèmes sont NP-difficiles dans le cas général, ils deviennent souvent solubles en temps sous-exponentiel lorsqu'ils sont restreints aux graphes planaires ou aux graphes de genre borné [58, 87]. Lorsqu'un problème n'est pas soluble en temps sous-exponentiel, la question se pose alors de la complexité algorithmique que l'on est en droit d'attendre. Pour un tel problème, existe-t-il un algorithme de résolution en temps $O^*(c^n)$, pour une certaine constante $1 < c < 2$?

Préliminaires. Soient $G = (V, E)$ un graphe, et $\varphi : V \to \mathbb{N}$ une coloration de V. On dit que (G, φ) est un graphe *colorié*, et on pose $C = \{\varphi(v) : v \in V\}$ l'ensemble des couleurs de G. Pour un sous-ensemble de sommets S d'un graphe colorié (G, φ), on note $c(S) = \{c(v) : v \in S\}$ l'ensemble des couleurs de S. S est donc tropical si $c(S) = C$. Pour un graphe colorié (G, φ), on dit qu'une couleur apparaît parmi les sommets de G si cette couleur est attribuée à au moins un sommet de V. Pour cette Section, on notera $l_1(G)$ le nombre de couleurs qui apparaissent exactement une fois, tandis que l'on notera $l_2(G)$ le nombre de couleurs apparaissant au moins deux fois parmi les sommets d'un graphe colorié (G, φ). Pour le problème Ensemble Tropical Connexe, on se restreint naturellement aux graphes d'entrée connexes.

5.2.2 Motivation Algorithmique

Dans cette Sous-Section, nous montrons que si le problème Ensemble Tropical Connexe admet un algorithme sous-exponentiel lorsque le graphe d'entrée appartient à une famille qui peut contenir les arbres, alors l'hypothèse de complexité **SNP** $\subseteq$ **SUBEXP** n'est pas valide. Autrement dit, sous les hypothèses de la Théorie de la Complexité présentées en Section 1.2, ce problème Ensemble Tropical Connexe n'a aucune chance de pouvoir être résolu par un algorithme en temps sous-exponentiel, et ce même lorsque les graphes d'entrée sont restreints aux arbres.

Nous avons vu en Section 1.2 que si cette hypothèse n'est pas valide, alors l'hypothèse **ETH** s'effondre également. L'hypothèse **ETH** a été définie par Impagliazzo et al. [117]. L'hypothèse **ETH** est une hypothèse de travail forte, et il semble peu probable qu'elle s'effondre. Une démonstration de cette dernière aurait beaucoup de conséquences en Théorie de la Complexité. Le fait que sous l'hypothèse **ETH**, le problème Ensemble Tropical Connexe n'admette pas d'algorithme sous-exponentiel motive d'autant plus la construction d'un algorithme de résolution exacte en temps $O^*(c^n)$, avec $1 < c < 2$ la plus petite possible. Nous montrons notre résultat en utilisant la réduction du problème Ensemble Dominant au problème Ensemble Tropical Connexe décrite par Angles d'Auriac et al. [8], et en nous appuyant sur la preuve que le problème Ensemble Dominant n'admet pas d'algorithme sous-exponentiel sur les graphes satisfaisant $\Delta(G) \leq 6$ [86].

Théorème 5.1. *Si le problème Ensemble Tropical Conexe restreint aux arbres de hauteur au plus 3 admet un algorithme sous-exponentiel, alors* ***SNP*** $\subseteq$ ***SUBEXP****.*

Démonstration. Angles d'Auriac et al. [8] ont montré que le problème Ensemble Tropical Connexe est NP-difficile même lorsque l'entrée est un arbre de hauteur au plus 3, en utilisant une réduction depuis le problème Ensemble Dominant. Nous rappelons brièvement cette construction. Soit $G = (V, E)$ une instance du problème Ensemble Dominant. Nous construisons un arbre colorié (T, φ) somme suit. Soit $\sigma : V \to \mathbb{N}$ un ordre arbitraire sur les sommets de V. Initialement, T contient un sommet unique r de couleur $R \notin \{0, 1, \ldots, |V|\}$. Ensuite, pour tout sommet $v \in V(G)$, nous ajoutons à T une étoile dont le centre est colorié $\sigma(v)$, avec une feuille coloriée $\sigma(u)$ pour

tout voisin u de v dans G, et une feuille supplémentaire de couleur 0 reliée à la racine r de T. La Figure 5.1 montre un exemple [1] de cette réduction.

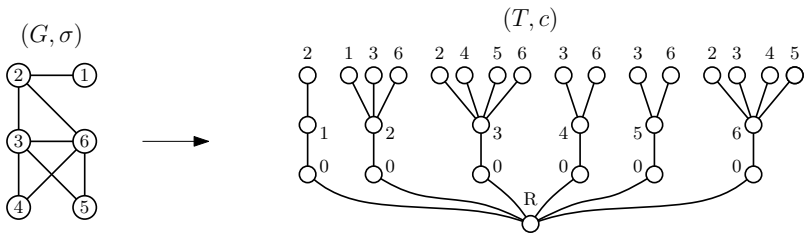


Figure 5.1 – Réduction du problème Ensemble Dominant à une instance de problème d'Ensemble Tropical Connexe [8].

Observons que pour une instance $G = (V, E)$ satisfaisant $\Delta(G) \leq 6$ du problème Ensemble Dominant, la réduction décrite précédemment aboutit à une instance du problème Ensemble Tropical Connexe qui est un arbre de hauteur 3, et qui contient au plus $8n + 1 = O(n)$ sommets. En 2005, Fomin et al. ont montré que le problème Ensemble Dominant n'admet pas d'algorithme sous-exponentiel sur un graphe G qui satisfait $\Delta(G) \leq 6$, à moins que **SNP**$\subseteq$ **SUBEXP**. Ce résultat de Fomin et al. conjugué à notre réduction qui se calcule en temps linéaire achève notre preuve. □

Le Théorème 5.1 motive les algorithmes que nous construisons dans le reste de cette Section pour le problème Ensemble Tropical Connexe.

5.2.3 Algorithme Exact

Nous présentons dans cette Sous-Section un algorithme de résolution exacte pour le problème Ensemble Tropical Connexe. Pour cela, nous réduisons le problème Ensemble Tropical Connexe à deux autres problèmes bien connus, à savoir les problèmes Arbre de Steiner et Ensemble Dominant Rouge-Bleu Connexe. En s'appuyant sur les meilleurs algorithmes pour résoudre ces problèmes, nous utiliserons la technique de *Balancement d'Algorithmes*. Cette méthode nous permettra d'établir un temps d'exécution de $O^*(1.5359^n)$ pour notre algorithme qui résoud le problème Ensemble Tropical Connexe.

Arbre de Steiner
Version : *Optimisation*

Entrée : Un graphe $G = (V, E)$, une fonction de pondération $w : E \to \mathbb{N}$, et un ensemble de terminaux $K \subseteq V$.

Sortie : Déterminer un sous-arbre $T = (V', E')$ de G, avec $K \subseteq V'$, et qui minimise $\sum_{e \in E'} w(e)$.

1. Pour la Figure 5.1, et avec nos notations, on remarquera que $\varphi = c$

Ensemble Dominant Rouge-Bleu Connexe

Version : *Optimisation*

Entrée : $G = (R \cup B, E)$ un graphe dont les sommets sont coloriés soit en rouge (les sommets de R) soit en bleu (sommets de B).

Sortie : Déterminer un sous-ensemble $S \subseteq R$ de sommets rouges de plus petite taille possible qui soit connexe, et tel que tout sommet bleu de B possède dans G au moins un voisin rouge qui soit dans S, c'est-à-dire $B \subseteq N(S)$.

Ces deux problèmes sont NP-difficiles [94]. Les meilleurs algorithmes qui nous concernent pour résoudre les problèmes Arbre de Steiner et Ensemble Dominant Rouge-Bleu Connexe sont dus respectivement à Nederlof [163] et Abu-Khzam et al. [1]. Ces algorithmes s'exécutent respectivement en temps $O^*(W \cdot 2^{|K|})$, où K est l'ensemble de terminaux et W le poids maximum, et en temps $O^*(1.36443^n)$.

Nous décrivons ici une construction qui sera utilisée pour nos algorithmes tant pour réduire le problème Ensemble Tropical Connexe au problème Arbre de Steiner qu'au problème Ensemble Dominant Rouge-Bleu Connexe. Soient $G = (V, E)$ un graphe, φ une coloration de V et C l'ensemble des couleurs de G.

Premièrement, nous définissons le graphe $G' = (R' \cup B', E')$, qui sera utilisé pour réduire Ensemble Tropical Connexe à Arbre de Steiner, de la manière suivante : $R' = V$, $B' = C$, et $E' = E \cup \{\{v, \varphi(v)\} \ : \ v \in V\}$.

Deuxièmement, nous construisons le graphe $G" = (R" \cup B", E")$ en modifiant le graphe $G' = (R' \cup B', E')$. Initialement, nous définissons $R" = R'$, $B" = B'$ et $E" = E'$. Puis pour chaque sommet $v \in V$ dont la couleur n'apparaît qu'une unique fois dans G, nous supprimons de $B"$ le sommet couleur $\varphi(v) \in B'$ qui correspond à sa couleur dans G', et nous déplaçons le sommet $v = v' \in G'$ de $R"$ vers $B"$. À l'issue de cette étape, on pose $B_1, \ldots, B_p$ les composantes connexes du sous-graphe induit $G"[B"]$ par les sommets qui ont été déplacés de $R"$ vers $B"$ au cours de l'opération précédente. Nous procédons alors comme suit. Pour tout $i = 1, 2, \ldots p$, nous contractons dans $G"$ les composantes connexes B_i de $G"[B"]$, de sorte que toutes ces composantes soient remplacées par une unique sommet $b_i \in B"$. Observons qu'à l'issue de l'opération précédente, $B"$ est un stable de $G"$. Finalement, nous procédons à la dernière opération, qui est pour tout b_i, $1 \leq i \leq p$, nous ajoutons dans $G"$ toutes les arêtes nécessaires pour que $N_{G"}(b_i) \subseteq R"$ soit transformé en une clique de $G"$. Le graphe résultant $G" = (R" \cup B", E")$ sera utilisé pour réduire Ensemble Tropical Connexe à Ensemble Dominant Rouge-Bleu Connexe.

Il convient de mentionner la construction de ces deux graphes auxiliaires G' et $G"$ peut être réalisée en temps polynomial $O(n^{O(1)})$. Un exemple[2] d'une telle construction est donné dans la Figure 5.2.

Comme décrite dans la Section 1.5, si l'on souhaite appliquer la technique de *Balancement d'Algorithmes*, il faut avant tout disposer de plusieurs algorithmes pour résoudre le même problème, et qui doivent bien souvent utiliser différentes approches de résolution. Ainsi, pour pouvoir mettre en œuvre cette technique et l'appliquer à notre problème d'Ensemble Tropical Connexe, nous décrivons trois approches de résolution : par la méthode *Brute-Force*, en utilisant une réduction vers Arbre de Steiner, et en utilisant une réduction vers Ensemble Dominant Rouge-Bleu Connexe.

2. Pour la Figure 5.2, et avec nos notations, on remarquera que $\varphi = c$

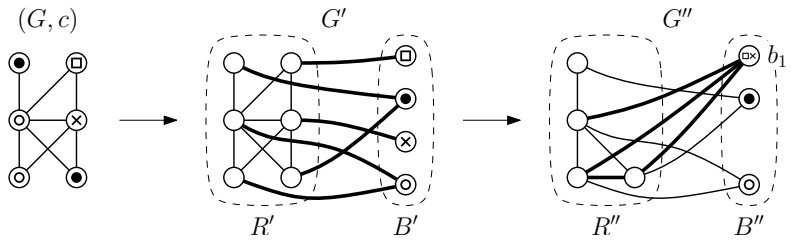


Figure 5.2 – Exemple de construction, de gauche à droite, des graphes auxiliaires G' et G" pour nos réductions : à partir d'un graphe colorié (G, c) avec ensemble de couleurs $C = \{\ \square, \bullet, \circ, \times\}$, nous obtenons le graphe intermédiaire G' avec ensemble de sommets $R' \cup B'$, à partir duquel nous obtenons le graphe G" final avec ensemble de sommets $R'' \cup B''$. Les arêtes surlignées correspondent aux arêtes ajoutées à chaque étape de la construction.

Brute-force. Soit (G, φ) une instance du problème Ensemble Tropical Connexe. Notre algorithme *Brute-Force* génère dans un premier temps l'ensemble U de tous les sommets dont la couleur n'apparaît qu'une unique fois dans G. Cet ensemble U apparaît nécessairement dans tout ensemble tropical de G, donc dans toute solution au problème Ensemble Tropical Connexe. Puis pour chaque sous-ensemble $A \subseteq V - U$ de sommets, l'algorithme vérifie en temps polynomial si $U \cup A$ est un ensemble tropical connexe. Parmi ces solutions, l'algorithme retourne celle de plus petite taille. Cet algorithme s'exécute en temps $O^*(2^{n-l_1(G)})$.

Arbre de Steiner. Le problème d'Ensemble Tropical Connexe peut facilement être réduit au problème Arbre de Steiner de la façon suivante. Soit (G, φ) une instance d'Ensemble Tropical Connexe. Nous construisons l'instance (G', w, K) pour le problème Arbre de Steiner, où $G' = (R' \cup B', E')$ le graphe auxiliaire dont la construction a été décrite ci-dessus, et avec $K = B'$. Observons que $|K| = |B'| = |C|$. Nous définissons la fonction de pondération w comme suit :
$\diamond$ $w(e) = n = |V|$ pour toute arête $e \in E(G')$ incidente à un sommet de B'
$\diamond$ $w(e) = 1$ sinon, c'est-à-dire $w(e) = 1$ pour toute arête $e = \{u', v'\}$ satisfaisant $u', v' \in R'$.

Lemme 5.2. *Le graphe colorié (G, φ) admet un ensemble tropical connexe de taille k si et seulement si (G', w, B') admet un arbre de Steiner de poids $k' = k - 1 + |C|n$.*

Démonstration. Pour cette preuve, nous ne faisons pas de distinction entre un sommet $v \in V$ de G et son sommet correspondant $v' \in R'$ dans G'.
Soit S un ensemble tropical connexe (G, φ) avec $|S| = k$. S étant connexe, il admet en particulier au moins un arbre couvrant dans $G[S] = G'[S]$ constitué de $k - 1$ arêtes. Soit A un tel arbre couvrant pouvant être choisi arbitrairement. Nous construisons un arbre de Steiner $\tilde{T}$, dont l'ensemble d'arêtes $\tilde{E}$ est initialement vide, pour l'instance (G', w, B'). Dans un premier temps, nous sélectionnons les $k - 1$ arêtes de A que nous insérons dans $\tilde{E}$. Par construction, ces arêtes sont toutes de poids 1. Puis pour tout terminal $i \in B'$, où $i \in C$ est une couleur de φ, nous ajoutons dans $\tilde{E}$ une arête $e_i = \{v, i\} \in E' : \varphi(v) = i, v \in S$. Pour tout $i \in C$, une telle arête existe nécessairement par construction de G' et puisque S est tropical dans G. De plus, les arêtes $e_i, i \in C$, vérifient toutes $w(e_i) = n$. Nous vérifions finalement que $\tilde{T}$ est bien un arbre de Steiner pour (G', w, B'), et qu'il satisfait $k' = w(\tilde{T}) = \sum_{e \in \tilde{E}} w(e) = k - 1 + |C|n$.
Réciproquement, soit $\tilde{E}$ l'ensemble d'arêtes d'un arbre de Steiner $\tilde{T}$ of (G', w, B') de poids $k' = k - 1 + |C|n$, avec $k \leq n$. Nous construisons un ensemble $S^* \subseteq V$ tropical connexe de taille k. Par construction, B' est un ensemble stable dans G'. En conséquence, $\tilde{E}$ contient pour chaque sommet $b \in B'$ une arête incidente à b. $\tilde{E}$ contient en conséquence au moins $|B'| = |C|$ arêtes de poids n.

Puisque $w(\tilde{T}) < (|C|+1)\cdot n$, nous pouvons déduire que $\tilde{E}$ possède exactement $|C|$ arêtes de poids n. Soit $S^* \subseteq R' = V$ l'ensemble des sommets de $\tilde{T} - B'$. Remarquons que le sous-graphe induit $\tilde{T}[S^*]$ de $\tilde{T}$ est un sous-graphe de G. Avec la remarque précédente, $\tilde{T}[S^*]$ contient de plus exactement $k-1$ arêtes. Nous montrons que S^* est tropical. Soit S l'ensemble des sommets de R' incidents aux arêtes de poids n dans $\tilde{E}$. Avec $R' = V$, par construction de G', et puisque $\tilde{T}$ est un arbre de Steiner avec ensemble de terminaux $B' = C$, $S \subseteq V$ est tropical dans G. Par construction, $S \subseteq S^*$, il vient que S^* est tropical. Montrons que S^* est connexe. Nous avons montré que chaque sommet de B' est incident à exactement une arête de $\tilde{E}$. Avec la propriété qu'aucune arête de $\tilde{E}$ ne peut relier deux sommets de B' car B' est un ensemble stable de G', nous avons que les terminaux B' sont des feuilles de $\tilde{T}$. Le sous-graphe $\tilde{T} - B' = \tilde{T}[S^*]$ de $\tilde{T}$ est par conséquent un sous-abre de $\tilde{T}$, ce qui implique que S^* est connexe dans G. Finalement, $\tilde{G}[S^*]$ étant un arbre couvrant de S^* dans G avec $k-1$ arêtes, nous concluons que $|S^*| = k$. □

Nous utilisons l'Algorithme de Nederlof [163] pour résoudre le problème Arbre de Steiner en temps $O^*(W2^k)$. Avec le nombre de terminaux $k = |C|$, et $W = n$, cette réduction conduit à un algorithme pour résoudre Ensemble Tropical Connexe en temps $O^*(2^{|C|})$.

Ensemble Dominant Rouge-Bleu Connexe. Le problème d'Ensemble Tropical Connexe peut être réduit au problème d'Ensemble Dominant Rouge-Bleu Connexe de la manière suivante. Soit (G, φ) une instance d'Ensemble Tropical Connexe. Nous construisons une instance $G" = (R" \cup B", E")$ d'Ensemble Dominant Rouge-Bleu Connexe, où $G"$ est le graphe auxiliaire construit comme définit ci-dessus, $R"$ est l'ensemble des sommets rouges et $B"$ est l'ensemble des sommets bleus.

Lemme 5.3. *(G, φ) admet un ensemble tropical connexe de taille k si et seulement si $G" = (R" \cup B", E")$ admet un ensemble dominant rouge-bleu connexe de taille $k' = k - l_1(G)$.*

Démonstration. Pour cette preuve, nous ne faisons pas de distinction entre un sommet $v \in V$ de G et son sommet correspondant v' dans $G"$.
Soit S un ensemble tropical connexe de G de taille k. Soit U l'ensemble (peut-être vide) des sommets dont les couleurs apparaissent une unique fois dans G. Il est clair que $U \subseteq S$. Nous affirmons que $D^* = S - U$ est un ensemble dominant rouge-bleu connexe de $G"$. Trivialement, $D^* \subseteq R"$ par construction de $G"$. Montrons que D^* est connexe dans $G"$. Par construction, $B_1, \ldots, B_p$ sont les composantes connexes de $G[U]$, et pour chaque $i \in \{1, \ldots, p\}$, l'ensemble connexe B_i de G est contracté en un sommet b_i de $G"$. Puisque S est connexe dans G, $D = (S - U) \cup \{b_1, \ldots, b_p\}$ est connexe dans $G"$, puisque nous contractons des sous-ensembles connexes dans G. Maintenant en supprimant $\{b_1, \ldots, b_p\}$ de D nous obtenons D^*. Considérons son sous-graphe induit $G"[D^*]$ dans $G"$. Par construction de $G"$, les voisins de chaque sommet b_i sont des cliques dans $G"$. Supprimer les sommets b_i ne peut avoir pour effet de rendre D^* non connexe. En conséquence, D^* est connexe dans $G"$. Finalement par construction, $D^* \subseteq R"$, et $N(D^*) \cap B"$ contient tous les sommets $c \in B"$, représentant les couleurs $c \in C$ de φ apparaissant au moins deux fois dans G. Puisque S est connexe dans G et que $\{b_1, b_2, \ldots, b_p\} \subseteq B"$ est un ensemble stable dans $G"$, alors chaque sommet de $\{b_1, b_2, \ldots, b_p\}$ admet un voisin dans D^*, ce qui implique que D^* est un ensemble dominant rouge-bleu dans $G"$. En résumé, D^* est un ensemble dominant rouge-bleu connexe de $G"$ satisfaisant $|D^*| = |S| - |U| = k - l_1(G) = k'$.
Réciproquement, soit $D \subseteq R"$ un ensemble dominant rouge-bleu connexe de $G"$ avec $|D| = k' = k - l_1(G)$. Par construction de $G"$ et par propriété de D, pour tout $i = 1, \ldots, p$: l'ensemble D contient au moins un sommet de $N(b_i) \cap R"$ dans $G"$. En conséquence $D \cup \bigcup_{i=1}^{p} B_i$ est un ensemble connexe dans G', ainsi que dans G. Rappelons que pour tout i, le sommet b_i est obtenu par contraction

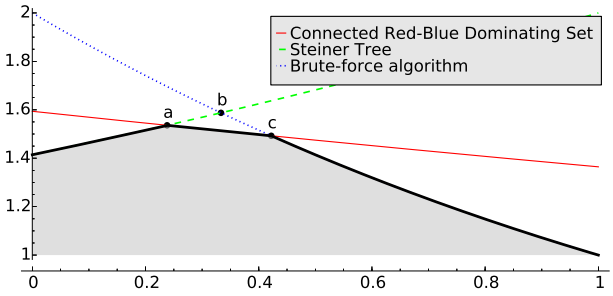


Figure 5.3 – Représentation par une courbe noire du temps d'exécution de l'Algorithme du Théorème 5.4. L'axe des abscisses représente le ratio α pour lequel $l_1(G) = \alpha \cdot n$. L'axe des ordonnées représente la constante c du temps d'exécution $O^*(c^n)$ de notre algorithme en fonction de α. Les points a à c correspondent respectivement aux points d'abscisse $\alpha_a = 0.23814$, $\alpha_b = \frac{1}{3}$ et $\alpha_c = 0.42218$.

des composantes connexes B_i. De plus, remarquons que $D \cup \bigcup_{i=1}^{p} B_i$ est tropical puisqu'il contient tous les sommets de G ayant une couleur qui apparaît une unique fois dans G, et que D domine $B" - \bigcup_{i=1}^{p} B_i$, i.e. D continent un sommet de chaque couleur apparaissant au moins deux fois. Finalement $D \cup \cup_{i=1}^{p} B_i$ est un ensemble tropical connexe de G de taille $k' + l_1(G) = k$. □

L'Algorithme exact de Abu-Khzam et al. [1] résout le problème Ensemble Dominant Rouge-Bleu Connexe en temps $\mathcal{O}^*(1.36443^n)$. Notre réduction conduit donc à un algorithme pour résoudre le problème Ensemble Tropical Connexe sur l'instance (G, φ) en temps $O^*(1.36443^{n+l_2(G)})$.

Balancement des Algorithmes. Nous décrivons finalement notre algorithme pour résoudre le problème Ensemble Tropical Connexe sur les graphes en général. Nous appliquons la technique de *Balancement d'Algorithmes*, en choisissant judicieusement les trois différents algorithmes décrits ci-dessus selon leur temps d'exécution en fonction de $l_1(G)$. Soit (G, φ) une instance du problème Ensemble Tropical Connexe, avec $l_1(G) = \alpha \cdot n$, pour une certaine constante $\alpha \in \mathbb{R}^+_{\leq 1}$.

- ◇ Si $l_1(G) < 0.23814 \cdot n$, alors nous réduisons l'instance à une instance du problème Arbre de Steiner. Nous utilisons l'Algorithme de Nederlof [163] sur l'instance (G', w, K). Dans ce cas, notre algorithme pour résoudre le problème Ensemble Tropical Connexe s'exécute en temps $O^*(2^{|C|})$. Observons que dans ce cas, $|C| = l_1(G) + l_2(G) < 0.23814 \cdot n + \frac{1-0.23814}{2} \cdot n = 0.61907 \cdot n$.
- ◇ Si $0.23814 \cdot n \leq l_1(G) \leq 0.42218 \cdot n$, alors nous réduisons l'instance à une instance du problème Ensemble Dominant Rouge-Bleu Connexe. Nous utilisons l'Algorithme de Abu-Khzam et al. [1] sur l'instance $G"$. Notre algorithme s'exécute alors en temps $O^*(1.36443^{n+l_2(G)})$ pour résoudre le problème Ensemble Tropical Connexe sur l'instance originelle. Observons que dans ce cas, $n + l_2(G) \leq n + \frac{1}{2} \cdot n - \frac{1}{2} \cdot l_1(G) \leq 1.38093 \cdot n$.
- ◇ Finalement, si $0.42218 \cdot n < l_1(G)$, nous utilisons l'algorithme *Brute-Force* directement sur l'instance (G, φ). Dans ce cas, notre algorithme s'exécute en temps $O^*(2^{n-l_1(G)})$ pour résoudre le problème Ensemble Tropical Connexe. Obsrvons de plus que $n - l_1(G) \leq (1 - 0.42218) \cdot n = 0.57782 \cdot n$.

Théorème 5.4. *Il y a un algortihme en temps $O^*(1.5359^n)$ et espace polynomial pour résoudre le problème Ensemble Tropical Connexe.*

Démonstration. L'espace polynomial utilisé vient du fait que notre algorithme balancé utilise pour chaque routine différente un espace polynomial. La Figure 5.3 montre la variation du temps d'exécution $O^*(c^n)$ de l'algorithme en fonction de la valeur $l_1(G) = \alpha \cdot n$ de l'instance (G, φ) considérée. □

5.3 Tropicalité et Connexité sur les Arbres

5.3.1 Introduction

La version optimisation du problème Ensemble Tropical Connexe sur les arbres de hauteur au plus trois est NP-difficile [8]. Nous avons montré en Sous-Section 5.2.2, que sous des hypothèses communément admises de la Théorie de la Complexité, il n'y a pas d'algorithme en temps sous-exponentiel pour résoudre ce problème sur les arbres. Dans cette Section, nous présentons des algorithmes de *Branchement* pour ce problème. Dans un premier temps, nous proposons un algorithme en temps optimal $O^*(3^{n/3})$ pour énumérer les ensembles connexes tropicaux minimaux d'un arbre. Cet algorithme nous donne une borne supérieure pour le temps d'exécution d'un algorithme de résolution du problème Ensemble Tropical Connexe sur les arbres. Pour le problème Ensemble Tropical Connexe restreint aux arbres, nous construisons un algorithme spécifique qui s'exécute en temps $O^*(1.2721^n)$. Le reste de cette Sous-Section est dédiée à la description des notations utilisées et des instances de sous-problèmes de nos problèmes étudiés.

Notation. Nous référons à la Sous-Section 1.3.1 pour les notations usuelles sur les graphes, et à la Sous-Section 1.3.3 pour toute terminologie propre aux arbres. Soit $T = (V, E)$ un arbre et soit (T, φ) une instance du problème Ensemble Tropical Connexe. Dans le reste de cette Section, nous considérons T enraciné de racine $r \in V$. Pour un sommet $v \in V$, on note $T(v)$ le sous-arbre de T enraciné à v, et on note $|T(v)|$ le nombre de ses sommets. Pour un sommet $v \in (V \setminus \{r\})$, on note $p(v)$ le père de v dans l'arbre enraciné T. Finalement, pour un nœud interne $v \in V$ de T, on note $s_1(v), \ldots, s_k(v)$, $k \geq 1$, les fils de v dans T.

Instances et Sous-Problèmes. Comme présenté dans la Sous-Section 1.5.2, un algorithme de *Branchement* résout un sous-problème en appliquant différentes règles de réduction et de branchement. Soit une instance du problème Ensemble Tropical Connexe composée d'un arbre enraciné $T = (V, E)$, d'une coloration φ de V, avec C l'ensemble des couleurs de φ. Pour nos algorithmes récursifs, les variables T, φ et C sont globales. Pour notre étude, une instance de sous-problème est une 3-partition (S, F, D) de V. Pour une instance (S, F, D), l'objectif est de déterminer un ensemble tropical connexe S^* de (T, φ) qui satisfait $\{S \subseteq S^*, S^* \cap D = \emptyset, C' = \emptyset\}$, et qui soit respectivement pour le problème considéré minimal ou minimum, au sens de la Section 1.1. Nous appelerons un tel ensemble S^* une solution au sous-problème (S, F, D). Une instance (S, F, D) est donc définie comme suit.

- S est l'ensemble des sommets *séléctionnés*, c'est-à-dire les sommets qui ont déjà été choisis pour former un sous-ensemble de toute solution au sous-problème.
- F est l'ensemble des sommets *libres*, c'est-à-dire qu'aucune décision n'a été prise à leur sujet concernant leur appartenance à une solution.
- D est l'ensemble des sommets *écartés*, traduit de l'anglais *discarded*, c'est-à-dire les sommets qui n'appartiennent à aucune solution au sous-problème.

Nous verrons clairement que la construction de nos algorithmes implique qu'une instance (S, F, D) satisfait toujours les propriétés suivantes :

(i) La racine r de T appartient à S
(ii) S et $S \cup F$ sont des ensembles connexes de T
(iii) Pour tout $v \in D$: tous les sommets de $T(v)$ appartiennent à D.

Finalement, pour une instance (S, F, D), on dénote $T' = T[S \cup F]$ le sous-arbre de T induit par $S \cup F$, et on dénote $C' \subseteq C$ le sous-ensemble des couleurs de φ qui ne sont attribuées à aucun sommet de S.

$\texttt{ADD}(X)$	$\texttt{RMV}(X)$
$T''_r \leftarrow T'_r$	$T''_r \leftarrow T'_r - \bigcup_{v\in X} T'_r(v)$
$S' \leftarrow S + \bigcup_{v\in X} v \rightsquigarrow r$	$S' \leftarrow S$
$F' \leftarrow F - \bigcup_{v\in X} v \rightsquigarrow r$	$F' \leftarrow F - \bigcup_{v\in X} T_r(v)$
$D' \leftarrow D$	$D' \leftarrow D + \bigcup_{v\in X} T_r(v)$
$C'' \leftarrow C' - \bigcup_{v\in X}\{\bigcup_{y\in v\rightsquigarrow r} c(y)\}$	$C'' \leftarrow C'$

Figure 5.4 – Description des Procédures basiques ADD et RMV pour nos algorithmes de *Branchement* de la Section 5.3

Pour analyser le temps d'exécution de notre algorithme, nous *mesurons* la taille d'une instance de sous-problème par son nombre $|F|$ de sommets libres.

Procédures. Pour créer des sous-problèmes à partir d'une instance (S, F, D), c'est-à-dire générer des instances de sous-problèmes de plus petite taille sur lesquelles nos algorithmes seront appliqués récursivement, les règles de réduction et de branchement utiliseront les deux procédures basiques suivantes, pour $v \in F$.

$\texttt{ADD}(v)$: Ajouter dans S tous les sommets libres du chemin $v \rightsquigarrow r$ de T' qui relie v à la racine r, supprimer ces sommets de F, et supprimer de C' toutes les couleurs rencontrées sur ce chemin.

$\texttt{RMV}(v)$ Supprimer de F tous les sommets libres de $T(v)$, et les ajouter dans D.

Si notre algorithme utilise la procédure $\texttt{ADD}(v)$, on dit que l'on *sélectionne* le sommet v, et si notre algorithme utilise la procédure $\texttt{RMV}(v)$, on dit que l'on *écarte* ce sommet. Observons que sélectionner un sommet libre $v \in F$ ($\texttt{ADD}(v)$) implique de sélectionner également tous les sommets libres de $v \rightsquigarrow r$. En effet, si un sommet libre de $v \rightsquigarrow r$ appartenait à D, soit donc si l'on écartait un tel sommet libre, alors aucun super-ensemble de $S \cup \{v\}$ ne serait connexe. Pour les mêmes raisons, écarter un sommet libre $v \in F$ implique d'écarter tous les sommets rencontrés dans son sous-arbre enraciné $T'(v) = T(v) \cap F$. Nous étendons ces procédures basiques ADD et RMV à tout sous-ensemble $X \subseteq F$ de sommets libres. Pour $X \subseteq F$, l'application d'une procédure $\texttt{ADD}(X)$ ou d'une procédure $\texttt{RMV}(X)$ transforme l'instance (S, F, D) en une instance de sous-problème (S', F', D') de plus petite taille. La Figure 5.4 décrit de manière formelle ces procédures basiques. Finalement, si $X = \{x_1, \ldots, x_p\}$, nous pourrons écrire plus facilement $\texttt{ADD}(x_1, \ldots, x_p)$ et $\texttt{RMV}(x_1, \ldots, x_p)$ en lieu et place respective de $\texttt{ADD}(\{x_1, \ldots, x_p\})$ et $\texttt{RMV}(\{x_1, \ldots, x_p\})$. Observons que par leur définition, une somme d'application quelconques des procédures ADD et RMV diminue toujours la taille d'un sous-problème dans l'instance résultante.

5.3.2 Algorithme d'Énumération

Nous décrivons un algorithme de *Branchement* pour l'énumération des ensembles tropicaux connexes minimaux d'un arbre. Soit $T = (V, E)$ un arbre, φ une coloration de V et C l'ensemble des couleurs de φ. Soit $c \in C$ la couleur la moins représentée dans C. Alors pour tout $v \in V : \varphi(v) = c$, nous enracinons T en v et nous appliquons l'Algorithme MTCS-Tree 5.1 avec pour instance initiale $(S, F, D) = (\{v\}, V - \{v\}, \emptyset)$.
Sans perte de généralité, nous restreignons notre analyse pour une racine quelconque $r \in V$ de T, avec $r \in S^*$ pour tout ensemble tropical connexe minimal S^* de T.

Théorème 5.5. *L'Algorithme MTCS-Tree 5.1 énumère les Ensembles Tropicaux Connexes Minimaux d'un arbre colorié en temps $O^*(3^{n/3})$. De plus, le temps d'exécution de cet algorithme est optimal.*

Algorithme MTCS-Tree(S, F, D)

Règles de Réduction.

R0.1. Si $C' = \emptyset$: `STOP`, S est un ensemble tropical connexe minimal de T.

R0.2. Si $F = \emptyset$: `STOP`, l'instance n'admet pas de solution.

R1. S'il existe $v \in F$ feuille de T', telle que $\varphi(v) \notin C'$: $\texttt{RMV}(v)$.

R2. S'il existe $v \in F$ telle que $\varphi(v) \in C'$ et telle que pour tout $u \in F - \{v\}$, $\varphi(v) \neq \varphi(u)$: $\texttt{ADD}(v)$.

Règles de Branchement. Pour une règle de branchement, nous écrivons $\langle \mathcal{O}_1 \mid\mid \ldots \mid\mid O_p \rangle$ pour exprimer le fait que l'algorithme créé p instances de sous-problèmes. Le i-ème sous-problème est obtenu en appliquant sur l'instance (S, F, D) les opérations décrites dans l'ensemble $\mathcal{O}_i$.

B1 : S'il existe $v \in F$ feuille de T', telle qu'il y ait au moins un nœud interne $u \in F$ de T' satisfaisant $\varphi(v) = \varphi(u)$: $\langle \{\texttt{ADD}(u), \texttt{RMV}(v)\} \mid\mid \texttt{RMV}(u) \rangle \quad (\mathbf{2}, \mathbf{2})$

B2 : S'il existe $v \in F$ feuille de T', telle qu'il y ait $k-1$ autres feuilles distinctes $x_2, \ldots, x_k \in F$ de T' satisfaisant $\varphi(v) = \ldots = \varphi(x_k)$. En posant $x_1 = v$, et en notant $X_i = \{x_1, \ldots, x_k\} - \{x_i\}$: $\langle \{\texttt{ADD}(v_1), \texttt{RMV}(X_1)\} \mid\mid \ldots \mid\mid \{\texttt{ADD}(v_i), \texttt{RMV}(X_i)\} \mid\mid \ldots \mid\mid \{\texttt{ADD}(v_k), \texttt{RMV}(X_k)\} \rangle \quad (\mathbf{3}, \mathbf{3}, \mathbf{3})$

Algorithme 5.1 – Algorithme de Branchement du Théorème 5.5 pour l'énumération des Ensembles Tropicaux Connexes Minimaux.

Démonstration. Les règles de réduction de l'Algorithme MTCS-Tree constituent un sous-ensemble des règles utilisées par l'Algorithme du Théorème 5.6. Nous conservons donc la même numérotation de ces dernières. Nous référons à la preuve du Théorème 5.6 pour la validité des règles de réduction de l'Algorithme MTCS-Tree.
Les règles de branchement `B1` et `B2` sont correctes. En effet, dans la première branche de `B1`, le fait d'écarter v après avoir sélectionné u est justifié par la règle de réduction `R1`. Le fait d'écarter X_i dans la i-ème branche de `B2` après avoir sélectionné x_i se justifie pareillement.
Dans un arbre T, un ensemble tropical connexe S^* est minimal si et seulement si pour tout $x \in X$: $S^* - \{x\}$ n'est pas tropical, où X est l'ensemble des feuilles de $T[S^*]$. Nous justifions qu'à l'issue de la règle `R0.1`, l'ensemble S retourné par l'algorithme est minimal. Pour voir cela, il faut observer que par la règle `R1`, l'algorithme sélectionne un sommet $v \in F$ pour l'ajouter à S et transformer l'instance (S, F, D) en une instance (S', F', D'), $S \subseteq S'$, à l'unique condition que sa couleur $\varphi(v)$ ne soit pas encore représentée dans S. De plus, observons que le sommet v sélectionné constitue toujours une feuille dans $T[S']$. Les autres sommets de $v \rightsquigarrow r$, par l'application de la procédure $\texttt{ADD}(v)$, ne représentent pas des feuilles de T" mais sont ajoutés dans S car ils sont nécessaires à la condition de connexité de S. Ainsi, on montre par récurrence que si pour l'instance (S, F, D), S est un ensemble tropical connexe minimal pour l'ensemble de couleurs $C - C'$, alors pour l'instance (S', F', D') : S' est un ensemble tropical connexe minimal pour l'ensemble de couleurs $C - C' + \varphi(v)$.

Nous rappelons que pour notre analyse, la taille d'un sous-problème est donnée par le nombre de sommets libres de l'instance considérée. Par définition et application des procédures `ADD` et `RMV`, les règles de réduction de l'Algorithme MTCS-Tree n'augmentent jamais la taille des sous-problèmes.

Nous analysons le nombre de branchement des règles de réduction. Le nombre de branchement de la règle `B1` est donné par $\alpha_1 = \tau(d(u,S) + |T'(v)|, |T'(u)|) \leq \tau(2,2)$. Le nombre de branchement de la règle `B2` est donné par $\alpha_2 = \max_{k \in \mathbb{N}_{\geq 2}}\{\tau(k,\ldots,k)\}$, où k apparaît k fois dans le vecteur de branchement. Par définition, α_2 est le maximum parmi les racines des polynômes $P_k(X) = 1 - \sum_{i=1}^{k} X^{-k} = 1 - k \cdot X^{-k}$. Les racines de ces polynômes sont de la forme $\alpha(k) = k^{1/k}$. Le maximum pour $k \in \mathbb{N}_{\geq 2}$ parmi ces racines est atteint pour $\alpha_2 = \alpha(3) = 3^{1/3}$. Le temps d'exécution de l'Algorithme MTCS-Tree est donc de la forme $O^*(c^n)$, où $c = \max\{\alpha_1, \alpha_2\} = \alpha_2$. L'Algorithme MTCS-Tree s'exécute donc en temps $O^*(3^{n/3})$.

Nous montrons finalement que le temps d'exécution de l'Algorithme MTCS-Tree est optimal. Dans ce but, nous construisons une famille infinie d'arbres coloriés, en fait des étoiles coloriées, dont le nombre d'ensembles tropicaux connexes minimums est $3^{(n-\epsilon)/3}$, $\epsilon \leq 1$. Soit T une étoile avec $n = 3k+1$ sommets, x le centre de l'étoile, et $\{v_i\ ,\ 0 \leq i \leq 3k-1\}$ les feuilles de T. Nous définissons la coloration φ par $c(x) = k$ et $\forall\ 0 \leq i \leq k-1 : c(v_{3i}) = c(v_{3i+1}) = c(v_{3i+2}) = i$. Nous observons que par cette construction, un ensemble tropical minimum doit contenir x, ainsi que pour tout $0 \leq i \leq k-1$, un unique sommet de couleur i. Ainsi, il y a trois possibilités pour un ensemble tropical connexe minimum de représenter la couleur i. Le nombre d'ensembles tropicaux connexes minimums dans ces étoiles est donc $3^{(n-1)/3}$. Bien entendu, ce résultat montre qu'il existe une famille infinie d'étoiles coloriées qui admettent $\Theta(3^{n/3})$ ensembles tropicaux connexes minimaux, puisque tout ensemble tropical connexe minimum est en particulier minimal. □

5.3.3 Algorithme d'Optimisation

Nous décrivons un algorithme de *Branchement* pour résoudre la version optimisation du problème Ensemble Tropical Connexe sur les arbres. Soit $T = (V, E)$ un arbre, φ une coloration de V et C l'ensemble des couleurs de φ. Soit $c \in C$ la couleur la moins représentée dans C. Alors pour tout $v \in V : \varphi(v) = c$, nous enracinons T en v et nous appliquons l'Algorithme TCS-Tree de la Figure 5.2 avec pour instance initiale $(S, F, D) = (\{v\}, V - \{v\}, \emptyset)$. Un ensemble tropical connexe minimum de l'instance (T, φ) est la plus petite taille d'un ensemble S solution de $(\{v\}, V - \{v\}, \emptyset)$ retournée par l'algorithme, sur tous les sommets v racines de T considérés.
Sans perte de généralité, nous restreignons notre analyse pour une racine quelconque $r \in V$ de T, avec $r \in S^*$ pour tout ensemble tropical connexe minimum S^* de T. Cette Sous-Section est consacrée à montrer le Théorème 5.6 suivant :

Théorème 5.6. *L'Algorithme TCS-Tree de la Figure 5.2 calcule un ensemble tropical connexe minimum d'un arbre colorié en temps* $O^*(1.2721^n)$.

Lemme 5.7. *Les règles de réduction de l'algorithme TCS-Tree sont correctes.*

Démonstration. Soit (S, F, D) une instance de sous-problème. On note S^* une solution de (S, F, D). Ainsi $S^* \subseteq V$ est un ensemble tropical connexe satisfaisant $S \subseteq S^*$, $D \cap S^* = \emptyset$, et de taille minimum parmi tous ces censembles.
R0.1. Soit $C' = \emptyset$. Par définition et construction des instances, $G[S]$ est connexe, et puisque $C' = \emptyset$, S est tropical. S est donc un ensemble connexe tropical de T et S est une solution de (S, F, D).
R0.2. Soit $C' \neq \emptyset$ et $F = \emptyset$. Par définition et construction des instances, $G[S]$ est un ensemble connexe de T. Cependant, puisque $C' \neq \emptyset$, alors S n'est pas tropical. Puisque S ne peut être augmenté d'un sommet libre, (S, F, D) n'admet pas de solution.
R1. Soit $v \in F$ une feuille libre de $T' = T[S \cup F]$ telle que $\varphi(v) \notin C'$. Soit $v' \in S$ avec $\varphi(v) = \varphi(v')$. Supposons que S^* soit une solution de (S, F, D), et supposons $v \in S^*$. Puisque $S \subseteq S^*$ et $v' \in S$,

nous avons également $v' \in S^*$. Toutefois, $S^* \setminus \{v\}$ est tropical et connexe. En conséquence, S^* n'est pas une solution de (S, F, D) car non minimal, une contradiction. Nous pouvons donc écarter v et appliquer `RMV`(v).
R2. Soit $v \in F$ l'unique sommet libre de F de couleur $\varphi(v) \in C'$. Toute solution S^* de (S, F, D) doit contenir v pour être tropicale. Nous pouvons donc appliquer `ADD`(v).
R3. Soit $v \in F$ une feuille libre de $T' = T[S \cup F]$, et soit $u \in F - \{v\}$ tel que $\varphi(u) = \varphi(v)$ et $d(u, S) = 1$. Supposons que $v \in S^*$, pour une certaine solution S^* de (S, F, D). Si S^* contient u, alors S^* n'est pas minimum, puisque $S^* - \{v\}$ est tropical et connexe. Si S^* ne contient pas u, alors $(S^* - \{v\}) \cup \{u\}$ est tropical et connexe également. En conséquence, s'il existe une solution contenant v, il existe de même une solution ne contenant pas v. Nous pouvons donc appliquer `RMV`(v).
R4. Soit $v \in F$ une feuille libre de T' et soit $u \in F - \{v\}$ vérifiant $\varphi(u) = \varphi(v)$ et $d(u, v \rightsquigarrow r) \leq 1$. Soit S^* une solution contenant v. Si $u \in v \rightsquigarrow r$, alors $S^* \setminus \{v\}$ est connexe et tropical, et donc S^* n'est pas une solution, ce qui est une contradiction. Si $u \notin v \rightsquigarrow r$, alors il existe $x \in v \rightsquigarrow r$, $x \neq v$, tel que $\{x, u\} \in E$. De plus, puisque S^* est connexe, $v \in S^*$ implique que $x \in S^*$. Alors $(S^* - \{v\}) \cup \{u\}$ est tropical et connexe. Il y a donc une solution qui ne contient pas v, et nous pouvons appliquer `RMV`(v).
R5. Soit $u \in F$ un nœud libre de T', et soit v une feuille sélectionnée de T', $v \in S$, et avec $\varphi(u) = \varphi(v)$. Soit S^* une solution de (S, F, D) contenant un sommet $x \in T'(u)$. Observons que $u \in x \rightsquigarrow r$. Á présent, $r, x \in S^*$ et S^* est connexe implique que $u \in S^*$. Clairement, $v \in S$ implique que $v \in S^*$. En réunissant tous les arguments ensembles avec $u \in S^*$, nous obtenons $S^* - \{v\}$ est tropical et connexe, ce qui contredit $v \in S$. Par conséquent, $u \notin S^*$ et nous pouvons appliquer `RMV`(u).
R6. Lorsque cette règle s'applique, alors tout sous-ensemble de feuilles libres de T', et de couleur deux à deux distinctes, peut être ajouté à S pour obtenir une solution S^*. Nous pouvons sélectionner une feuille quelconque de T' pour l'ajouter à S. □

Lemme 5.8. *Les règles de branchement de l'Algorithme TCS-Tree sont correctes.*

Démonstration. Nous commençons par des commentaires généraux concernant les règles de branchement de l'algorithme. Les règles `B1`, `B2` et `B3` effectuent une analyse de cas qui couvre toutes les possibilités pour lesquelles il existe un sommet libre $v \in F$ satisfaisant $d(v, S) \geq 2$. Si aucun de ces cas ne peut être appliqué, alors tous les sommets libres sont voisins à des sommets de S, et dans ce cas les règles de réduction s'appliquent. Considérons maintenant les neufs règles de branchement. Chacun d'elles soit sélectionne un sommet libre x ou alors l'écarte. En conséquence une branche de la règle est obenue par l'application de la procédure `ADD` et l'autre est obtenue par l'application de la procédure `RMV`. Une telle règle est toujours correcte. Consécutivement, dans les branches de la règle sont appliquées lorsque c'est possible certaines règles de réductions. Cela garantit la validité d'une règle de branchement. Nous discutons en détail l'application successive des règles de réductions aux branches des règles de branchement.

B1.a. Dans la première branche, écarter v' et v'' de F après avoir ajouté v à S est justifié par la règle de réduction `R5`.
B1.b. Dans la première branche, après avoir ajouté $p(v)$ à S, la distance de v à S devient 1. Par conséquent, supprimer v' après avoir sélectionné $p(v)$ est justifié par la règle de réduction `R3`.
B1.c. Dans la première branche, écarter v' de F consécutivement à la sélection de v dans S est justifié par la règle de réduction `R5`. Dans la deuxième branche, écarter v de F a pour conséquence que v' devient l'unique sommet libre représentant de sa couleur. En conséquence, sélectionner v' après avoir écarté v de F est justifié par la règle de réduction `R2`.

B2.a. Dans la première branche, après avoir ajouté $p(v)$ à S, la distance de v à S devient 1. Par conséquent, supprimer v' et v'' après avoir sélectionné $p(v)$ est justifié par la règle de réduction `R3`.
B2.b. Dans la première branche, écarter v' et v'' de F après avoir sélectionné v dans S est justifié par la règle de réduction `R5`.
B2.c. Dans la première branche, après avoir ajouté v'' à S, la couleur de v et v' n'apparaissent plus dans C'. En conséquence, écarter v et v' de F après avoir ajouté v'' à S est justifié par la règle de réduction `R1`.
B2.d. Dans la première branche, écarter v' de F après avoir ajouté v à S est justifié par la règle de réduction `R5`. Dans la deuxième branche, v' devient l'unique sommet libre représentant de sa couleur après avoir écarté v de F. Par conséquent, sélectionner v' dans S après avoir écarté v de F est justifié par la règle de réduction `R2`.
B3.a Dans la première branche, observons dans un premier temps que $v_1, \ldots, v_k$ sont des feuilles de T' puisque la règle de réduction `R3` ne s'est pas appliquée pour écarter v. De plus, observons qu'après avoir ajouté $p(v)$ à S, la distance de v à S devient 1. Ainsi, écarter $v_1, \ldots, v_k$ de F après avoir sélectionner $p(v)$ dans S est justifié par la règle de réduction `R3`. De plus, après avoir écarté $v_1, \ldots, v_k$ de F, v devient l'unique représentant libre de sa couleur, il faut donc le sélectionner dans S, justifié par la règle de réduction `R2`.
B3.b. Les opérations successives de la première branche sont justifiées de manière similaire à la première branche de la règle `B3.a`. Dans la deuxième branche, observons qu'écarter $p(v)$ de F écarte également v de F, et alors v' devient l'unique représentant libre de sa couleur. En conséquence, sélectionner v' dans S est justifié par la règle de réduction `R2`. □

Théorème 5.6. *L'Algorithme TCS-Tree de la Figure 5.2 calcule un ensemble tropical connexe minimum d'un arbre colorié en temps* $O^*(1.2721^n)$.

Démonstration. Soit $c \in C$ est la couleur la moins représentée dans C. Alors pour tout $v \in V$: $\varphi(v) = c$, nous enracinons T en v et nous appliquons l'algorithme TCS-Tree de la Figure 5.2 avec pour instance initiale $(\{v\}, V - \{v\}, \emptyset)$. Pour chaque exécution de l'algorithme, et pour chaque $S \subseteq V$ renvoyé par la règle `R0.1`, nous conservons celui de plus petite taille possible. Les Lemmes 5.7 et 5.8 montrent la validité de notre algorithme. Nous analysons le temps d'exécution de l'Algorithme TCS-Tree. Rappelons que nous mesurons la taille d'un sous-problème par le nombre $|F|$ de sommets libres que contient son instance (S, F, D).
Commençons par quelques commentaires généraux concernant l'analyse de notre algorithme. Typiquement, nos arguments sont les suivants. Lorsque la procédure `ADD`(x) est appliquée, nous déplaçons $d(x, S)$ sommets de F à S, ce qui a pour effet de décroître le nombre de sommets libres par $d(x, S)$. Lorsque la procédure `RMV`(x) est appliquée, tous les sommets qui appartiennent au sous-arbre $T'(x)$ de T' enraciné à x sont déplacés de F vers D. Cette opération a donc pour effet de diminuer la mesure de l'instance courante par $|T'(x)|$. Observons que l'opération `RMV`(x) implique une diminution du nombre de sommets libres par au moins 2 si x est un nœud interne de T'. De plus, observons que lorsque une règle de branchement est appliquée, donc lorsque toutes les règles de réduction ont échouées à être appliquées, alors toutes les feuilles libres $v \in F$ de T' vérifient $d(v, S) \geq 2$. Nous analysons à présent toutes les règles de réduction de notre algorithme.

Pour déterminer le nombre de branchement de la règle `B1.a`, nous distinguons deux cas. Premier cas : $v'' \in T'(v')$ (ou pareillement $v' \in T'(v"))$, alors le nombre de branchement est $\tau(d(v, S) + |T'(v')|, |T'(v)|)$. En observant que $|T'(v')| \geq 3$ puisque v'' est un nœud interne de T', le nombre de branchement est alors $\tau(7, 1)$. Premier cas : le nombre de branchement est $\tau(d(v, S) + |T'(v')| + |T'(v'')|, |T'(v)|) \leq \tau(8, 1)$. Le nombre de branchement de la règle est alors le plus grand des deux, c'est-à-dire pour le premier cas $\tau(7, 1) \leq 1.2555$.

La règle `B1.b` a un nombre de branchement $\tau(d(p(v), S) + |T'(v')|, |T'(p(v))|) \leq \tau(4, 2) \leq 1.2721$.
La règle `B1.c` a un nombre de branchement $\tau(d(v, S)+|T'(v')|, |T'(v)|+d(v', S)) \leq \tau(6, 3) \leq 1.1740$. Observons que $d(v', S) \geq 2$ puisque la règle de réduction `R3` ne s'est pas appliquée.
La règle `B2.a` a un nombre de branchement $\tau(d(p(v), S) + |T'(v')| + |T'(v'')|, |T'(p(v))|) \leq \tau(4, 2) \leq 1.2721$.
La règle `B2.b` a un nombre de branchement $\tau(d(v, S)+|T'(v')|+|T'(v'')|, |T'(v)|) \leq \tau(7, 1) \leq 1.2555$. Observons que $d(v', S) \geq 2$ et $d(v'', S) \geq 2$ puisque la règle `R3` ne s'est pas appliquée. De plus, aucune feuille libre de T' ne peut être à distance au 4 ou plus de S, autrement la règle `B1` se serait appliquée. Par conséquent, puisque v' et v'' sont des nœuds internes, il vient que ni v' ne peut appartenir à $T'(v'')$, ni v'' ne peut appartenir à $T'(v')$.
La règle `B2.c` a un nombre de branchement $\tau(d(v'', S) + |T'(v)| + |T'(v')|, |T'(v'')|)$. Observons que $d(v'', S) \geq 2$ puisque la règle `R3` ne s'est pas appliquée. Par conséquent, le nombre de branchement de la règle est au plus $\tau(4, 2) \leq 1.2721$.
La règle `B2.d` a un nombre de branchement $\tau(d(v, S)+|T'(v')|, |T'(v)|+d(v', S)) \leq \tau(4, 3) \leq 1.2208$. Rappelons que $d(v', S) \geq 2$ puisque la règle `R3` ne s'est pas appliquée.
La règle `B3.a` a un nombre de branchement $\tau(d(p(v), S) + 1 + |T'(v_1)| + \ldots + |T'(v_k)|, T'(p(v))|) \leq \tau(4, 2) \leq 1.2721$.
La règle `B3.b` a un nombre de branchement $\tau(d(p(v), S)+1+|T'(v')|, |T'(p(v))|+d(v', S)) \leq \tau(3, 4) \leq 1.2208$.
Notre analyse montre que le plus grand nombre de branchement parmi les règles de branchement est $\tau(2, 4) \leq 1.2721$. Par conséquent, puisque nous exécutons l'Algorithme TCS-Tree au plus n fois, notre algorithme pour résoudre le problème Ensemble Tropical Connexe sur les arbres s'exécute en temps $O^*(1.2721^n)$. □

5.4 Domination Romaine Faible

5.4.1 Introduction

De nombreuses variantes du problème d'Ensemble Dominant ont été introduites et longuement étudiées autant d'un point de vue structurel qu'algorithmique. Il y a des centaines de références sur la domination dans les graphes et ses variantes, et de nombreux livres et études bien connus y sont entièrement consacrés [111]. Une de ces variantes appelée Domination Romaine a été introduite en [45] et a été motivée par les articles de Stewart [179] et de ReVelle et Rosing [167]. En général, l'objectif est de protéger un ensembles de stations (sommets du graphe) en utilisant le plus petit nombre possible de légions romaines (à placer sur ces stations). Motivée par un décret de l'Empereur Constantin le Grand au cours du IVè siècle ap. J.-C., la Domination Romaine utilise les règles suivantes pour protéger un graphe. Un sommet peut se *protéger* lui-même s'il dispose d'une légion, et protège tous ses voisins s'il dispose de deux légions, car Constantin décréta que deux légions devaient être positionnées avant qu'une ne puisse se mouvoir vers une destination adjacente pour le défendre. Le problème Domination Romaine demande de minimiser le nombre de légions utilisées pour défendre tous les sommets.
De nombreux articles ont été publiés autour de ce problème, qui a été étudié sous différents points de vue [32, 35, 44, 78, 144, 147, 191]. En particulier, ce problème est NP-difficile et a en conséquence été abordé d'un point de vue Algorithmique Exacte Exponentielle. Le premier algorithme non trivial avait un temps d'exécution de $O^*(1.6183^n)$ et utilisait un espace polynomial [142]. Ce résultat a été récemment amélioré en temps $O^*(1.5673^n)$ [183], et peut encore l'être en temps $O^*(1.5014^n)$ au prix d'un coût en espace exponentiel [183]. Le problème de Domination Romaine peut être

mis en relation avec différentes autres variantes de *defense-like* domination, telles que DOMINATION SÉCURISÉE [43, 44, 102], ou DOMINATION EXTERNE [99, 100].
Dans cette Section, nous nous concentrons sur une autre variante problème de DOMINATION ROMAINE. En 2003, Henning et al. [112] ont considéré l'idée suivante : une station t peut également être protégée si une des stations voisines possède une légion qui peut se mouvoir jusqu'à t sans que cela ne remette en cause la protection de l'ensemble des stations (ensemble de sommets). Cette variante ajoute une certaine forme de dynamisme au problème et introduit le problème de DOMINATION ROMAINE FAIBLE.

Domination Romaine Faible. Etant donné un graphe non orienté $G = (V, E)$, nous souhaitons trouver une fonction *wrd-fonction* (*weak roman domination function*) de poids minimum, c'est-à-dire une fonction $f : V \to \{0, 1, 2\}$ telle que tout sommet $v \in V$ soit *défendu* (c'est-à-dire il existe un voisin u de v, avec la possibilité que $u = v$, tel que $f(u) \geqslant 1$) et pour tout sommet $v \in V$ avec $f(v) = 0$ il existe un voisin u de v tel que $f(u) \geqslant 1$ et la fonction $f_{u\to v}$ définie par :

$$f_{u\to v}(x) = \begin{cases} 1 & \text{si } x = v \\ f(u) - 1 & \text{si } x = u \\ f(x) & \text{si } x \notin \{u, v\} \end{cases}$$

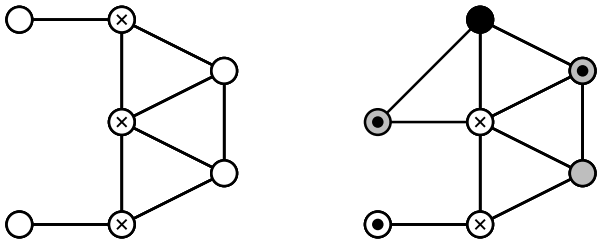
ne contienne aucun sommet non défendu.
Le *poids* d'une wrd-fonction f est défini par $cost(f) = \sum_{v\in V} f(v)$.

DOMINATION ROMAINE FAIBLE
VERSION : *OPTIMISATION*

Entrée : Un graphe $G = (V, E)$
Sortie : Déterminer une wrd-fonction f de G de poids minimum

L'algorithme trivial d'énumération s'exécute en temps $O^*(3^n)$ et espace polynomial et est le meilleur connu pour ce problème jusqu'à présent. Nous améliorons cette borne du temps d'exécution en produisant deux algorithmes plus rapides que l'algorithme trivial d'énumération. Nous montrons dans un premier temps que le problème peut être résolu en temps $O^*(2^n)$ avec espace exponentiel. Puis nous construisons dans un deuxième temps un algorithme en temps $O^*(2.2279^n)$ avec espace polynomial. Nos résultats sont reliés aux propriétés structurelles d'une wrd-fonction. Nous aurons besoin de plus du meilleur algorithme avec espace polynomial pour le problème ENSEMBLE DOMINANT ROUGE-BLEU. ♣

Le problème DOMINATION ROMAINE FAIBLE se présente comme une variante du problème de domination. Dans le problème classique ENSEMBLE DOMINANT, on se donne un graphe non orienté $G = (V, E)$, et on cherche un ensemble dominant S, i.e. tout sommet $v \in V$ soit appartient à S soit a un voisin dans S, de taille minimum. Le problème ENSEMBLE DOMINANT est un des célèbres problèmes NP-difficiles [94], et a retenu beaucoup d'attention au cours de la dernière décennie. En particulier, l'algorithme d'énumération trivial en temps $O^*(2^n)$ a été amélioré par toute une série d'articles [85, 119, 184]. Les meilleurs algorithmes de résolution connus actuellement pour ce problème s'exécutent respectivement en temps $O^*(1.4864^n)$ et espace polynomial, et en temps $O^*(1.4689^n)$ et espace exponentiel [119].



(a) (b)

Figure 5.5 – (a) Un graphe $G = (V, E)$, et une wrd-fonction où chaque légion est représentée par une croix. Tous les sommets sont défendus-sécurisés. Le sommet noir (b) est défendu-sécurisé (on peut y déplacer une légion sans créer de sommet non défendu), les sommets gris sont non défendus-sécurisés (tout mouvement créé un sommet non défendu) et les sommets représentés par un cercle sont faiblement défendus.

5.4.2 Propriétés Structurelles

Légions et wrd-Fonctions.

Une fonction $f : V \to \{0, 1, 2\}$ est appelée une *fonction légion*. Etant donnée f, un sommet $v \in V$ est dit *securisé* si $f(v) \geqslant 1$, et *non sécurisé* sinon. De la même manière, un sommet $v \in V$ est dit *défendu* s'il existe $u \in N[v]$ tel que $f(u) \geqslant 1$. Sinon, v est dit *non défendu*. La fonction f est une *wrd-fonction*, ou fonction de domination romaine faible, si f n'engendre pas de sommet non défendu, et pour tout sommet $v \in V$ non sécurisé il existe un sommet sécurisé $u \in N(v)$ tel que la fonction légion $f_{u\to v}$, obtenue à partir de f en ayant *déplacé* une légion de u vers v, n'admet pas de sommet non défendu (voir Figure 5.5 (a)).

Soit une fonction légion f, nous dénotons respectivement V_f^1 et V_f^2 les ensembles $\{v \in V \ : \ f(v) = 1\}$ et $\{v \in V \ : \ f(v) = 2\}$, et nous définissons son *ensemble induit* par $V_f = V_f^1 \cup V_f^2$. Le *poids* de f est alors défini par $cost(f) = \sum_{v\in V} f(v) = |V_f^1| + 2|V_f^2|$. Observons que lorsque f est une wrd-fonction, l'ensemble V_f est un ensemble dominant (non nécessairement minimal) de G.

Sommets défendus-sécurisés.

Nous distinguons deux types de sommmets défendus. Soient $v \in V$ un sommet et f une fonction légion. On dit que v est *défendu-sécurisé* par f si une des propriétés suivantes est satisfaite :
- ◇ v est sécurisé (i.e. $f(v) \geqslant 1$).
- ◇ il existe un voisin u de v tel que $f(u) = 2$.
- ◇ il existe un voisin u de v tel que $f(u) = 1$ et les sommets non défendus par $f_{u\to v}$ sont les mêmes que les sommets non défendus par f, *i.e.*, $f_{u\to v}$ ne créé par de nouveau sommet non défendu.

Dans le cas contraire, on dit que v est *non défendu-sécurisé*. Notons qu'une fonction légion f est une wrd-fonction si et seulement si tous les sommets $v \in V$ sont défendus-sécurisés par f.

Observons que pour chaque sommet non défendu-sécurisé v, nous avons $f(v) = 0$, $f(u) = 1$ pour chaque voisin sécurisé u de v et la fonction légion $f_{u\to v}$ définie précédemment contient (entre autres) un sommet non défendu $w \in N(u)$ pour tout tel voisin u. Dans la suite, nous nous référons à w comme *faiblement défendu* par u, faiblement défendu à cause de v, ou simplement faiblement défendu lorsque le contexte est clair. Observons qu'un sommet faiblement défendu possède exactement un voisin sécurisé. Ces notions sont illustrées dans la Figure 5.5 (b).

Dans cette section, nous montrons plusieurs propriétés structurelles importantes d'une wrd-fonction qui seront utilisées par nos algorithmes.
Etant donné un graphe $G = (V, E)$ et un sous-ensemble de sommets $V' \subseteq V$, on définit une fonction légion $\chi^{V'}$ comme la fonction indicatrice du sous-ensemble V' :

$$\chi^{V'}(x) = \begin{cases} 1 & \text{si } x \in V' \\ 0 & \text{sinon.} \end{cases}$$

Lemme 5.9. *Soient $G = (V, E)$ un graphe, f une wrd-fonction de G de poids minimum, et V_f son ensemble induit. Alors V_f^2 est un ensemble dominant minimum des sommets non défendus-sécurisés par χ^{V_f}.*

Démonstration. Soit $u \in V - V_f$ un sommet non défendu-sécurisé par χ^{V_f}. Rappelons que u est non défendu-sécurisé par χ^{V_f} si pour tout $u' \in N_{V_f}(u)$ la fonction légion $\chi^{V_f}_{u' \to u}$ contient un sommet non défendu. Ainsi, pour tout sommet $u' \in N_{V_f}(u)$, il existe un sommet u'' faiblement défendu à cause de u. En particulier, cela signifie que $\{u', u''\} \in E$ et $\{u, u''\} \notin E$. Nous montrons le Lemme 5.9 à travers les affirmations ci-dessous.

Propriété 5.10. *V_f^2 est un ensemble dominant des sommets non défendus-sécurisés par χ^{V_f}.*

Démonstration. Supposons par contradiction qu'il existe un sommet $u \in V - V_f$ non défendu-sécurisé par χ^{V_f} tel que $N_{V_f^2}(u) = \emptyset$. Soit u'' un sommet faiblement défendu à cause de u, et soit u' le voisin commun de u et u'' dans V_f. Rappelons que $N(u'') \cap V_f = \{u'\}$, sans quoi u'' serait défendu par $\chi^{V_f}_{u' \to u}$. De plus, nous savons par hypothèse que $f(u') = 1$. Ainsi, le sommet u'' est non-défendu par $\chi^{V_f}_{u' \to u}$, ce qui contredit le fait que f est une wrd-fonction. □

Propriété 5.11. *V_f^2 est un ensemble dominant minimal des sommets non défendus-sécurisés par χ^{V_f}.*

Démonstration. Supposons par contradiction qu'il existe $u \in V_f^2$ tel que $V_f^2 - \{u\}$ est un ensemble dominant des sommets non défendus-sécurisés par χ^{V_f}. Nous affirmons que la fonction légion f_u définie par :

$$f_u(x) = \begin{cases} 1 \text{ si } x = u \\ f(v) \text{ sinon} \end{cases}$$

est une wrd-fonction. Pour voir cela, nous observons que puisque $V_f^2 - \{u\}$ est un ensemble dominant des sommets non défendus-sécurisés par χ^{V_f}, tout sommet de $N_{V-V_f}(u)$ est défendu-sécurisé par f_u. Il vient que f_u est une wrd-fonction avec $cost(f_u) < cost(f)$, une contradiction. □

Pour achever notre preuve du Lemme 5.9, nous déduisons du fait que f est une wrd-fonction de poids minimum, et des affirmations 5.10 et 5.11 que V_f^2 est un ensemble dominant minimum des sommets non défendus-sécurisés par χ^{V_f}. □

Nous concluons cette Sous-Section en montrant que, étant donné un ensemble dominant V' d'un graphe $G = (V, E)$, une wrd-fonction peut être obtenue en calculant un ensemble dominant de l'ensemble $\overline{D}$ de tous les sommets non défendus-sécurisés par $\chi^{V'}$.

Lemme 5.12. *Soit $V' \subseteq V$ un ensemble dominant du graphe $G = (V, E)$, et soit S un ensemble dominant des sommets $\overline{D}$ non défendus-sécurisés par $\chi^{V'}$. Alors la fonction $f : V \to \{0, 1, 2\}$ définie par*

$$f(x) = \begin{cases} 2 & si\ x \in (V' \cap S) \\ 1 & si\ x \in (V' \cup S) - (V' \cap S) \\ 0 & sinon \end{cases}$$

est une wrd-fonction.

Démonstration. Soit S un ensemble dominant de $\overline{D}$ in G. Observons dans un premier temps que puisque $V_f = V' \cup S$, et puisque V' est un ensemble dominant, alors il en est de même de V_f. Nous montrons maintenant que l'ensemble $\overline{D}'$ des sommets non défendus-sécurisés par f est vide. Observons que puisque $V' \subseteq V_f$, nous avons $\overline{D}' \subseteq \overline{D} - S$. Supposons par contradiction que $\overline{D}' \neq \emptyset$, et soit $x \in \overline{D}'$. Nous distinguons deux cas :

(i) Si $N(x) \cap (V' \cap S) \neq \emptyset$ alors x a un voisin de f-value 2, et donc x est défendu-sécurisé, contredisant le choix de x.

(ii) Sinon, par définition de V' et S, x a un voisin $y \in S$ qui n'appartient pas à V'. Nous affirmons que la fonction légion $f_{y \to x}$ ne peut contenir aucun sommet non défendu. En effet, puisque y n'appartient pas à l'ensemble dominant originel V', tous les sommets sont défendus par V' dans $f_{y \to x}$ (rappelons que tout sommet v de V' satisfait $f(v) \geqslant 1$).

Ces deux cas impliquent que $\overline{D}'$ est vide, et donc f est une wrd-fonction. □

5.4.3 Algorithmes Exacts

Nous décrivons à présent nos algorithmes exacts exponentiels pour résoudre le problème Domination Romaine Faible. Notons que ce problème peut trivialement être résolu en temps $O^*(3^n)$ en énumérant toutes les 3-partitions des sommets possibles, ce qui constitue la meilleure borne de temps d'exécution connue jusqu'à présent. Nous présentons premièrement un algorithme en temps $O^*(2^n)$ et espace exponentiel, et deuxièmement un algorithme en temps $O^*(2.2279^n)$ et espace polynomial.

En Espace Exponentiel

Nous montrons qu'une wrd-fonction de poids minimum peut être calculée en temps $O^*(2^n)$ et espace exponentiel. En s'appuyant sur le Lemme 5.9, une wrd-fonction f de poids minimum peut être obtenue en devinant d'abord son ensemble induit V_f et en calculant ensuite un ensemble dominant minimum $V_f^2 \subseteq V_f$ des sommets non défendus-sécurisés par χ^{V_f}. Déterminer un tel ensemble V_f^2 est effectué par une étape de *Pré-Traitement* elle-même réalisée par *Programmation Dynamique Classique* inspirée par celle donnée en [143]. Cette étape de pré-traitement requiert un espace exponentiel, qui sera par la suite réduit en espace polynomial. Cependant, au lieu de garantir que $V_f^2 \subseteq V_f$, l'étape de pré-traitement construit un ensemble dominant de poids minimum V_f^2 des sommets non défendus-sécurisés par χ^{V_f} sans contrainte, *i.e.* on s'autorise $V_f^2 \subseteq V$. Nous montrons dans le Lemme 5.13 la validité de cette approche. Décrivons dans un premier temps cette étape de pré-traitement ; la preuve de celle-ci est établie après la description de l'algorithme principal.

Soit $G = (V, E)$ un graphe du problème Domination Romaine Faible, et soit $V = \{v_1, v_2, \ldots, v_n\}$. Pour chaque sous-ensemble $X \subseteq V$ nous débutons par calculer un ensemble dominant minimum Y de X dans G, i.e. un sous-ensemble $Y \subseteq V$ tel que $X \subseteq N[Y]$. Ceci est réalisé par programmation dynamique : pour chaque sous-ensemble X et chaque entier k ($1 \leqslant k \leqslant n$), $\mathsf{DS}[X, k]$ dénote un

Algorithme 5.3: Etape de pré-traitement de l'algorithme.

for $k = 0$ **to** n **do**
  $\mathsf{DS}[\emptyset, k] = \emptyset$;
foreach $X \subseteq V$ *s.t.* $|X| \geq 1$ **do**
  $\mathsf{DS}[X, 0] = \{\infty\}$;
  // L'ensemble $\{\infty\}$ est une sentinelle utilisée pour établir la non-existence d'un ensemble Y_k qui domine un ensemble non-vide X; sa cardinalité est définie à ∞.
foreach $X \subseteq V$ *dans l'ordre croissant de leur taille* **do**
  for $k = 1$ **to** n **do**

$$\mathsf{DS}[X, k] = \left\{ \begin{array}{l} \text{un ensemble de taille minimum choisi parmi} \\ \mathsf{DS}[X, k-1] \text{ et } \{v_k\} \cup \mathsf{DS}[X - N[v_k], k-1]. \end{array} \right\}$$

ensemble dominant minimum Y_k de X tel que $Y_k \subseteq \{v_1, v_2, \ldots, v_k\}$, s'il existe. L'algorithme 5.3 calcule une table de correspondance DS par programmation dynamique.

Algorithme Principal. Les étapes essentielles de notre algorithme exact sont décrite dans l'Algorithme 5.4. Pour chaque sous-ensemble $V' \subseteq V$, nous vérifions d'abord si $\chi^{V'}$ est (déjà) une wrd-fonction, *i.e.*, si l'ensemble $\overline{D}$ des sommets non défendus-sécurisés par $\chi^{V'}$ est vide. Si ce n'est pas le cas, nous devons calculer l'ensemble V'^2_f. L'étape de pré-traitement garantit que $S = \mathsf{DS}[\overline{D}, n]$ est un ensemble dominant minimum de $\overline{D}$. Si S est un sous-ensemble de V', alors une wrd-fonction f peut être calculée par le Lemme 5.12 ; sinon le Lemme 5.13 garantit qu'il n'existe pas d'autre ensemble induit V'', qui soit mieux que V'.

Lemme 5.13. *Soit $V'_1 \subseteq V$ un ensemble dominant du graphe $G = (V, E)$ et soit S_1 un ensemble dominant minimum de l'ensemble $\overline{D_1}$ de tous les sommets non défendus-sécurisés par $\chi^{V'_1}$. Supposons que $S_1 \nsubseteq V'_1$. Alors il existe un super-ensemble $V'_2 \supset V'_1$ tel que pour n'importe quel ensemble dominant minimum S_2 de $\overline{D_2}$ des sommets non défendus-sécurisés par $\chi^{V'_2}$, nous avons $cost(f_2) \leq cost(f_1)$, où f_i ($i \in \{1, 2\}$) est une fonction légion définie par :*

$$f_i(x) = \left\{ \begin{array}{ll} 2 & \textit{si } x \in (V'_i \cap S_i) \\ 1 & \textit{si } x \in (V'_i \cup S_i) - (V'_i \cap S_i) \\ 0 & \textit{sinon} \end{array} \right.$$

Démonstration. Supposons qu'il existe trois ensembles V'_1, S_1 et $\overline{D_1}$ comme stipulé dans le lemme et supposons que $S_1 \nsubseteq V'_1$. Soit $V'_2 = V'_1 \cup S_1$. Puisque $S_1 \nsubseteq V'_1$, il vient $V'_2 \supset V'_1$. Soit $\overline{D_2}$ l'ensemble des sommets non défendus-sécurisés par $\chi^{V'_2}$. Observons que $\overline{D_2} \subseteq \overline{D_1}$, puisque $V'_1 \subset V'_2$. Par le Lemme 5.12, nous savons qu'une fonction légion f_1 est en fait une wrd-fonction. Ainsi, par le Lemme 5.9, nous avons également que $(V'_1 \cap S_1)$ est un ensemble dominant de $\overline{D_1}$, et donc de $\overline{D_2}$. Dénotons S_2 un ensemble dominant minimum de $\overline{D_2}$. Alors $|S_2| \leqslant |V'_1 \cap S_1|$. Considérons maintenant une fonction légion f_2 comme définie dans le lemme. Par le Lemme 5.12, nous savons que f_2 est une wrd-fonction. Finalement, puisque $|V'_2| = |V'_1| + |S_1 - V'_1|$ et $|S_2| \leq |V'_1 \cap S_1|$, nous concluons notre preuve par la relation $cost(f_1) = |V'_1| + |S_1| = |V'_1| + |S_1 - V'_1| + |S_1 \cap V'_1| \geq |V'_2| + |S_2| = cost(f_2)$. □

Algorithme 5.4: Algorithme du Théorème 5.14 en temps et espace $O^*(2^n)$ pour la résolution exacte du problème Domination Romaine Faible.

```
1  foreach ensemble dominant V' ⊆ V do
2      foreach v ∈ V do
3          Soit f(v) = 1 si v ∈ V', et f(v) = 0 sinon;
4      Calculer l'ensemble D̄ des sommets non défendus-sécurisés par χ^{V'};
5      if D̄ ≠ ∅ then
6          S = DS[D̄, n];
7          if S ⊆ V' then
8              foreach v ∈ S do
9                  Incrémenter f(v) = f(v) + 1;
10 return La wrd-fonction f calculé de poids minimum;
```

Validité. La validité de l'étape de pré-traitement est basée sur les arguments de [143]. Si l'ensemble X est vide alors l'initialisation $\mathsf{DS}[\emptyset, k] = \emptyset$, pour chaque $0 \leq k \leq n$, est clairement correcte. Si l'ensemble X est non vide mais qu'aucun sommet ne peut être utilisé pour dominer X (i.e. $k = 0$), alors $\mathsf{DS}[X, 0]$ est défini à $\{\infty\}$ comme sentinelle, signifiant qu'il n'existe pas d'ensemble Y (avec $Y = \emptyset$) qui puisse dominer X. La taille de $\{\infty\}$ est donnée à ∞. Finalement le calcul de $\mathsf{DS}[X, k]$ est effectué par une formule de récurrence : soit $v_k \notin \mathsf{DS}[X, k]$ ou $v_k \in \mathsf{DS}[X, k]$ et dans le dernier cas, $N(v_k)$ est dominé par v_k. Puisque les ensembles X sont considérés dans l'ordre croissant de leur taille tout comme les valeurs de k, nous pouvons observer que les valeurs $\mathsf{DS}[X, k-1]$ et $\mathsf{DS}[X - N[v_k], k-1]$ ont déjà été calculées lorsque survient le calcul de $\mathsf{DS}[X, k]$.
Nous montrons maintenant la validité de l'algorithme 5.4. L'algorithme énumère tous les ensembles V' comme possible candidats pour l'ensemble induit V_f. En particulier, nous éliminons les ensembles V' qui ne forment pas un ensemble dominant. Par le Lemme 5.9, il suffit de calculer un ensemble dominant $S \subseteq V'$ des sommets $\overline{D}$ qui sont non défendus-sécurisés par $\chi^{V'}$. Le Lemme 5.13 montre que si S n'est pas inclus dans V', alors il existe un super-ensemble strict de V' qui donne une wrd-fonction de poids pas supérieur à celui obtenu depuis V' et S, par le Lemme 5.12. Soit $V'_0 = V'$ et $S_0 = S$. Puisque le graphe est fini et que le super-ensemble donné par le Lemme 5.13 est strict, il existe un entier $\ell \leq n$ fini et une séquence $V'_0 \subset V'_1 \subset ... \subset V'_\ell \subseteq V$ tels que $S_i \not\subseteq V'_i$, pour chaque $0 \leq i < \ell$, et $S_\ell \subseteq V'_\ell$. Du fait que l'algorithme énumère tous les super-ensembles de V', il vient que l'ensemble V'_ℓ sera évalué à une certaine itération de la boucle for. Cela montre la validité de l'algorithme 5.4.

Complexité. L'étape de pré-traitement calcule et considère une valeur dans DS pour chaque sous-ensemble $X \subseteq V$ et pour tout entier $k \in \mathbb{N}^*_{\leq n}$. Pour chaque couple (X, k), l'algorithme récupère les valeurs $\mathsf{DS}[X, k-1]$ et $\mathsf{DS}[X - N[v_k], k-1]$ préalablement calculées afin de stocker la valeur $\mathsf{DS}[X, k]$. Cette étape de pré-traitement requiert en conséquence un temps et espace exponentiel $O^*(2^n)$. La partie principe de l'Algorithme 5.4 chaque ensemble dominant $V' \subseteq V$, et calcule en temps polynomial l'ensemble $\overline{D}$ des sommets non défendus-sécurisés par $\chi^{V'}$. Un ensemble dominant S de $\overline{D}$ est alors obtenu en récupérant dans la table DS, en temps constant, la valeur $\mathsf{DS}[\overline{D}, n]$ précédémment calculée.

Théorème 5.14. *L'Algorithme 5.4 résout le problème de Domination Romaine Faible en temps et espace $O^*(2^n)$.*

En Espace Polynomial

Dans le but de construire un algorithme exact exponentiel pour le problème Domination Romaine Faible qui n'utilise un espace mémoire que polynomial, nous devons interdire l'utilisation de l'espace exponentiel engendré par l'étape de pré-traitement de l'Algorithme 5.4. Pour cela, nous utilisons à la place un algorithme de résolution exacte en espace polynomial pour résoudre le problème Ensemble Dominant Rouge-Bleu, et qui décide quels sommets $v \in V$ devront satisfaire $f(v) = 2$ pour dominer les sommets non défendus-sécurisés. Le problème Ensemble Dominant Rouge-Bleu se présente comme le problème Ensemble Dominant Rouge-Bleu Connexe que nous avons rencontré en Sous-Section 5.2.3, la propriété de connexité en moins.

Ensemble Dominant Rouge-Bleu
Version : *Optimisation*

Entrée : $G = (R \cup B, E)$ un graphe biparti.
Sortie : Déterminer $S \subseteq R$ de plus petite taille possible qui domine B.

Théorème 5.15 ([183]). *L'Algorithme 5.5 résout le problème de* Ensemble Dominant Rouge-Bleu *en temps* $O^*(1.2279^{|R|+|B|})$ *et espace polynomial.*

Algorithme. Observons qu'avant de calculer un ensemble dominant rouge-bleu minimum du graphe biparti $(\overline{C} \cup \overline{D}, E)$, nous pouvons modifier les ensembles $\overline{C}$ et $\overline{D}$ comme suit : pour tout sommet $v \in \overline{C}$, si v possède au moins deux sommets faiblement défendus dans son voisinage, alors nous posons $f(v) = 2$, et nous supprimons v de $\overline{C}$ et $N_{\overline{D}}(v)$ de $\overline{D}$.

Proposition 5.16. *L'étape précédente, dite de nettoyage de* $\overline{C}$ *et* $\overline{D}$ *ne modifie pas une solution au problème Ensemble Dominant Rouge-Bleu sur l'instance* $I = (\overline{C} \cup \overline{D}, E)$.

Démonstration. Soit $v \in \overline{C}$ un sommet sécurisé avec au moins deux sommets w_1 et w_2 faiblement défendus dans son voisinage. Puisque w_1 et w_2 sont faiblement défendus, leur seul voisin sécurisé est $v \in V'$; puisqu'ils ne sont pas défendus-sécurisés, ils ont besoin d'être dominés par V_f^2 pour que f devienne une wrd-fonction (Lemme 5.9). En conséquence, nous devons imposer $f(v) = 2$. Il vient que pour tout ensemble rouge-bleu dominant minimum de l'instance $I = (\overline{C} \cup \overline{D}, E)$, nous devons avoir $v \in \overline{C}$ dans l'ensemble dominant rouge-bleu dans le but de dominer tous les voisins non défendus-sécurisés de v dans $\overline{D}$.
Maintenant, observons que puisque les voisins de v sont défendus-sécurisés, car dominés par V_f^2, alors ils peuvent sereinement être retirés de $\overline{D}$. Il vient que, puisque v ne possède plus de sommet non défendus-sécurisés dans son voisinage, nous pouvons supprimer v de $\overline{C}$. □

Validité. La validité de l'algorithme se déduit du Lemme 5.9 et de la preuve de validité de l'Algorithme 5.4. La différence principale tient dans le calcul de l'ensemble dominant des sommets non défendus-sécurisés par $\chi^{V'}$. En effet, dans ce cas, nous utilisons le Théorème 5.15 pour déterminer les sommets de V' qui doivent avoir pour valeur f égal à 2 dans le but de dominer les sommets non défendus-sécurisés par $\chi^{V'}$. La validité de cette étape est déduite du Lemme 5.9 et de la Proposition 5.16.
Complexité. Il est clair que pour tout sous-ensemble $V' \subseteq V$, l'initialisation de $f(x)$ pour tout $x \in V$, de même que le calcul de l'ensemble $\overline{D}$, et de même que l'étape de nettoyage, peut être effectué en temps et espace polynomial.
Pour une fonction légion f en cours de construction donnée, pour tout $V' \subseteq V$, notre algorithme construit et réduit l'ensemble $\overline{D}$ des sommets non défendus-sécurisés par $\chi^{V'}$, et l'ensemble $\overline{C}$ des

Algorithme 5.5: Un algorithme en temps $O^*(2.2279^n)$ et espace polynomial pour résoudre le problème Domination Romaine Faible.

```
foreach ensemble dominant V' ⊆ V do
    foreach v ∈ V do
        f(v) = 1 si v ∈ V', f(v) = 0 sinon;
    Calculer l'ensemble D̄ des sommets non défendus-sécurisés par χ^{V'};
    if D̄ ≠ ∅ then
        Calculer l'ensemble C̄ ⊆ V' des sommets sécurisés qui ont au moins un voisin dans D̄;
        /* Étape de nettoyage */
        foreach v ∈ C̄ avec au moins deux voisins non défendus-sécurisés dans D̄ do
            f(v) = 2;
            Supprimer N_{D̄}(v) de D̄;
            Supprimer v de C̄;
        Soit I = (C̄ ∪ D̄, E) une instance du problème Ensemble Dominant Rouge-Bleu;
        if I admet un ensemble dominant rouge-bleu minimum S ⊆ C̄ then
            f(v) = 2 pour tout v ∈ S;
        else
            La fonction courante f ne peut être modifiée en wrd-fonction;
return La wrd-fonction calculée f de poids minimum;
```

sommets sécurisés ayant au moins un voisin dans $\overline{D}$. Ces deux ensembles forment une instance du problème Ensemble Dominant Rouge-Bleu qui peut être résolu en temps $O^*(1.2279^{|\overline{D}|+|\overline{C}|})$ et espace polynomial, en utilisant l'algorithme de van Rooij [183]. Nous avons besoin de la Proposition 5.17 suivante :

Proposition 5.17. *Pour tout $V' \subseteq V$, $|\overline{D}| + |\overline{C}| \leq |V| - |V'|$.*

Démonstration. Pour tout sommet $v \in V'$, une de ces propriétés est vérifiée :

(i) v n'a pas de voisin dans $\overline{D}$, c'est-à-dire aucun voisin non défendu-sécurisé par $\chi^{V'}$;

(ii) il y a au moins un sommet $w \in V - V'$ faiblement défendu par v.

Observons que les cas (i) et (ii) sont les seuls possibles. En effet, s'il existe $v \in V'$ tel que $N_{\overline{D}}(v) \neq \emptyset$ mais aucun sommet de $V - V'$ n'est faiblement défendu par v, alors les sommets de $N_{\overline{D}}(v)$ sont défendus-sécurisés, ce qui est une contradiction.
Si la première propriété est vérifiée, alors v n'est pas inclus dans $\overline{C}$. Si la deuxième propriété est satisfaite, alors soit w est défendu-sécurisé, ou alors w n'est pas défendu-sécurisé. Si w est défendu-sécurisé (c'est-à-dire aucun voisin de w n'est faiblement défendu par v), alors w n'est pas inclus dans $\overline{D}$. Si w n'est pas défendu-sécurisé, alors v possède au moins deux voisins faiblement et non défendus-sécurisés. En effet, puisque w est faiblement défendu par w, v est le seul voisin de w dans V'. En conséquence, il existe un ensemble non vide $\overline{D}_{v,w} = N_{\overline{D}}(v) - N(w)$ tel que w est faiblement défendu à cause de chaque sommet dans $\overline{D}_{v,w}$. Maintenant, puisque w est non défendu-sécurisé par v, alors il doit exister un sommet $w' \in \overline{D}_{v,w}$ qui est également faiblement défendu à cause de w. Alors l'étape de nettoyage sur $\overline{D}$ et $\overline{C}$ s'applique, ce qui implique que v est retiré de $\overline{C}$ et tous ses voisins (dont w) sont supprimés de $\overline{D}$. Tout ensemble, pour tout sommet $v \in V'$, au moins un sommet de V n'est pas inclus dans $\overline{C} \cup \overline{D}$, et donc au moins $|V'|$ sommet de V ne sont pas inclus dans $\overline{D} \cup \overline{C}$. □

Le temps d'exécution total de l'algorithme $T(n)$ est alors donné par :

$$T(n) = O^*(\sum_{i=1}^{n} \binom{n}{i} \cdot P(n-i)) = O^*(\sum_{i=1}^{n} \binom{n}{i} \cdot 1.2279^{n-i}) = O^*(2.2279^n)$$

où $P(p)$ est le temps d'un algorithme pour construire un ensemble dominant rouge-bleu minimum d'un graphe à p sommets [183].

Théorème 5.18. *L'Algorithme 5.5 résout le problème* Domination Romaine Faible *en temps* $O^*(2.2279^n)$ *et espace polynomial.*

5.5 Conclusion

Dans ce chapitre, nous avons étudié différents problèmes de graphes. Pour chacun de ces problèmes, nous avons proposé différentes alternatives algorithmiques selon l'environnement de contraintes au sein duquel nous étudions le problème.

Le premier problème que nous avons étudié est le problème Ensemble Tropical Connexe, pour lequel nous avons construit un algorithme exact en temps $O^*(1.5359^n)$, ainsi qu'un algorithme en temps $O^*(1.2721^n)$ lorsqu'on se restreint aux arbres. Ces algorithmes sont d'autant plus pertinents que sous une hypothèse de complexité plus forte que **ETH**, il n'y a aucun algorithme en temps sous-exponentiel pour résoudre ce problème même lorsqu'on se restreint aux arbres.

Le problème Ensemble Tropical Connexe est très fortement relié au problème de Graphe Motif. Nous pensons que les algorithmes de la Section 5.3 peuvent être adaptés pour d'autres variantes de Graphe Motif.

Peut-on adapter nos algorithmes pour le problème Ensemble Tropical Connexe *pour résoudre d'autres variantes de* Graphe Motif *?*

Nous avons donné une borne inférieure sur le nombre d'ensembles tropicaux connexes que peut admettre un graphe colorié. Cette borne donnée en Section 5.3 est obtenue pour la classe des arbres, et pour cette classe cette borne est optimale. Cependant, nous ne savons pas s'il y a une famille infinie de graphes coloriés qui admettent un nombre supérieur de tels sous-ensembles de sommets.

Existe-t-il un graphe colorié qui admet plus que $3^{n/3}$ *ensembles tropicaux connexes ?*

Le deuxième problème que nous avons étudié est le problème Domination Romaine Faible, pour lequel nous avons proposé des algorithmes de résolution respectivement en temps $O^*(2.23^n)$ avec espace polynomial et $O^*(2^n)$ avec espace exponentiel. Nous avons vu que ce problème ajoute une dynamique de mouvement au problème classique d'Ensemble Dominant. En effet pour des légions romaines, il s'agit non seulement de protéger un ensemble de stations, mais de conserver cette propriété de protection si une des stations venait à être attaquée, c'est-à-dire si une légion venait à se mouvoir au long d'une arête. Nous pouvons généraliser ce problème en introduisant naturellement deux paramètres $\sigma, \rho \in \mathbb{N}$ supplémentaires.

DOMINATION ROMAINE (σ, ρ)-FAIBLE

VERSION : *OPTIMISATION*

Entrée : Un graphe $G = (V, E)$, $\sigma, \rho \in \mathbb{N}$

Sortie : Déterminer un nombre minimum de légions à attribuer aux sommets de V de sorte que le graphe G soit protégé, et qu'il le demeure après ρ mouvements de troupes consécutifs, où à chaque mouvement au plus σ stations sont attaquées.

En introduisant ce problème généralisé DOMINATION ROMAINE (σ, ρ)-FAIBLE, nous soulèvons un nouveau pan d'études pour chaque champ disciplinaire ou spécialité de l'INFORMATIQUE THÉORIQUE.

Algorithme TCS-Tree(S, F, D)

Règles de Réduction.

R0.1. Si $C' = \emptyset$: STOP, S est un ensemble tropical connexe de T.

R0.2. Si $F = \emptyset$: STOP, cette instance n'admet pas de solution.

R1. S'il existe $v \in F$, v feuille de T', tel que $\varphi(v) \notin C'$: $\texttt{RMV}(v)$.

R2. S'il existe $v \in F$ tel que $c(v) \in C'$ et tel que $\forall\ u \in F - \{v\}$, $\varphi(v) \neq \varphi(u)$: $\texttt{ADD}(v)$.

R3. S'il existe $v \in F$, v feuille de T', tel qu'il existe $u \in F - \{v\}$ vérifiant $\varphi(u) = \varphi(v)$ et $d(u, S) = 1$: $\texttt{RMV}(v)$.

R4. S'il existe $v \in F$, v feuille de T', tel qu'il existe $u \in F - \{v\}$ vérifiant $\varphi(u) = \varphi(v)$ et $d(u, v \rightsquigarrow r) \leq 1$: $\texttt{RMV}(v)$.

R5. S'il existe $u \in F$, tel qu'il existe une feuille v de T', $v \in S$, vérifiant $\varphi(u) = \varphi(v)$: $\texttt{RMV}(u)$.

R6. Si tous les sommets libres de F sont des feuilles dans T' : $\texttt{ADD}(v)$, pour $v \in F$ quelconque.

Règles de Branchement. Chaque règle de branchement de l'algorithme crée deux sous-problèmes. Pour une règle de branchement, nous écrivons $\langle \mathcal{O}_1 \mid\mid \ldots \mid\mid O_p \rangle$ pour exprimer le fait que l'algorithme crée p instances de sous-problèmes. Le i-ème sous-problème est obtenu en appliquant sur l'instance (S, F, D) les opérations décrites dans l'ensemble $\mathcal{O}_i$.

B1. S'il existe une feuille libre $v \in F$ de T' vérifiant $d(v, S) \geq 4$, alors nous distinguons trois cas.

(a) S'il existe deux nœuds internes libres $v', v'' \in F$ vérifiant $\varphi(v) = \varphi(v') = \varphi(v'')$:
$\langle \{\texttt{ADD}(v), \texttt{RMV}(\{v', v''\})\} \mid\mid \texttt{RMV}(v) \rangle \quad (\mathbf{7}, \mathbf{1})$

(b) S'il existe une feuille libre $v' \in F - \{v\}$ de T' vérifiant $\varphi(v) = \varphi(v')$:
$\langle \{\texttt{ADD}(p(v)), \texttt{RMV}(v')\} \mid\mid \texttt{RMV}(p(v)) \rangle \quad (\mathbf{4}, \mathbf{2})$

(c) S'il existe un unique nœud interne libre $v' \in F$ de T' vérifiant $\varphi(v) = \varphi(v')$:
$\langle \{\texttt{ADD}(v), \texttt{RMV}(v')\} \mid\mid \{\texttt{RMV}(v), \texttt{ADD}(v')\} \rangle \quad (\mathbf{6}, \mathbf{3})$

B2. S'il existe une feuille libre $v \in F$ de T' vérifiant $d(v, S) = 3$, alors nous distinguons quatre cas.

(a) S'il existe deux feuilles libres distinctes $v', v'' \in F - \{v\}$ de T' vérifiant $\varphi(v) = \varphi(v') = \varphi(v'')$:
$\langle \{\texttt{ADD}(p(v)), \texttt{RMV}(\{v', v''\})\} \mid\mid \texttt{RMV}(p(v)) \rangle \quad (\mathbf{4}, \mathbf{2})$

(b) S'il existe deux nœuds internes libres $v', v'' \in F$ de T' vérifiant $\varphi(v) = \varphi(v') = \varphi(v'')$:
$\langle \{\texttt{ADD}(v), \texttt{RMV}(\{v', v''\})\} \mid\mid \texttt{RMV}(v) \rangle \quad (\mathbf{7}, \mathbf{1})$

(c) S'il existe une feuille libre $v' \in F - \{v\}$ et un nœud interne libre $v'' \in F$ de T' vérifiant $\varphi(v) = \varphi(v') = \varphi(v'')$:
$\langle \{\texttt{ADD}(v''), \texttt{RMV}(\{v, v'\})\} \mid\mid \texttt{RMV}(v'') \rangle \quad (\mathbf{4}, \mathbf{2})$

(d) S'il existe un unique sommet libre $v' \in F$ de T' vérifiant $\varphi(v) = \varphi(v')$:
$\langle \{\texttt{ADD}(v), \texttt{RMV}(v')\} \mid\mid \{\texttt{RMV}(v), \texttt{ADD}(v')\} \rangle \quad (\mathbf{4}, \mathbf{3})$

B3. S'il existe une feuille libre $v \in F$ de T' vérifiant $d(v, S) = 2$, alors nous distinguons deux cas.

(a) S'il existe $k \geq 2$ sommets libres distincts $v_1, \ldots, v_k \in F - \{v\}$ de T' vérifiant $\varphi(v) = \varphi(v_1) = \ldots = \varphi(v_k)$:
$\langle \{\texttt{ADD}(p(v), v), \texttt{RMV}(\{v_1, \ldots, v_k\})\} \mid\mid \texttt{RMV}(p(v)) \rangle \quad (\mathbf{4}, \mathbf{2})$

(b) S'il existe un unique sommet libre $v' \in F - \{v\}$ de T' vérifiant $c(v) = c(v')$:
$\langle \{\texttt{ADD}(\{p(v), v\}), \texttt{RMV}(v')\} \mid\mid \{\texttt{RMV}(p(v)), \texttt{ADD}(v')\} \rangle \quad (\mathbf{3}, \mathbf{4})$

Algorithme 5.2 – Algorithme de Branchement du Théorème 5.6 pour résoudre le problème Ensemble Tropical Connexe sur les arbres. Le plus grand nombre de branchement est $\tau(4, 2) \leq 1.2721$.

Conclusion

Cette Thèse s'inscrit dans la résolution exacte, mais également paramétrée, de problèmes NP-difficiles, mais également de problèmes se présentant comme plus difficiles encore, qui concerne les graphes ou les hypergraphes. Dans un premier temps, rappelons brièvement nos résultats.

Dans le Chapitre 2 de cette Thèse, nous nous sommes intéressés à la résolution exacte de différents problèmes de coloration. Chacun de ces problèmes peut se présenter comme une variante du problème classique du Nombre Chromatique. Ces problèmes diffèrent les uns des autres par l'introduction d'une contrainte plus forte que doivent satisfaire les classes de couleurs entre elles, ou par une difficulté accrue d'identification d'une classe de couleur potentielle. Comme nous l'avons vu, les problèmes de coloration de graphes ont souvent un lien avec des notions ou des problèmes classiques d'hypergraphes.
Il existe des méthodes standards pour résoudre le problème du Nombre Chromatique. Ces techniques majeures sont la *Programmation Dynamique Classique*, le principe d'*Inclusion-Exclusion*, ou encore pour certaines classes de graphes la *Programmation Dynamique Arborescente*. Chacune de ces méthodes a déjà pu être appliquée efficacement pour résoudre ce problème. Nous avons donc tenté d'étendre le champ d'application de ces méthodes aux problèmes que nous avons étudiés.
Nous avons proposé des algorithmes de *Programmation Dynamique Classique* respectivement en temps $O^*(2.4423^n)$ avec espace $O^*(2^n)$ et en temps $O^*(4^n)$ avec espace $O^*(3^n)$ pour résoudre les problèmes du Nombre Grundy et Nombre Grundy Partiel. Nous avons construit des algorithmes de *Programmation Dynamique Arborescente* pour obtenir les premiers algorithmes en temps *single-exponentiel* sur certaines classes de graphes pour les problèmes du Nombre a-Chromatique, Nombre Pseudo-a-Chromatique, Nombre Harmonieux Chromatique, ou du Nombre Pseudo-Harmonieux Chromatique. En particulier, notre algorithme calcule le nombre a-chromatique d'un arbre en temps $O^*(2^n)$ ou d'un graphe planaire en temps $O^*(8^n)$. Finalement, nous avons pu étendre le principe d'*Inclusion-Exclusion* pour obtenir des algorithmes respectivement en temps $O^*(3^n)$ avec espace $O^*(2^n)$ et en temps et espace $O^*(2^n)$ pour les problèmes du Nombre b-Chromatique ou du Nombre Clique-Chromatique.

Dans le Chapitre 3, nous nous sommes intéressés aux différentes versions du problème de transversal d'hypergraphe, problème appelé Hitting Set. Les hypergraphes sont une généralisation naturelle des graphes, le problème Hitting Set est une généralisation naturelle du problème Couverture de Sommets. Nous avons proposé dans ce Chapitre une borne combinatoire, qui montre que plus le rang de l'hypergraphe augmente, moins on a de chance de pouvoir obtenir un algorithme exact exponentiel pour la version énumération du problème, qui soit plus efficace que l'algorithme trivial en temps $O^*(2^n)$. Cependant, nous avons vu qu'il y avait une marge importante entre la meilleure borne inférieure que nous puissions fournir à l'heure actuelle, et le temps d'exécution des algorithmes actuels pour l'énumération des transversaux minimaux d'un hypergraphe de rang borné. Nos résultats dans ce Chapitre se concentrent essentiellement sur des algorithmes plus efficaces pour la

version énumération du problème k-Hitting Set.
Pour $k = 3$, nous avons construit un algorithme de *Branchement*. Par la technique *Mesurer et Conquérir*, nous avons montré que notre algorithme s'exécute en temps $O^*(1.6755^n)$. Pour $k = 4$, nous avons montré comment utiliser l'algorithme de 3-Hitting Set, via la technique de *Compression Iterative*, afin de construire un algorithme en temps $O^*(1.8866^n)$. Pour $k = 5$ et $k = 6$, nous proposons des algorithmes généralisés qui s'exécutent respectivement en temps $O^*(1.9538^n)$ et $O^*(1.9779^n)$. Ces algorithmes généralisés peuvent être appliqués pour tout $k \geq 7$. Tous ces algorithmes améliorent le temps d'exécution des meilleurs algorithmes connus pour la version énumération du problème k-Hitting Set, et peuvent également améliorer dans certains cas le temps d'exécution des meilleurs algorithmes pour d'autres versions du problème, ainsi que pour des versions du problèmes co$k$-Hitting Set.

Dans le Chapitre 4, nous nous sommes intéressés à la résolution à paramètre fixe du problème de Racine Carrée de graphe, problème déjà bien étudié en Théorie des Graphes. Pour un problème, être soluble à paramètre fixe est équivalent à admettre un noyau. Construire un noyau pour un problème consiste à réduire toute instance du problème en une instance du même problème dont la taille ne dépend que du paramètre. Un algorithme paramétré peut alors simplement être l'algorithme trivial sur cette instance réduite. Ainsi, bien que l'efficacité d'un algorithme paramétré soit recherchée, nous souhaitons plus simplement montrer que le temps d'exécution d'un algorithme de résolution peut dépendre exclusivement de la valeur du paramètre. L'Algorithmique Paramétrée tend donc à identifier et confiner la nature de la difficulté de résolution d'un problème dans un paramètre.
Préalablement, nous construisons un algorithme polynomial pour décider si un graphe de degré maximum $\Delta(G) \leq 6$ admet une racine carrée. Puis nous montrons que le problème Racine Carrée Minimum est FPTde paramètre $k \in \mathbb{N}$. Pour obtenir ce résultat, nous construisons un noyau généralisé quadratique pour ce problème, c'est-à-dire que nous réduisons toute instance du problème à une instance d'un problème plus général, et dont la taille est $O(k^2)$. Finalement, nous montrons que le problème Racine Carrée Maximum est également FPT, et nous proposons un algorithme de résolution exacte en temps $O^*(3^{m/3})$ pour ce problème.

En dernier lieu, dans le Chapitre 5, nous construisons des algorithmes exacts pour différents problèmes de graphes. En particulier, nous construisons des algorithmes respectivement en temps $O^*(1.5359^n)$ et $O^*(1.2721^n)$ pour résoudre le problème Ensemble Tropical Connexe sur les graphes en général et sur les arbres. Nous construisons également deux algorithmes respectivement en temps $O^*(2.23^n)$ avec espace polynomial et $O^*(2^n)$ avec espace exponentiel pour résoudre le même problème Domination Romaine Faible sur les graphes en général.

Nous présentons ici quelques pistes de recherches pour poursuivre ces travaux. Nous renvoyons à la Conclusion de chaque Chapitre pour des questions ouvertes en rapport direct avec nos travaux. Nous résumons brièvement des pistes de réflexion intéressantes directement reliées à nos chapitres.
Pour le chapitre 2, nous ne savons pas s'il existe un algorithme de résolution plus efficace que l'algorithme trivial pour des problèmes de coloration acyclique ou coloration étoile. Plus généralement, il semble intéressant d'étudier les problèmes de coloration où chaque paire de classes de couleurs induit un graphe appartenant à une classe donnée en entrée.
Pour le Chapitre 3, il semble qu'il soit possible d'améliorer encore le temps d'exécution des algorithmes présentés en les généralisant à p_k hyperarêtes. L'analyse qui en découle paraît difficile à effectuer, mais nous avons confiance dans le fait que les outils que nous avons développés pour l'étude des algorithmes de *Branchement* pourraient être d'une aide précieuse, tant pour la formalisation que

pour l'analyse des algorithmes. Également, essayer d'adapter nos algorithmes à d'autres versions du problème k-Hitting Set ou au problème co$k$-Hitting Set semble être une idée prometteuse.
En ce qui concerne le Chapitre 4, il reste en suspens beaucoup de questions sur la résolution à paramètre fixe du problème de Racine de Graphes, mais encore plus en ce qui concerne la résolution exacte de chaque variante du problème, que ce soit sur les graphes en général ou sur certaines classes de graphes particulières.
Finalement, le Chapitre 5 laisse la porte ouverte à l'étude de tout un tas de problèmes, qu'ils concernent les graphes ou les hypergraphes, qu'ils se ressemblent ou qu'ils diffèrent tous toujours plus les uns des autres. Notons cependant que l'étude des variantes du problème de Graphe Motif semble être une piste de réflexion fructueuse pour construire des algorithmes exacts exponentiels efficaces.

À présent, nous orientons nos questions sur les techniques d'Algorithmique Exacte Exponentielle. Dans cette Thèse, nous avons vu et mis en œuvre une grande variété de ces techniques. Ces techniques que nous avons présentées en Section 1.5 revêtent une importance majeure mais ont une portée différente. Nos questions concernent alors leurs champs d'application et leurs limites.

La *Programmation Dynamique Classique* est une technique difficile à mettre en œuvre. La technique d'*Inclusion-Exclusion* est une variante qui semble plus facile de mise en œuvre et performante au premier abord, mais dont le champ semble plus limité. Nous avons vu au Chapitre 2 que ce principe permet de construire des algorithmes exacts exponentiels efficaces pour des problèmes de coloration. Une question se pose sur l'existence d'une technique qui améliore ces algorithmes. Un résultat négatif en ce sens est qu'il a été montré en 2012 que sous l'hypothèse de complexité **SETH**, les algorithmes connus de nombreux problèmes ne peuvent être améliorés [53]. Il est conjecturé que sous cette hypothèse, on ne peut résoudre la version décisionnelle du problème Partition d'Ensembles en un temps meilleur que $O^*(2^n)$. L'algorithme d'*Inclusion-Exclusion* de Björklund, Husfeldt, et Koivisto calcule le nombre chromatique d'un graphe en temps $O^*(2^n)$, car il résout dans les faits n'importe quelle instance du problème Partition d'Ensembles. Pour aller plus loin dans la résolution du Nombre Chromatique, il semble alors qu'il faille une toute nouvelle approche.

Peut-on construire un algorithme en temps $O^*(c^n)$, $1 < c < 2$ pour résoudre le problème du Nombre Chromatique *?*

La technique de *Branchement* est une technique plutôt facile à mettre en œuvre. L'analyse d'un algorithme qui utilise cette technique peut quant à elle se réveler très complexe. Les temps d'exécution annoncés des algorithmes de *Branchement* sont des bornes supérieures sur leur véritable temps d'exécution. Dans la pratique, un algorithme peut être bien plus efficace. Il est cependant difficile de voir les conséquences de l'imbrication de règles de branchement successives. En ce sens, la technique de *Mesurer et Conquérir* vient réduire ce vide, mais ne le comble pas totalement. Nous choisissons une mesure différente de l'entrée, mais le temps d'exécution obtenu représente toujours une borne supérieure sur le temps d'exécution réel de l'algorithme. Aussi, nous posons la question de l'analyse du pire des cas d'un algorithme de *Branchement*. Notre technique employée au Chapitre 3 pour construire nos algorithmes généralisés va dans ce sens. Nous évaluons toutes les imbrications successives possibles de deux règles de branchement parmi un ensemble de règles initial. Pour notre problème, nous pensons qu'il est possible d'aller plus loin dans les imbrications consécutives de nos règles. Cependant, nous ne savons pas si notre approche, qui de plus est complexe, peut être appliquée pour un autre problème ou un autre algorithme.

Peut-on, et jusqu'à quel point, améliorer l'analyse des algorithmes de Branchement ?

Bibliographie

[1] F. N. Abu-Khzam, A. E. Mouawad, and M. Liedloff, *An Exact Algorithm for Connected Red-Blue Dominating Set*, Journal of Discrete Algorithms, 9 (2011), pp. 252–262.

[2] A. Adamaszek and M. Adamaszek, *Uniqueness of Graph Square Roots of Girth Six*, Electronic Journal of Combinatorics, 18 (2011).

[3] D. Aingworth, R. Motwani, and F. Harary, *The difference between a graph and its square*, Utilitas Mathematica, 54 (1998), pp. 223–228.

[4] J. Alber, H. Fernau, and R. Niedermeier, *Graph separators : a parameterized view*, Journal of Computer and System Sciences, 67 (2003), pp. 808–832.

[5] ——, *Parameterized complexity : exponential speed-up for planar graph problems*, Journal of Algorithms, 52 (2004), pp. 26–56.

[6] N. Alon, G. Gutin, E. J. Kim, S. Szeider, and A. Yeo, *Solving MAX-r-SATsat Above a Tight Lower Bound*, Algorithmica, 61 (2011), pp. 638–655.

[7] A. Ambalath, R. Balasundaram, C. Rao H., V. Koppula, N. Misra, G. Philip, and M. Ramanujan, *On the Kernelization Complexity of Colorful motifs*, in Parameterized and Exact Computation, vol. 6478 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2010, pp. 14–25.

[8] J.-A. Angles D'Auriac, N. Cohen, A. El Maftouhi, A. Harutyunyan, S. Legay, and Y. Manoussakis, *Connected tropical subgraphs in vertex-colored graphs*.

[9] S. Arnborg, D. Corneil, and A. Proskurowski, *Complexity of Finding Embeddings in a k-Tree*, SIAM Journal on Algebraic Discrete Methods, 8 (1987), pp. 277–284.

[10] S. Arnborg, J. Lagergren, and D. Seese, *Easy problems for tree-decomposable graphs*, Journal of Algorithms, 12 (1991), pp. 308–340.

[11] S. Arora and B. Barak, *Computational complexity : a modern approach*, Cambridge University Press, 2009.

[12] G. Bacsó, S. Gravier, A. Gyárfás, M. Preissmann, and A. Sebö, *Coloring the Maximal Cliques of Graphs*, SIAM Journal on Discrete Mathematics, 17 (2004), pp. 361–376.

[13] J. Bailey, T. Manoukian, and K. Ramamohanarao, *A Fast Algorithm for Computing Hypergraph Transversals and its Application in Mining Emerging Patterns*, in Proceedings of the Third IEEE International Conference on Data Mining, ICDM '03, Washington, DC, USA, 2003, IEEE Computer Society, pp. 485–.

[14] R. Balasubramanian, V. Raman, and V. Yegnanarayanan, *On the pseudoachromatic number of join of graphs*, International Journal of Computer Mathematics, 80 (2003), pp. 1131–1137.

[15] D. Barth, J. Cohen, and T. Faik, *On the b-continuity property of graphs*, Discrete Applied Mathematics, 155 (2007), pp. 1761–1768.

[16] R. Beigel and D. Eppstein, *3-coloring in Time $O(1.3289^n)$*, Journal of Algorithms, 54 (2005), pp. 168–204.

[17] C. Berge, *Hypergraphs : combinatorics of finite sets*, North holland, 1984.

[18] C. Berge and P. Duchet, *Strongly perfect graphs*, Annals of Discrete Mathematics, 21 (1984), pp. 57–61.

[19] N. Betzler, R. Van Bevern, M. R. Fellows, C. Komusiewicz, and R. Niedermeier, *Parameterized algorithmics for finding connected motifs in biological networks*, IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB), 8 (2011), pp. 1296–1308.

[20] A. Björklund, T. Husfeldt, and M. Koivisto, *Set Partitioning via Inclusion-Exclusion*, SIAM Journal on Computing, 39 (2009), pp. 546–563.

[21] H. L. Bodlaender, *Achromatic number is NP-complete for cographs and interval graphs*, Information Processing Letters, 31 (1989), pp. 135–138.

[22] ——, *A Tourist Guide through Treewidth*, Acta Cybernetica, 11 (1993), pp. 1–23.

[23] ——, *A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth*, SIAM Journal on Computing, 25 (1996), pp. 1305–1317.

[24] ——, *A partial k-arboretum of graphs with bounded treewidth*, Theoretical Computer Science, 209 (1998), pp. 1–45.

[25] N. Bonichon, C. Gavoille, N. Hanusse, D. Poulalhon, and G. Schaeffer, *Planar Graphs, via Well-Orderly Maps and Trees*, Graphs and Combinatorics, 22 (2006), pp. 185–202.

[26] F. Bonomo, G. Duran, F. Maffray, J. Marenco, and M. Valencia-Pabon, *On the b-Coloring of Cographs and P_4-Sparse Graphs*, Graphs and Combinatorics, 25 (2009), pp. 153–167.

[27] A. Brandstädt, J. P. Spinrad, et al., *Graph classes : a survey*, vol. 3, SIAM, 1999.

[28] L. Cai and D. Juedes, *On the existence of subexponential parameterized algorithms*, Journal of Computer and System Sciences, 67 (2003), pp. 789–807.

[29] N. Cairnie and K. Edwards, *Some results on the achromatic number*, Journal of Graph Theory, 26 (1997), pp. 129–136.

[30] N. Cairnie and K. Edwards, *The achromatic number of bounded degree trees*, Discrete Mathematics, 188 (1998), pp. 87–97.

[31] M. Cesati, *Compendium of Parameterized Problems*, Department of Computer Science, Systems, and Industrial Engineering, University of Rome "Tor vergata", (2006).

[32] E. Chambers, B. Kinnersley, N. Prince, and D. West, *Extremal Problems for Roman Domination*, SIAM Journal on Discrete Mathematics, 23 (2009), pp. 1575–1586.

[33] M. Chapelle, M. Cochefert, J.-F. Couturier, D. Kratsch, M. Liedloff, and A. Perez, *Exact Algorithms for Weak Roman Domination*, in Combinatorial Algorithms, vol. 8288 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 81–93.

[34] G. Chapuy, É. Fusy, O. Giménez, B. Mohar, and M. Noy, *Asymptotic enumeration and limit laws for graphs of fixed genus*, Journal of Combinatorial Theory, Series A, 118 (2011), pp. 748–777.

[35] M. Chellali, N. Rad, and L. Volkmann, *Some results on Roman domination edge critical graphs*, AKCE International Journal of Graphs and Combinatorics, 9 (2012), pp. 195–203.

[36] J. Chen, I. Kanj, J. Meng, G. Xia, and F. Zhang, *On the Pseudo-achromatic Number Problem*, in Graph-Theoretic Concepts in Computer Science, vol. 5344 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 78–89.

[37] J. Chen, I. Kanj, and G. Xia, *Improved Parameterized Upper Bounds for Vertex Cover*, in Mathematical Foundations of Computer Science 2006, vol. 4162 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 238–249.

[38] J. Chen, Y. Liu, S. Lu, B. O'sullivan, and I. Razgon, *A Fixed-parameter Algorithm for the Directed Feedback Vertex Set Problem*, Journal of the ACM, 55 (2008), pp. 21 :1–21 :19.

[39] C. A. Christen and S. M. Selkow, *Some Perfect Coloring Properties of Graphs*, Journal of Combinatorial Theory, Series B, 27 (1979), pp. 49–59.

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

[41] M. Cochefert, J.-F. Couturier, P. Golovach, D. Kratsch, and D. Paulusma, *Sparse Square Roots*, in Graph-Theoretic Concepts in Computer Science, vol. 8165 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 177–188.

[42] M. Cochefert and D. Kratsch, *Exact Algorithms to Clique-Colour Graphs*, in SOFSEM 2014 : Theory and Practice of Computer Science, vol. 8327 of Lecture Notes in Computer Science, Springer International Publishing, 2014, pp. 187–198.

[43] E. Cockayne, O. Favaron, and C. Mynhardt, *Secure domination, weak roman domination and forbidden subgraphs*, Bull. Inst. Combin. Appl., 39 (2003), pp. 87–100.

[44] E. Cockayne, P. Grobler, W. Gründlingh, J. Munganga, and J. van Vuuren, *Protection of a Graph*, Utilitas Mathematica, 67 (2005), pp. 19–32.

[45] E. Cockayne, P. D. Jr., S. Hedetniemi, and S. Hedetniemi, *Roman domination in graphs*, Discrete Mathematics, 278 (2004), pp. 11–22.

[46] S. A. Cook, *The Complexity of Theorem-proving Procedures*, in Proceedings of the Third Annual ACM Symposium on Theory of Computing, STOC '71, New York, NY, USA, 1971, ACM, pp. 151–158.

[47] D. Corneil, Y. Perl, and L. Stewart, *A Linear Recognition Algorithm for Cographs*, SIAM Journal on Computing, 14 (1985), pp. 926–934.

[48] B. Courcelle, *The monadic second-order logic of graphs. I. Recognizable sets of finite graphs*, Information and Computation, 85 (1990), pp. 12–75.

[49] ——, *The monadic second-order logic of graphs III : Tree-decompositions, minor and complexity issues*, ITA, 26 (1992), pp. 257–286.

[50] B. Courcelle and J. Engelfriet, *Graph Structure and Monadic Second-Order Logic : A Language-Theoretic Approach*, vol. 138, Cambridge University Press, 2012.

[51] B. Courcelle, J. A. Makowsky, and U. Rotics, *Linear Time Solvable Optimization Problems on Graphs of Bounded Clique-Width*, Theory of Computing Systems, 33 (2000), pp. 125–150.

[52] J.-F. Couturier, P. Heggernes, P. van't Hof, and D. Kratsch, *Minimal Dominating Sets in Graph Classes : Combinatorial Bounds and Enumeration*, Theoretical Computer Science, 487 (2013), pp. 82–94.

[53] M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlstrom, *On Problems as Hard as CNF-SAT*, in Computational Complexity (CCC), 2012 IEEE 27th Annual Conference on, June 2012, pp. 74–84.

[54] M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. Wojtaszczyk, *Scheduling Partially Ordered Jobs Faster than* 2^n, Algorithmica, 68 (2014), pp. 692–714.

[55] D. Défossez, *Clique-coloring some classes of odd-hole-free graphs*, Journal of Graph Theory, 53 (2006), pp. 233–249.

[56] F. Dehne, M. Fellows, M. Langston, F. Rosamond, and K. Stevens, *An $O(2^{O(k)}n^3)$ FPT Algorithm for the Undirected Feedback Vertex Set Problem*, Theory of Computing Systems, 41 (2007), pp. 479–492.

[57] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos, *Subexponential Parameterized Algorithms on Bounded-genus Graphs and H-minor-free Graphs*, Journal of the ACM, 52 (2005), pp. 866–893.

[58] E. D. Demaine and M. Hajiaghayi, *The Bidimensionality Theory and Its Algorithmic Applications*, The Computer Journal, 51 (2008), pp. 292–302.

[59] R. Diestel, *Graph Theory*, vol. 173 of Graduate Texts in Mathematics, Springer, Heidelberg, fourth ed., 2010.

[60] H. Djidjev, *On the Problem of Partitioning Planar Graphs*, SIAM Journal on Algebraic Discrete Methods, 3 (1982), pp. 229–240.

[61] H. Djidjev and S. Venkatesan, *Planarization of graphs embedded on surfaces*, in Graph-Theoretic Concepts in Computer Science, M. Nagl, ed., vol. 1017 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 1995, pp. 62–72.

[62] R. Dondi, G. Fertin, and S. Vialette, *Complexity issues in vertex-colored graph pattern matching*, Journal of Discrete Algorithms, 9 (2011), pp. 82–99.

[63] F. Dorn, F. Fomin, and D. Thilikos, *Fast Subexponential Algorithm for Non-local Problems on Graphs of Bounded Genus*, in Algorithm Theory - SWAT 2006, vol. 4059 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 172–183.

[64] F. Dorn, E. Penninkx, H. Bodlaender, and F. Fomin, *Efficient Exact Algorithms on Planar Graphs : Exploiting Sphere Cut Branch Decompositions*, Algorithmica, 58 (2010), pp. 790–810.

[65] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, vol. 3 of Monographs in Computer Science, Springer-Verlag, New York, 1999.

[66] D. Duffus, B. Sands, N. Sauer, and R. Woodrow, *Two-Colouring All Two-Element Maximal Antichains*, Journal of Combinatorial Theory, Series A, 57 (1991), pp. 109–116.

[67] K. Edwards and C. McDiarmid, *New upper bounds on harmonious colorings*, Journal of Graph Theory, 18 (1994), pp. 257–267.

[68] K. Edwards and C. McDiarmid, *The complexity of harmonious colouring for trees*, Discrete Applied Mathematics, 57 (1995), pp. 133–144.

[69] T. Eiter, G. Gottlob, and K. Makino, *New Results on Monotone Dualization and Generating Hypergraph Transversals*, SIAM Journal on Computing, 32 (2003), pp. 514–537.

[70] H. Elghazel, V. Deslandres, M.-S. Hacid, A. Dussauchoy, and H. Kheddouci, *A New Clustering Approach for Symbolic Data and Its Validation : Application to the Healthcare Data*, in Foundations of Intelligent Systems, vol. 4203 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2006, pp. 473–482.

[71] D. Eppstein, *Quasiconvex analysis of backtracking algorithms*, in Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '04, Philadelphia, PA, USA, 2004, Society for Industrial and Applied Mathematics, pp. 788–797.

[72] P. Erdős, W. Hare, S. T. Hedetniemi, and R. Laskar, *On the Equality of the Grundy and Ochromatic Numbers of a Graph*, Journal of Graph Theory, 11 (1987), pp. 157–159.

[73] P. Erdős, S. T. Hedetniemi, R. C. Laskar, and G. C. Prins, *On the equality of the partial Grundy and upper ochromatic numbers of graphs*, Discrete Mathematics, 272 (2003), pp. 53–64.

[74] R. Fagin, *Generalized first-order spectra and polynomial-time recognizable sets*, in Complexity of Computation, SIAM-AMS Proceedings, vol. Vol. 7, 1974.

[75] M. Farber, G. Hahn, P. Hell, and D. Miller, *Concerning the Achromatic Number of Graphs*, Journal of Combinatorial Theory, Series B, 40 (1986), pp. 21–39.

[76] B. Farzad and M. Karimi, *Square-Root Finding Problem In Graphs, A Complete Dichotomy Theorem*, CoRR, abs/1210.7684 (2012).

[77] B. Farzad, L. C. Lau, V. B. Le, and N. N. Tuy, *Complexity of Finding Graph Roots with Girth Conditions*, Algorithmica, 62 (2012), pp. 38–53.

[78] O. Favaron, K. Karami, R. Khoeilar, and S. Sheikholeslami, *On the Roman domination number of a graph*, Discrete Mathematics, 309 (2009), pp. 3447–3451.

[79] M. R. Fellows, G. Fertin, D. Hermelin, and S. Vialette, *Sharp Tractability Borderlines for Finding Connected Motifs in Vertex-Colored Graphs*, in Automata, Languages and Programming, vol. 4596 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, pp. 340–351.

[80] M. R. Fellows, G. Fertin, D. Hermelin, and S. Vialette, *Upper and lower bounds for finding connected motifs in vertex-colored graphs*, Journal of Computer and System Sciences, 77 (2011), pp. 799–811.

[81] J. Flum and M. Grohe, *Parameterized Complexity Theory*, Texts in Theoretical Computer Science. An EATCS Series, Springer Berlin Heidelberg, 2006.

[82] F. Fomin, S. Gaspers, D. Kratsch, M. Liedloff, and S. Saurabh, *Iterative compression and exact algorithms*, in Mathematical Foundations of Computer Science 2008, vol. 5162 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 335–346.

[83] F. Fomin, S. Gaspers, and S. Saurabh, *Improved Exact Algorithms for Counting 3- and 4-Colorings*, in Computing and Combinatorics, vol. 4598 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2007, pp. 65–74.

[84] F. Fomin, F. Grandoni, and D. Kratsch, *Measure and Conquer : Domination - A Case Study*, in Automata, Languages and Programming, vol. 3580 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 191–203.

[85] F. Fomin, F. Grandoni, and D. Kratsch, *A Measure & Conquer Approach for the Analysis of Exact Algorithms*, Journal of the ACM, 56 (2009).

[86] F. Fomin, D. Kratsch, and G. Woeginger, *Exact (Exponential) Algorithms for the Dominating Set Problem*, in Graph-Theoretic Concepts in Computer Science, vol. 3353 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 245–256.

[87] F. Fomin and D. Thilikos, *A Simple and Fast Approach for Solving Problems on Planar Graphs*, in STACS 2004, vol. 2996 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2004, pp. 56–67.

[88] F. Fomin and D. Thilikos, *Fast Parameterized Algorithms for Graphs on Surfaces : Linear Kernel and Exponential Speed-Up*, in Automata, Languages and Programming, vol. 3142 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2004, pp. 581–592.

[89] F. V. Fomin, F. Grandoni, A. V. Pyatkin, and A. A. Stepanov, *Combinatorial Bounds via Measure and Conquer : Bounding Minimal Dominating Sets and Applications*, ACM Transactions on Algorithms, 5 (2008), pp. 9 :1–9 :17.

[90] F. V. Fomin and D. Kratsch, *Exact Exponential Algorithms*, Springer, 2011.

[91] F. V. Fomin and D. M. Thilikos, *New Upper Bounds on the Decomposability of Planar Graphs*, Journal of Graph Theory, 51 (2006), pp. 53–81.

[92] D. Fulkerson and O. Gross, *Incidence matrices and interval graphs*, Pacific journal of mathematics, 15 (1965), pp. 835–855.

[93] D. Gaceb, V. Eglin, F. Lebourgeois, H. Emptoz, and H. Emptoz, *Graph b-Coloring for Automatic Recognition of Documents*, ICDAR '09. 10th International Conference on Document Analysis and Recognition, (2009), pp. 261–265.

[94] M. R. Garey and D. S. Johnson, *Computers and Intractability : A Guide to the Theory of NP-Completeness*, W. H. Freeman & Co., New York, NY, USA, 1979.

[95] S. Gaspers, *Algorithmes exponentiels*, master's thesis, Université de Metz, June 2005.

[96] S. Gaspers, *Exponential Time Algorithms : Structures, Measures, and Bounds*, VDM Verlag Dr. Mueller e.K., 2010.

[97] S. Gaspers, D. Kratsch, and M. Liedloff, *On Independent Sets and Bicliques in Graphs*, in Graph-Theoretic Concepts in Computer Science, vol. 5344 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2008, pp. 171–182.

[98] J. R. Gilbert, J. P. Hutchinson, and R. E. Tarjan, *A separator theorem for graphs of bounded genus*, Journal of Algorithms, 5 (1984), pp. 391–407.

[99] W. Goddard, S. Hedetniemi, and S. Hedetniemi, *Eternal Security in Graphs*, Journal of Combinatorial Mathematics and Combinatorial Computing, 52 (2005), pp. 160–180.

[100] J. Goldwasser and W. Klostermeyer, *Tight bounds for eternal dominating sets in graphs*, Discrete Mathematics, 308 (2008), pp. 2589–2593.

[101] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, vol. 57, Elsevier, 2004.

[102] P. Grobler and C. Mynhardt, *Secure domination critical graphs*, Discrete Mathematics, 309 (2009), pp. 5820–5827.

[103] S. Guillemot and F. Sikora, *Finding and Counting Vertex-Colored Subtrees*, Algorithmica, 65 (2013), pp. 828–844.

[104] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, and S. Wernicke, *Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization*, Journal of Computer and System Sciences, 72 (2006), pp. 1386–1396.

[105] R. P. Gupta, *Bounds on the chromatic and achromatic numbers of complementary graphs*, in Recent Progress in Combinatorics, Academic Press, 1969, pp. 229–235.

[106] M. Hagen, *Lower bounds for three algorithms for transversal hypergraph generation*, Discrete Applied Mathematics, 157 (2009), pp. 1460–1469.

[107] F. Harary, *Graph theory. 1969.*

[108] F. Harary and S. Hedetniemi, *The Achromatic Number of a Graph*, Journal of Combinatorial Theory, 8 (1970), pp. 154–161.

[109] F. Havet, C. L. Sales, and L. Sampaio, *b-coloring of tight graphs*, Discrete Applied Mathematics, (2011).

[110] F. Havet and L. Sampaio, *On the Grundy and b-Chromatic Numbers of a Graph*, Algorithmica, 65 (2013), pp. 885–899.

[111] T. Haynes, S. Hedetniemi, and P. Slater, *Domination in graphs : advanced topics*, vol. 209 of Pure and Applied Mathematics, Marcel Dekker Inc., 1998.

[112] M. Henning and S. Hedetniemi, *Defending the Roman Empire–A new strategy*, Discrete Mathematics, 266 (2003), pp. 239–251.

[113] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier, *Fixed-Parameter Algorithms for Cluster Vertex Deletion*, Theory of Computing Systems, 47 (2010), pp. 196–217.

[114] J. Hopcroft and M. Krishnamoorthy, *On the Harmonious Coloring of Graphs*, SIAM Journal on Algebraic Discrete Methods, 4 (1983), pp. 306–311.

[115] J. Hopcroft and R. Tarjan, *Efficient Planarity Testing*, Journal of the ACM, 21 (1974), pp. 549–568.

[116] W.-L. Hsu, *Recognizing Planar Perfect Graphs*, Journal of the ACM, 34 (1987), pp. 255–288.

[117] R. Impagliazzo and R. Paturi, *On the Complexity of k-SAT*, Journal of Computer and System Sciences, 62 (2001), pp. 367–375.

[118] R. W. Irving and D. F. Manlove, *The b-chromatic number of a graph*, Discrete Applied Mathematics, 91 (1999), pp. 127–141.

[119] Y. Iwata, *A Faster Algorithm for Dominating Set Analyzed by the Potential Method*, in IPEC'11, Proceedings of the 11th International Symposium on Parameterized and Exact Computation, vol. 7112 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 41–54.

[120] M. Jakovac and S. Klavžar, *The b-Chromatic Number of Cubic Graphs*, Graphs and Combinatorics, 26 (2010), pp. 107–118.

[121] R. Karp, *Reducibility among Combinatorial Problems*, in Complexity of Computer Computations, The IBM Research Symposia Series, Springer US, 1972, pp. 85–103.

[122] D. J. Kavvadias and E. C. Stavropoulos, *An Efficient Algorithm for the Transversal Hypergraph Generation*, Journal of Graph Algorithms and Applications, 9 (2005), pp. 239–264.

[123] K. Kawarabayashi, B. Mohar, and B. Reed, *A Simpler Linear Time Algorithm for Embedding Graphs into an Arbitrary Surface and the Genus of Graphs of Bounded Tree-Width*, in Foundations of Computer Science, 2008. FOCS '08. IEEE 49th Annual IEEE Symposium on, Oct 2008, pp. 771–780.

[124] S. Klein and A. Morgana, *On clique-colouring of graphs with few P4's*, Journal of the Brazilian Computer Society, 18 (2012), pp. 113–119.

[125] T. Kloks, *Treewidth : computations and approximations*, vol. 842, Springer, 1994.

[126] D. König, *Theory of Finite and Infinite Graphs*, in Theory of Finite and Infinite Graphs, Birkhäuser Boston, 1990, pp. 45–421.

[127] E. Korach and N. Solel, *Tree-width, path-width, and cutwidth*, Discrete Applied Mathematics, 43 (1993), pp. 97–101.

[128] G. Kortsarz, J. Radhakrishnan, and S. Sivasubramanian, *Complete partitions of graphs*, in Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '05, Society for Industrial and Applied Mathematics, 2005, pp. 860–869.

[129] M. Kouider and M. Zaker, *Bounds for the b-chromatic number of some families of graphs*, Discrete Mathematics, 306 (2006), pp. 617–623.

[130] J. Kratochvíl and Z. Tuza, *On the complexity of bicoloring clique hypergraphs of graphs*, Journal of Algorithms, 45 (2002), pp. 40–54.

[131] J. Kratochvíl, Z. Tuza, and M. Voigt, *On the b-Chromatic Number of Graphs*, in Graph-Theoretic Concepts in Computer Science, vol. 2573 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2002, pp. 310–320.

[132] C. Kuratowski, *Sur le problème des courbes gauches en topologie*, Fundamenta Mathematicae, 15 (1930), pp. 271–283.

[133] V. Lacroix, C. G. Fernandes, and M.-F. Sagot, *Motif Search in Graphs : Application to Metabolic Networks*, IEEE/ACM Transactions on Computational Biology and Bioinformatics, 3 (2006), pp. 360–368.

[134] L. C. Lau, *Bipartite roots of graphs*, ACM Transactions on Algorithms, 2 (2006), pp. 178–208.

[135] L. C. Lau and D. G. Corneil, *Recognizing Powers of Proper Interval, Split, and Chordal Graph*, SIAM Journal on Discrete Mathematics, 18 (2004), pp. 83–102.

[136] E. L. Lawler, *A note on the complexity of the chromatic number problem*, Information Processing Letters, (1976), pp. 66–67.

[137] V. B. Le and N. N. Tuy, *The square of a block graph*, Discrete Mathematics, 310 (2010), pp. 734–741.

[138] ——, *A good characterization of squares of strongly chordal split graphs*, Information Processing Letters, 111 (2011), pp. 120–123.

[139] J. Lee and Y. hee Shin, *The achromatic number of the union of cycles*, Discrete Applied Mathematics, 143 (2004), pp. 330–335.

[140] S.-M. Lee and J. Mitchem, *An upper bound for the harmonious chromatic number of a graph*, Journal of Graph Theory, 11 (1987), pp. 565–567.

[141] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen, *Introduction to algorithms*, The MIT press, 2001.

[142] M. Liedloff, *Algorithmes exacts et exponentiels pour les problèmes NP-difficiles : domination, variantes et généralisations*, phd thesis, Laboratoire d'Informatique Théorique et Appliquée, Université Paul Verlaine, Metz, 2007.

[143] ——, *Finding a dominating set on bipartite graphs*, Information Processing Letters, 107 (2008), pp. 154–157.

[144] M. Liedloff, T. Kloks, J. Liu, and S.-L. Peng, *Efficient algorithms for Roman domination on some classes of graphs*, Discrete Applied Mathematics, 156 (2008), pp. 3400–3415.

[145] Y.-L. Lin and S. Skiena, *Algorithms for Square Roots of Graphs*, SIAM Journal on Discrete Mathematics, 8 (1995), pp. 99–118.

[146] R. Lipton and R. Tarjan, *A Separator Theorem for Planar Graphs*, SIAM Journal on Applied Mathematics, 36 (1979), pp. 177–189.

[147] C.-H. Liu and G. Chang, *Roman Domination on 2-Connected Graphs*, SIAM Journal on Discrete Mathematics, 26 (2012), pp. 193–205.

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

[149] G. MacGillivray and A. Rodriguez, *The achromatic number of the union of paths*, Discrete Mathematics, 231 (2001), pp. 331–335.

[150] D. Marx, *Complexity of clique coloring and related problems*, Theoretical Computer Science, 412 (2011), pp. 3487–3500.

[151] A. Máté, *A lower estimate for the achromatic number of irreducible graphs*, Discrete Mathematics, 33 (1981), pp. 171–183.

[152] A. Maury, *Énumération des dominants minimaux d'un graphe*, PhD thesis, Université Blaise Pascal, 2013.

[153] F. McMorris, T. Warnow, and T. Wimer, *Triangulating vertex-colored graphs*, SIAM Journal on Discrete Mathematics, 7 (1994), pp. 296–306.

[154] M. Milanic and O. Schaudt, *Computing square roots of trivially perfect and threshold graphs*, Discrete Applied Mathematics, in press.

[155] G. Miller, *Isomorphism testing for graphs of bounded genus*, in Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing, STOC '80, New York, NY, USA, 1980, ACM, pp. 225–235.

[156] Z. Miller and D. Pritikin, *The harmonious coloring number of a graph*, Discrete Mathematics, 93 (1991), pp. 211–228.

[157] J. Mitchem, *On the Harmonious Chromatic Number of a Graph*, in Graph Colouring and Variations, vol. 39 of Annals of Discrete Mathematics, Elsevier, 1989, pp. 151–157.

[158] B. Mohar and R. Škrekovski, *The Grötzsch Theorem for the hypergraph of maximal cliques*, Electronic Journal of Combinatorics, 6 (1999), p. 2.

[159] J. Moon and L. Moser, *On cliques in graphs*, Israel Journal of Mathematics, 3 (1965), pp. 23–28.

[160] R. Motwani and M. Sudan, *Computing roots of graphs is hard*, Discrete Applied Mathematics, 54 (1994), pp. 81–88.

[161] A. Mukhopadhyay, *The Square Root of a Graph*, Journal of Combinatorial Theory, 2 (1967), pp. 290–295.

[162] S.-i. Nakano and T. Uno, *Constant Time Generation of Trees with Specified Diameter*, in Graph-Theoretic Concepts in Computer Science, vol. 3353 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 33–45.

[163] J. Nederlof, *Fast Polynomial-Space Algorithms Using Inclusion-Exclusion*, Algorithmica, 65 (2013), pp. 868–884.

[164] R. Niedermeier, *Invitation to Fixed-Parameter Algorithms*, vol. 31 of Oxford Lecture Series in Mathematics and its Applications, Oxford University Press, Oxford, 2006.

[165] R. Niedermeier and P. Rossmanith, *A general method to speed up fixed-parameter-tractable algorithms*, Information Processing Letters, 73 (2000), pp. 125–129.

[166] M. Rao, *MSOL partitioning problems on graphs of bounded treewidth and clique-width*, Theoretical Computer Science, 377 (2007), pp. 260–267.

[167] C. ReVelle and K. Rosing, *Defendens Imperium Romanum : A Classical Problem in Military Strategy*, American Mathematical Monthly, 107 (2000), pp. 585–594.

[168] R. Rizzi and F. Sikora, *Some Results on more Flexible Versions of Graph Motif*, in Computer Science - Theory and Applications, vol. 7353 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 278–289.

[169] N. Robertson, D. Sanders, P. Seymour, and R. Thomas, *The Four-Colour Theorem*, Journal of Combinatorial Theory, Series B, 70 (1997), pp. 2–44.

[170] N. Robertson and P. Seymour, *Graph minors. I. Excluding a forest*, Journal of Combinatorial Theory, Series B, 35 (1983), pp. 39–61.

[171] N. Robertson and P. Seymour, *Graph minors. X. Obstructions to tree-decomposition*, Journal of Combinatorial Theory, Series B, 52 (1991), pp. 153–190.

[172] J. Robson, *Algorithms for maximum independent sets*, Journal of Algorithms, 7 (1986), pp. 425–440.

[173] J. M. Robson, *Finding a maximum independent set in time $O(2^{n/4})$*. `https://www.labri.fr/perso/robson/mis/techrep.html`.

[174] D. Rose, R. Tarjan, and G. Lueker, *Algorithmic Aspects of Vertex Elimination on Graphs*, SIAM Journal on Computing, 5 (1976), pp. 266–283.

[175] I. C. Ross and F. Harary, *The Square of a Tree*, Bell System Technical Journal, 39 (1960), pp. 641–647.

[176] G. F. Royle, *Counting Set Covers and Split Graphs*, Journal of Integer Sequences, 3 (2000), p. 3.

[177] P. Seymour and R. Thomas, *Call routing and the ratcatcher*, Combinatorica, 14 (1994), pp. 217–241.

[178] Z. Shi, W. Goddard, S. T. Hedetniemi, K. Kennedy, R. Laskar, and A. McRae, *An algorithm for partial Grundy number on trees*, Discrete Mathematics, 304 (2005), pp. 108–116.

[179] I. Stewart, *Defend the Roman Empire !*, Scientific American, 281 (1999), pp. 136–139.

[180] J. Telle and A. Proskurowski, *Algorithms for Vertex Partitioning Problems on Partial k-Trees*, SIAM Journal on Discrete Mathematics, 10 (1997), pp. 529–550.

[181] C. Thomassen, *The graph genus problem is NP-complete*, Journal of Algorithms, 10 (1989), pp. 568–576.

[182] S. Tsukiyama, M. Ide, H. Ariyoshi, and I. Shirakawa, *A New Algorithm for Generating All the Maximal Independent Sets*, SIAM Journal on Computing, 6 (1977), pp. 505–517.

[183] J. van Rooij, *Exact exponential-time algorithms for domination problems in graphs*, phd thesis, Utrecht University, Netherlands, 2011.

[184] J. van Rooij and H. Bodlaender, *Exact algorithms for dominating set*, Discrete Applied Mathematics, 159 (2011), pp. 2147–2164.

[185] J. van Rooij, J. Nederlof, and T. van Dijk, *Inclusion/Exclusion Meets Measure and Conquer*, in Algorithms - ESA 2009, vol. 5757 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 554–565.

[186] M. Wahlström, *Exact algorithms for finding minimum transversals in rank-3 hypergraphs*, Journal of Algorithms, 51 (2004), pp. 107–121.

[187] ——, *Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems*, PhD thesis, Linköping University, Sweden, 2007.

[188] G. Woeginger, *Exact Algorithms for NP-Hard Problems : A Survey*, in Combinatorial Optimization - Eureka, You Shrink !, M. Jünger, G. Reinelt, and G. Rinaldi, eds., vol. 2570 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2003, pp. 185–207.

[189] M. Xiao and H. Nagamochi, *Exact Algorithms for Maximum Independent Set*, CoRR, abs/1312.6260 (2013).

[190] ——, *Exact Algorithms for Maximum Independent Set*, in Algorithms and Computation, vol. 8283 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 328–338.

[191] H.-M. Xing, X. Chen, and X.-G. Chen, *A note on Roman domination in graphs*, Discrete Mathematics, 306 (2006), pp. 3338–3340.

[192] M. Yannakakis and F. Gavril, *Edge Dominating Sets in Graphs*, SIAM Journal on Applied Mathematics, 38 (1980), pp. 364–372.

[193] V. Yegnanarayanan, *The Pseudoachromatic Number of a Graph*, Southeast Asian Bulletin of Mathematics, 24 (2000), pp. 129–136.

[194] V. Yegnanarayanan, *Graph colourings and partitions*, Theoretical Computer Science, 263 (2001), pp. 59–74.

[195] M. Zaker, *Results on the Grundy chromatic number of graphs*, Discrete Mathematics, 306 (2006), pp. 3166–3173.

[196] V. Zemlyachenko, N. Korneenko, and R. Tyshkevich, *Graph isomorphism problem*, Journal of Soviet Mathematics, 29 (1985), pp. 1426–1481.