

ÉCOLE DOCTORALE
MATHEMATIQUES, INFORMATIQUE, PHYSIQUE THEORIQUE
ET INGENIERIE DES SYSTEMES

Laboratoire d'Informatique Fondamentale d'Orléans

THÈSE

présentée par :

Romain LETOURNEUR

soutenue le : **9 juillet 2015**

pour obtenir le grade de : **Docteur de l'université d'Orléans**

Discipline/ Spécialité : **Informatique**

Algorithmes exacts et exponentiels
pour des problèmes de graphes

THÈSE dirigée par :

Ioan TODINCA

Professeur des Universités, Université d'Orléans

Mathieu LIEDLOFF

Maître de Conférences, Université d'Orléans

RAPPORTEURS :

Frédéric HAVET

Directeur de Recherche CNRS, Nice-Sophia-Antipolis

Cristina BAZGAN

Professeur des Universités, Université Paris Dauphine

JURY :

Cristina BAZGAN

Professeur des Universités, Université Paris Dauphine

Frédéric HAVET

Directeur de Recherche CNRS, Nice-Sophia-Antipolis

Mathieu LIEDLOFF

Maître de Conférences, Université d'Orléans

Alexandre PINLOU

Maître de Conférences HDR, Université Paul Valéry Montpellier 3

Vincent T'KINDT

Professeur des Universités, Université François Rabelais de Tours

Ioan TODINCA

Professeur des Universités, Université d'Orléans

Remerciements

Je tiens tout d'abord à remercier chaleureusement mes directeurs de thèse, Mathieu Liedloff et Ioan Todinca, présents tout au long de ma thèse, supportant mes blagues quelquefois limites, mes quelques moments de flemme et bien sûr leur aide et leur soutien au quotidien qui m'ont permis de réaliser cette thèse dans les meilleures conditions. Ce fut un réel plaisir de travailler avec eux, et de découvrir le plaisir et la fierté d'obtenir son premier résultat, publier son premier article, présenter ses résultats devant des experts mondiaux et finalement découvrir le travail de chercheur.

Je tiens également à remercier mes rapporteurs de thèse, Cristina Bazgan ainsi que Frédéric Havet qui ont donné de leur temps pour lire avec attention mon manuscrit et m'ont permis, par leurs précieux commentaires, de le perfectionner. Je remercie par la même occasion Vincent T'Kindt ainsi qu'Alexandre Pinlou qui ont accepté d'être examinateurs de ma thèse.

Je remercie bien évidemment toutes les personnes avec lesquelles j'ai eu l'occasion de travailler durant ces trois années. Je tiens tout particulièrement à remercier mes coauteurs auprès desquels j'ai énormément appris, Mathieu Chapelle, Manfred Cochefert, Jean-François Couturier, Dieter Kratsch et Mathieu Liedloff.

Un grand merci va également à tous les membres du LIFO, collègues que j'ai tout d'abord côtoyés en tant qu'étudiant, puis en tant que collègue. Je leur dois à tous énormément de choses, l'essentiel de mes connaissances acquises durant mes études de Licence et de Master ainsi que ma passion pour l'informatique, mais aussi tous ces bons moments passés durant ma thèse. Je remercie tout particulièrement les directeurs du LIFO, Jérôme Durand-Lose et Sébastien Limet, ainsi que le directeur de l'équipe de recherche GAMoC, Nicolas Ollinger, et bien évidemment tous les membres de l'équipe.

Bon, les chefs, c'est fait, les collègues aussi, ensuite ...

J'adresse un merci tout particulier à mes compagnons de galère, ceux qui sont devenus Docteurs au combat, les petits nouveaux pour lesquels l'aventure ne fait que commencer. Je pense en particulier (liste non exhaustive !) à Mathieu Chapelle et Maxime Senot, ceux qui m'ont donné envie de faire cette thèse, Pedro Montealegre, mon cobureau, le cador du "Confite de Pato" (Bonne année à toi copain !), Bastien Le Gloannec (souris un peu !) ou encore à Vivien Pelletier (le gars de l'équipe de nuit). Merci à tous, c'était bien sympa !

Je remercie également ma famille, en particulier mes parents, mon frère et Chacalou, qui ont également toujours été là pour moi, et qui m'ont soutenu durant toutes ces années.

Un merci tout particulier va à mes amis, je pense principalement à Bicu (Bro' !), Dindinou (Bro' #2 !), Couenne (d'abord en L1, puis en L1 puis en L1 ...), Jaune (check !), ainsi qu'à tous mes collègues avec lesquels j'ai partagé ces moments si précieux.

Je remercie également Linux, alias Nunux, cap'tain nunux, el diablo, nunuxinator ... (un nouveau surnom par jour), mon félin compagnon d'écriture criblé de dettes de jeu !

Mon plus grand merci va incontestablement à ma chérie, ma « femelle » comme ils disent, Marie-Pierre, ma petite Merguez, qui est présente au quotidien à mes côtés, supportant mes moments quasi permanents de boude et avec qui je partage ces moments si épiques. Merci à toi, on est vraiment un super binôme !

A tous merci,

Romain

« La photosynthèse, c'est qu'une question de volonté. »

Bicu, Letrome, Merguez [2015]

Sommaire

Introduction	1
1 Préliminaires et notations	7
1.1 Notions de graphes	8
1.1.1 Notations	8
1.1.2 Les classes de graphes	11
1.2 Conception d'algorithmes et analyse de leur efficacité	15
1.2.1 Complexité des algorithmes : notation asymptotique	15
1.2.2 Problèmes de décision : les classes P et NP	15
1.2.3 Problèmes d'optimisation et d'énumération	17
1.2.4 L'Hypothèse de Temps Exponentiel	18
1.3 Résoudre un problème difficile	19
1.3.1 Différentes techniques algorithmiques	19
1.3.2 Analyse du temps d'exécution	20
1.4 Quelques problèmes classiques	26
2 Recherche d'ensemble connexe tropical minimum	29
2.1 Introduction	30
2.1.1 Détection de motifs dans les graphes colorés	30
2.1.2 État de l'art	31
2.1.3 Résultats du chapitre	32
2.1.4 Préliminaires	33
2.2 Un algorithme exact pour les graphes quelconques	34
2.2.1 Un algorithme par recherche exhaustive	35
2.2.2 Un algorithme via Arbre de Steiner	35
2.2.3 Un algorithme via Ensemble Dominant Connexe Bleu et Rouge	39
2.2.4 Combinaison des trois approches	42
2.3 Non-existence d'un algorithme sous-exponentiel pour les arbres	44
2.4 Un algorithme exact pour les arbres	45
2.4.1 Instances et sous-problèmes	46
2.4.2 Description de l'algorithme	47
2.5 Conclusion	58

3	Énumération des ensembles dominants minimaux dans quelques classes de graphes	61
3.1	Introduction	62
3.1.1	Énumération d'ensembles dominants minimaux	62
3.1.2	État de l'art	63
3.1.3	Résultats du chapitre	66
3.1.4	Preliminaires	66
3.2	Énumération dans les graphes splits	67
3.2.1	L'algorithme	68
3.2.2	Correction de l'algorithme et borne combinatoire	71
3.3	Énumération dans les graphes cobipartis	75
3.3.1	L'algorithme	75
3.3.2	Correction de l'algorithme et borne combinatoire	78
3.4	Énumération dans les graphes d'intervalles	81
3.4.1	Graphes partiels et bons triplets	81
3.4.2	L'algorithme	82
3.4.3	Correction de l'algorithme et borne combinatoire	82
3.5	Conclusion	87
4	Algorithme pour la domination romaine faible sur les graphes d'intervalles	91
4.1	Introduction	92
4.1.1	Domination, domination romaine et domination romaine faible	92
4.1.2	État de l'art	94
4.1.3	Résultats de ce chapitre	95
4.1.4	Preliminaires et notations	95
4.2	Domination Romaine Faible dans le cas général	97
4.3	Domination Romaine Faible sur les graphes d'intervalles	99
4.3.1	Description de l'algorithme	99
4.3.2	Analyse du temps d'exécution	106
4.4	Conclusion	111
5	Conclusion	115

Bibliographie

Introduction

Résoudre un problème efficacement est une question classique pour un algorithmicien. L'étude de deux ressources est essentielle afin de mesurer l'efficacité d'un algorithme : son temps d'exécution et l'espace qu'il nécessite. Certains problèmes sont malheureusement plus difficiles que d'autres. Pour certains, les connaissances actuelles n'autorisent pas d'algorithmes efficaces ; pour d'autres, cela est inhérent à leur nature demandant de produire en sortie un grand nombre d'objets.

Cette thèse considère typiquement des problèmes difficiles pour lesquels des algorithmes modérément exponentiels, c'est-à-dire plus rapides que la recherche exhaustive, seront proposés. On s'intéresse également à des restrictions de ces problèmes, où nous serons capables de proposer des algorithmes polynomiaux, voire linéaires en la taille de l'entrée. On regardera finalement des problèmes demandant d'énumérer des objets combinatoires respectant certaines propriétés. Nous montrerons que ces objets sont en nombre exponentiel et donnerons des algorithmes les plus efficaces possibles, bien qu'exponentiels, pour les énumérer.

Parmi ces problèmes difficiles il y a les problèmes NP-complets. Leur étude commence dans les années 70 avec la preuve de NP-complétude pour le problème 3-SAT que l'on doit à Cook et Levin en 1971. Les problèmes NP-complets sont très nombreux, et présents dans des domaines variés comme en mathématiques, en biologie, en télécommunication, etc. Il est primordial de pouvoir les résoudre le plus efficacement possible. Nous ne savons pas à l'heure actuelle si de tels problèmes peuvent être résolus en temps polynomial. C'est la grande question ouverte : est-ce que $P = NP$?

Ces problèmes ne sont pour autant pas impossibles à traiter, ne serait-ce que par une recherche exhaustive de toutes les possibilités. La question de l'existence d'algorithmes plus rapides que la recherche exhaustive a été posée très tôt par Gödel dans une lettre à destination de von Neumann¹ en 1956 :

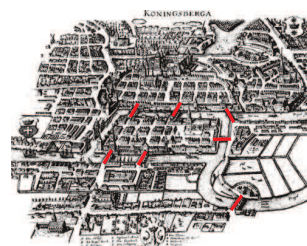
« It would be interesting to know, for instance, the situation concerning the determination of primality of a number and how strongly in general the number of steps in finite combinatorial problems can be reduced with respect to simple exhaustive search. »

1. L'intégralité de cette lettre peut être consulté sur le blog de Lipton à l'adresse rjlipton.wordpress.com/the-gdel-letter.

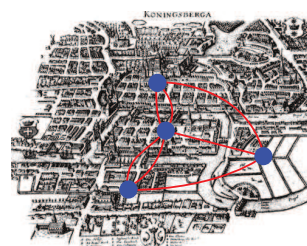
Si la réponse concernant le test de primalité est connue depuis peu, la question reste entière en général : « est-ce que tous les problèmes peuvent être résolus plus rapidement que par une recherche exhaustive ? »

Nous nous intéressons plus particulièrement dans cette thèse à l'étude des problèmes *NP*-complets de graphes.

Les graphes trouvent leur origine dans une problématique mettant en scène la ville originellement prussienne de Königsberg. C'est une ville traversée par un fleuve et séparée en 4 parties : celle de la rive nord, celle de la rive sud ainsi que deux îles centrales ; plusieurs ponts relient ces parties entre elles, comme en atteste la figure ci-contre qui est une gravure de la ville de Königsberg sur laquelle les 7 ponts ont été mis en évidence.



Euler énonça un problème mettant en scène ces ponts : étant donné un point de départ dans cette ville, existe-t-il une promenade passant exactement une fois par chacun de ces ponts et revenant au point de départ ? Il définit la notion de (multi)graphe pour y répondre. Les graphes sont formés par une collection de points, appelés sommets, qui peuvent être reliés entre eux par des liens, appelés arêtes. La figure ci-contre représente le graphe associé à Königsberg, les sommets représentant les différentes parties de la ville, et les arêtes les ponts les reliant.



Il énonça finalement le théorème suivant :

Théorème. [Théorème d'Euler (1736)]

Un graphe connexe est eulérien si et seulement si chacun de ses sommets est incident à un nombre pair d'arêtes.

Par le théorème d'Euler un graphe est eulérien si tous les points sont incidents à un nombre pair de liens. Ce théorème répond donc à la question posée initialement : étant donné qu'au moins un des points est adjacent à un nombre impair de liens (en fait ils le sont tous), le graphe représentant la ville de *Königsberg* n'est pas un graphe eulérien ; une telle promenade n'existe donc pas. Une preuve rigoureuse de ce théorème fut établie 137 ans plus tard par *Carl Hierholzer*. Plus de détails concernant ce problème et sa résolution sont donnés dans « Introduction to Graph Theory » de *West* [Wes00].

Les graphes, malgré leur simplicité, permettent de modéliser de nombreux problèmes en informatique, dont certains sont difficiles. Ces problèmes n'admettent a priori pas d'algorithmes ef-

ficaces (polynomiaux) pour les résoudre. Il existe néanmoins plusieurs approches pour les traiter. Une première approche consiste à ne pas mesurer le temps d'exécution d'un algorithme uniquement par la taille de l'entrée mais aussi par rapport à d'autres paramètres pertinents dans le but de réduire l'explosion combinatoire à ces paramètres. Concernant les problèmes pour lesquels on cherche à concevoir une solution optimale selon un critère donné, il est possible de concevoir des algorithmes rapides mais ne calculant qu'une solution sous-optimale. Une autre approche consiste à restreindre l'entrée à celles respectant certaines bonnes propriétés qui nous permettront d'obtenir des algorithmes plus performants.

Il peut également être intéressant de résoudre ces problèmes difficiles en concevant des algorithmes exacts qui soient modérément exponentiels. Bien que pour de grandes instances de tels algorithmes ne sont pas exploitables, ils peuvent l'être pour des instances de taille plus modeste.

Nous allons concevoir dans cette thèse des algorithmes exacts, exponentiels ou polynomiaux, pour divers problèmes d'optimisation ou d'énumération associés à des problèmes difficiles en général. Pour les algorithmes exponentiels, nous obtiendrons des temps d'exécution sensiblement meilleurs qu'une recherche exhaustive.

Nous nous intéressons à la résolution de problèmes d'énumération et d'optimisation difficiles dans les graphes.² L'étude se concentre en particulier sur trois problèmes. Nous étudierons tout d'abord le problème d'optimisation **ENSEMBLE CONNEXE TROPICAL MINIMUM** pour lequel nous cherchons, étant donné un graphe dont les sommets sont colorés (cette coloration n'étant pas nécessairement propre), un sous-ensemble de sommets connexe et de cardinalité minimum contenant chaque couleur du graphe. Le prochain problème étudié est le problème d'énumération **ENSEMBLES DOMINANTS MINIMAUX**. Nous chercherons alors à énumérer les ensembles dominants minimaux lorsque le graphe en entrée se restreint à certaines classes de graphes. Nous nous sommes finalement intéressés au problème d'optimisation **DOMINATION ROMAINE FAIBLE MINIMUM** pour lequel on cherche à placer des *légions* sur les sommets de telle sorte que le graphe respecte certaines propriétés. Le premier problème ne peut pas à notre connaissance être résolu en temps polynomial (car il est, dans sa version de décision, *NP*-complet). Le second problème demande d'énumérer en général un grand nombre d'objets. Ces trois problèmes font partie de problématiques actuellement très étudiées dans le domaine de l'algorithmique et la combinatoire des graphes.

La première de ces problématiques est l'énumération de structures particulières dans les graphes et l'établissement d'une borne combinatoire sur le nombre maximum de tels objets qu'un graphe peut avoir : citons notamment l'énumération des stables maximaux, des ensembles coupes-cycles minimaux ou encore l'énumération d'ensembles dominants minimaux, ce qui est le cas du

2. Nous donnons plus de détails concernant les problèmes d'énumération et d'optimisation dans la section 1.2 du chapitre 1.

problème **ENSEMBLES DOMINANTS MINIMAUX**.

La seconde concerne les problèmes pour lesquels le graphe en entrée arrive avec une coloration (pas nécessairement propre) sur les sommets. Ces problèmes, souvent inspirés par la biologie, sont intrinsèquement plus complexes que des problèmes impliquant des graphes simples. Avec le problème **ENSEMBLE CONNEXE TROPICAL MINIMUM** nous nous intéressons à la recherche de structures particulières dans des graphes colorés.

La dernière problématique dont nous parlerons ici concerne les problèmes introduisant une certaine notion de dynamique. L'étude de tels problèmes est relativement récente. Parmi ces problèmes citons les problèmes de reconfigurations ou les problèmes de jeux combinatoires. Une certaine idée de dynamique est présente dans le problème **DOMINATION ROMAINE FAIBLE MINIMUM**.

Plan de la thèse et résumé

Le **chapitre 1** est dédié à la présentation des notions essentielles à la bonne compréhension de cette thèse. Nous introduisons tout d'abord des notions propres aux graphes, ainsi que des classes de graphes, puis des notions concernant la complexité des problèmes et des algorithmes. Nous présentons ensuite plusieurs techniques algorithmiques ainsi que des outils afin d'analyser le temps d'exécution d'algorithmes. Finalement nous donnons un ensemble de problèmes classiques et utiles tout au long de cette thèse.

Le **chapitre 2** est dédié à la recherche d'un *ensemble connexe tropical* dans les graphes. Étant donné un graphe $G = (V, E)$ et une coloration de ses sommets c (coloration qui n'est pas nécessairement propre, deux sommets adjacents peuvent avoir la même couleur), un sous-ensemble $V' \subseteq V$ est un ensemble connexe tropical si le graphe induit par V' est connexe et si V' contient chacune des couleurs utilisées par la coloration c . Nous donnons dans ce chapitre un algorithme exact exponentiel calculant, étant donné un graphe à n sommets, un ensemble connexe tropical de cardinalité minimum en temps $\mathcal{O}^*(1.5359^n)$, sensiblement plus efficace que l'algorithme trivial énumérant tous les sous-ensembles de sommets possibles en temps $\mathcal{O}^*(2^n)$. Nous utilisons la notation « $\mathcal{O}^*$ », qui fonctionne comme la notation « $\mathcal{O}$ » mais ignore en plus les facteurs polynomiaux en n , cf section 1.2 du prochain chapitre. En utilisant la réduction établie par *Angles d'Auriac et al.* dans [AdCH⁺14], ainsi que la preuve par *Fomin et al.* [FKW04] qu'**ENSEMBLE DOMINANT** n'admet pas d'algorithme sous-exponentiel dans les graphes de degré maximum 6 sous l'*Exponential Time Hypothesis* proposée par *Impagliazzo et Paturi* dans [IP01], nous montrons qu'**ENSEMBLE CONNEXE TROPICAL MINIMUM** n'admet pas d'algorithme sous-exponentiel même lorsque le graphe en entrée est un arbre de hauteur au plus 3, à moins que l'*ETH* ne soit fausse. Pour clore ce chapitre, nous

donnons un algorithme exact exponentiel de branchement résolvant **ENSEMBLE CONNEXE TROPICAL MINIMUM** lorsque le graphe en entrée est un arbre, cet algorithme ayant un temps d'exécution de $\mathcal{O}(1.2721^n)$.

Les travaux de ce chapitre ont été établis en collaboration avec *Mathieu Chapelle, Manfred Cochefert, Dieter Kratsch et Mathieu Liedloff*. Ils ont fait l'objet d'un article présenté à la conférence *IPEC 2014* [CCK⁺14].

Nous nous intéressons dans le **chapitre 3** à l'énumération des ensembles dominants minimaux dans certaines classes de graphes. Étant donné un graphe $G = (V, E)$, un sous-ensemble $V' \subseteq V$ est un *ensemble dominant* si tout sommet de V est soit dans V' , soit a un voisin dans V' , et il est *minimal* s'il ne contient aucun sous-ensemble strictement inclus qui est lui-même un ensemble dominant. L'étude de ce chapitre porte donc sur la conception d'algorithmes énumérant ces ensembles lorsque l'entrée est restreinte à certaines classes de graphes. Nous donnons tout d'abord un algorithme pour **ENSEMBLES DOMINANTS MINIMAUX** lorsque le graphe en entrée est un graphe split. Cet algorithme énumère ces sous-ensembles en temps $\mathcal{O}(1.4423^n)$ et nous montrerons que cette borne est la meilleure borne possible. Nous donnons ensuite un algorithme énumérant les ensembles dominants minimaux en temps $\mathcal{O}(1.4511^n)$ lorsque le graphe en entrée est un graphe cobiparti. Nous donnons finalement un algorithme énumérant les ensembles dominants minimaux dans les graphes d'intervalles en temps $\mathcal{O}^*(3^{n/3})$ et nous montrons que c'est la meilleure borne possible. Cela améliore des résultats de *Couturier et al.* [CHvHK13] et répond à certaines de leurs conjectures.

Les travaux de ce chapitre ont fait l'objet d'un article en collaboration avec *Jean-François Couturier et Mathieu Liedloff*, présenté à la conférence *ICGT 2014* et publié dans le journal *Theoretical Computer Science* [CLL15].

Dans le **chapitre 4** nous nous intéressons au problème **DOMINATION ROMAINE FAIBLE MINIMUM** qui tient son nom de la problématique initiale : on cherche à sécuriser l'empire romain du IV^{ème} siècle en plaçant des légions pour dominer toutes les régions, le tout en utilisant le minimum de légions. La problématique est la suivante : étant donné un graphe $G = (V, E)$, une fonction $f : V \rightarrow \{0, 1, 2\}$ est une fonction de domination romaine faible si, pour tout sommet $v \in V$ tel que $f(v) = 0$, v a un voisin u tel que $f(u) \geq 1$ et tel que l'ensemble $D = \{x \mid f_{u \rightarrow v}(x) \geq 1\}$ est un ensemble dominant de G , la fonction $f_{u \rightarrow v}$ étant définie par :

$$f_{u \rightarrow v}(x) = \begin{cases} 1 & \text{si } x = v \\ f(u) - 1 & \text{si } x = u \\ f(x) & \text{si } x \notin \{u, v\}. \end{cases}$$

Le coût d'une telle fonction est égal à $\sum_{x \in V} f(x)$. Nous cherchons dans ce problème à calculer une fonction de domination romaine faible qui soit de coût minimum. Nous rappelons tout d'abord quelques résultats concernant ce problème dans le cas général, établis par *Chapelle et*

al. dans [CCC⁺13], puis nous montrons que ce problème peut être résolu en temps polynomial lorsque le graphe en entrée est un graphe d'intervalles. En introduisant des structures de données adéquates que nous précalculerons, nous montrons finalement que l'algorithme peut être modifié pour calculer une fonction de domination romaine faible en temps linéaire.

Les travaux de ce chapitre ont fait l'objet d'un article en collaboration avec *Mathieu Liedloff* actuellement soumis à *EUROCOMB 2015*. Il fera l'objet d'une soumission en journal, en combinaison avec l'article [CCC⁺13] de *Chapelle et al.*

Finalement, dans le **chapitre 5**, nous rappelons les différents résultats établis tout au long de cette thèse, et nous présentons les perspectives envisagées quant à la continuité de ces résultats.

1 Préliminaires et notations

Sommaire

1.1	Notions de graphes	8
1.1.1	Notations	8
1.1.2	Les classes de graphes	11
1.2	Conception d'algorithmes et analyse de leur efficacité	15
1.2.1	Complexité des algorithmes : notation asymptotique	15
1.2.2	Problèmes de décision : les classes P et NP	15
1.2.3	Problèmes d'optimisation et d'énumération	17
1.2.4	L'Hypothèse de Temps Exponentiel	18
1.3	Résoudre un problème difficile	19
1.3.1	Différentes techniques algorithmiques	19
1.3.2	Analyse du temps d'exécution	20
1.4	Quelques problèmes classiques	26

Ce chapitre est consacré à la définition de notions utiles tout au long de cette thèse. Nous donnons un ensemble de définitions et de notations générales concernant les graphes, ainsi qu'un ensemble de classes de graphes qui seront nécessaires par la suite. Nous donnons ensuite quelques notions clés concernant l'analyse des algorithmes, en rappelant quelques classes de complexité et des notations asymptotiques. Nous présentons des techniques algorithmiques étudiées et utilisées dans cette thèse afin de construire des algorithmes plus efficaces que la recherche exhaustive pour résoudre des problèmes de graphes, ainsi que différents outils classiques afin d'analyser le temps d'exécution de ces algorithmes. Nous rappelons finalement un ensemble de problèmes classiques.

1.1 Notions de graphes

Nous donnons dans cette section des définitions et des notations concernant les **graphes**. Les graphes sont des objets mathématiques, constitués d'un ensemble de *points* appelés **sommets**, ainsi que d'un ensemble de *liens* appelés **arêtes** ou **arcs** lorsqu'on leur donne une orientation. Sauf indication contraire, les graphes étudiés dans cette thèse sont des **graphes simples** (au plus une seule arête relie deux sommets et chaque arête relie deux sommets distincts, en opposition aux *multigraphes*) **et non-orientés**.

On ne parlera pas en détail ici de la représentation en mémoire d'une structure de graphe. Habituellement on les représente par *matrice d'adjacence*, par *matrice d'incidence* ou encore par *liste d'adjacence*.

Le lecteur pourra se référer à l'ouvrage « Graph Theory » de *Diestel* [Die12] et « Graphs and Hypergraphs » de *Berge* [Ber76] pour plus de détails concernant les notions et généralités sur les graphes.

1.1.1 Notations

Un graphe $G = (V, E)$ est un couple d'un **ensemble de sommets** V et d'un **ensemble d'arêtes** E , une arête étant une paire de sommets. On note généralement $V(G)$ l'ensemble des sommets d'un graphe G , $E(G)$ l'ensemble des arêtes d'un graphe G , $n(G)$ le nombre de sommets de G et $m(G)$ le nombre d'arêtes de G ou plus simplement V , E , n et m lorsqu'il n'y a aucune ambiguïté sur le graphe auquel ces objets font référence.

Étant donnés deux sommets u et $v \in V$, u est **adjacent** à v si l'arête $\{u, v\} \in E$.

On définit le **voisinage ouvert** d'un sommet $v \in V$ par $N_G(v) = \{u \mid \{u, v\} \in E\}$, et on définit le **voisinage fermé** (ou voisinage clos) de v par $N_G[v] = N_G(v) \cup \{v\}$.

Étant donné un sous-ensemble de sommets $W \subseteq V$, on définit $N_G(W) = \bigcup_{w \in W} N_G(w)$, $N_G[W] =$

$N_G(W) \cup W$, et étant donné un sommet u , on définit $N_W(u) = N_G(u) \cap W$ et $N_W[u] = N_G[u] \cap W$.

Le **degré** d'un sommet v est défini par $d_G(v) = |N(v)|$. On définit le degré maximum de G par $\Delta(G) = \max\{d_G(v) | v \in V(G)\}$. Un graphe G est *complet* si ses sommets sont tous de degré $n - 1$.

S'il n'y a pas d'ambiguïté sur le graphe auquel ces notions font référence on pourra supprimer l'indice G des notations précédentes.

Étant donné un sous-ensemble de sommets $W \subseteq V$, on appelle **sous-graphe induit** par W , noté $G[W]$, le graphe défini par $G[W] = (W, \{\{u, v\} | u \in W, v \in W \text{ et } \{u, v\} \in E\})$.

Un sous-ensemble de sommets $C \subseteq V$ est une **clique** de G si le graphe induit par C est un *graphe complet*. Un sous-ensemble de sommets $S \subseteq V$ est un **stable** de G si le graphe induit par S ne contient pas d'arête.

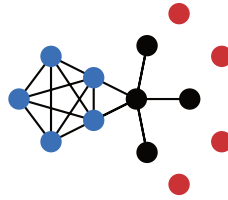


FIGURE 1.1 – Les sommets bleus forment une clique, les rouges forment un stable.

Une suite de sommets distincts $(u_1, u_2, \dots, u_k)$ est un **chemin** si $\{u_i, u_{i+1}\} \in E$ pour tout $i \in \{1, \dots, k-1\}$. La longueur d'un chemin est égale au nombre d'arêtes qui le compose. On définit de manière similaire un **cycle** comme un chemin $(u_1, u_2, \dots, u_k)$, $k > 2$ et l'arête $\{u_k, u_1\}$ appartient au graphe. La longueur d'un cycle est alors égale au nombre d'arêtes qui le compose.

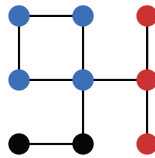


FIGURE 1.2 – Le graphe induit par les sommets rouges est un chemin de longueur 2, celui induit par les sommets bleus est un cycle de longueur 4.

Un graphe est **connexe** si pour chaque paire de sommets il existe un chemin les reliant. Une **composante connexe** de G est un sous-graphe induit $G' = (V', E')$ de G qui est connexe et maximal : pour tout $u \in V \setminus V'$, le graphe induit par $V' \cup \{u\}$ n'est pas connexe.

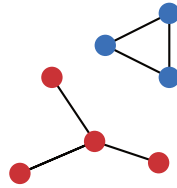
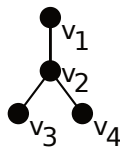
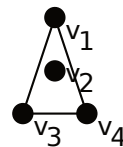


FIGURE 1.3 – Un graphe avec deux composantes connexes : le graphe induit par les sommets bleus et le graphe induit par les sommets rouges.

Le **complémentaire** de G , noté $\overline{G}$, est le graphe contenant les mêmes sommets que G , et tel que deux sommets sont adjacents dans $\overline{G}$ si et seulement s'ils ne sont pas adjacents dans G .



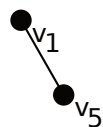
(a)



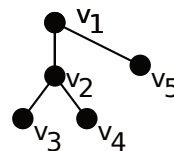
(b)

FIGURE 1.4 – (a) Un graphe G ; (b) Le complémentaire $\overline{G}$ de G .

L'**union** de deux graphes G et H , notée $G \cup H$, a comme ensemble de sommets $V(G) \cup V(H)$ et comme ensemble d'arêtes $E(G) \cup E(H)$.



(a)



(b)

FIGURE 1.5 – (a) Un graphe H ; (b) L'union $G \cup H$ du graphe G de la figure 1.4a et du graphe H .

Une **coloration propre** de G est une affectation d'une couleur à chaque sommet telle que deux sommets adjacents aient des couleurs différentes. Le *nombre chromatique* de G , noté $\chi(G)$ est le nombre minimum de couleurs nécessaires pour colorer G par une coloration propre.

Un sous-ensemble d'arêtes $E' \subseteq E$ est un **couplage** pour G si les arêtes de E' sont *deux à deux non-incidentes*. Autrement dit, le sous-graphe (V, E') est constitué d'une collection de sommets

isolés et de chemins de longueur 1. On note $\nu(G)$ la *taille d'un couplage maximum* pour G .

Un sous-ensemble de sommets $V' \subseteq V$ est un **ensemble dominant** si $N[V'] = V$. Autrement dit, tout sommet de V soit est dans V' , soit possède un voisin dans V' . On note $\gamma(G)$ la *taille du plus petit ensemble dominant* de G .

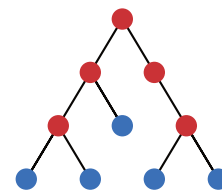
1.1.2 Les classes de graphes

Une classe de graphes peut être vue comme une famille de graphes partageant des caractéristiques structurelles communes. Il existe un très grand nombre de classes de graphes différentes. Nous décrivons quelques classes de graphes utiles à la compréhension des prochains chapitres.

Arbre et forêt

Une **forêt** est un graphe ne contenant pas de cycle. Un **arbre** est un graphe connexe et sans cycle. Tout sommet de degré 2 ou plus est un *nœud interne* et tout sommet de degré au plus 1 est une *feuille*.

L'arbre G est *enraciné* si on a désigné un sommet r comme étant sa *racine*. Dans un tel arbre, pour tout chemin $(r, \dots, v_1, v_2)$, v_1 est le *père* de v_2 et v_2 est le *fil*s de v_1 . La *profondeur* d'un arbre enraciné est la longueur du plus long chemin reliant la racine à une feuille. On note généralement T un arbre et $T(r)$ un arbre enraciné en un sommet r .

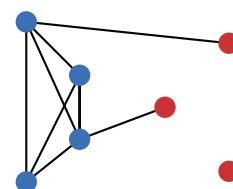


On donne un exemple d'arbre dans la figure ci-contre, dont les sommets bleus sont des feuilles, et les sommets rouges sont des nœuds internes.

Graphe split

Un graphe est **split** si on peut partitionner ses sommets en deux sous-ensembles V_1 et V_2 tels que V_1 est une clique de G et V_2 est un stable de G .

Le graphe de la figure ci-contre est un graphe split, les sommets bleus formant une clique et les sommets rouges formant un stable.



Graphe biparti

Un graphe $G = (V, E)$ est **biparti** si on peut partitionner V en deux sous-ensembles V_1 et V_2 , tels que V_1 et V_2 sont des stables de G . Le graphe G est **biparti complet** si tout sommet de V_1 est adjacent à tout sommet de V_2 .

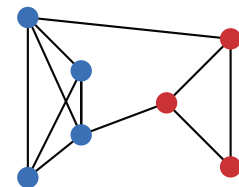
On note généralement K_{n_1, n_2} un graphe biparti complet dont le premier sous-ensemble contient n_1 sommets, et le second sous-ensemble contient n_2 sommets.

Le graphe de la figure ci-contre est biparti complet, dont les sommets sont partitionnés en sommets bleus et en sommets rouges. Chaque arête de ce graphe relie un sommet bleu à un sommet rouge, et tout sommet bleu est adjacent à tout sommet rouge.



Graphe cobiparti

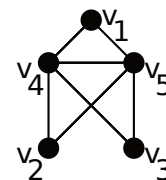
Un graphe est **cobiparti** si on peut partitionner ses sommets en deux sous-ensembles V_1 et V_2 tels que V_1 et V_2 sont des cliques de G . Le graphe ci-contre est un exemple de graphe cobiparti, dont les sommets sont partitionnés en sommets bleus et en sommets rouges. Les graphes induits par chacun de ces sous-ensembles sont des cliques.



Cographe

Un **cographe** est un graphe pouvant être construit selon les règles suivantes :

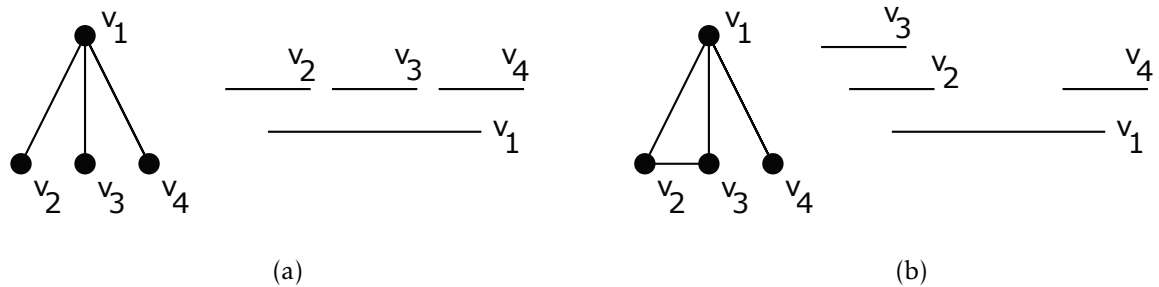
- Un sommet isolé est un cographe ;
- Le complémentaire d'un cographe est un cographe ;
- L'union de deux cographes disjoints est un cographe.



Le graphe ci-contre se construit par l'application successive des règles suivantes : ajout d'un sommet isolé v_1 , union avec un sommet isolé v_2 , union avec un sommet isolé v_3 , complémentaire, union avec un sommet isolé v_4 , union avec un sommet isolé v_5 , complémentaire.

Graphe d'intervalles

Un graphe $G = (V, E)$ est un **graphe d'intervalles** si on peut mettre en bijection V avec une famille d'intervalles des réels, tel que deux sommets soient adjacents si et seulement si leur intervalles



correspondants s'intersectent. On donne dans la figure (a) ci-dessus un exemple de graphe d'intervalles. On définit deux fonctions $l : V \rightarrow \mathbb{R}$ (respectivement $r : V \rightarrow \mathbb{R}$) associant à chaque sommet l'extrémité gauche (respectivement l'extrémité droite) de son intervalle correspondant. Ces deux fonctions définissent un *modèle d'intervalles* pour G .

Un modèle d'intervalles est *normalisé* si les intervalles associés aux sommets ont des extrémités distinctes, et les valeurs des extrémités sont dans $\{1, \dots, 2n\}$.

G est un **graphe d'intervalles propres** s'il existe un modèle d'intervalles pour G tel qu'aucun intervalle n'est inclus dans un autre. La figure (b) ci-dessus est un exemple de graphe d'intervalles propres.

Graphe triangulé

Un graphe est **triangulé** si tout cycle de longueur au moins 4 possède une *corde*, c'est-à-dire une arête reliant entre eux deux sommets non-consécutifs sur le cycle.

Graphe sans griffe

Un **graphe sans griffe** est un graphe ne contenant aucune *griffe* comme sous-graphe induit, c'est-à-dire aucun $K_{1,3}$ (un graphe biparti complet dont la première partie contient un sommet, et la seconde trois sommets).

Graphe d'intervalles circulaires

Un **graphe d'intervalles circulaires** est un graphe pour lequel on peut mettre en bijection ses sommets avec une famille d'arcs d'un cercle, deux sommets étant adjacents si et seulement si leurs arcs correspondant s'intersectent.

Comme nous l'avons précisé au début, cette liste n'est pas exhaustive. De nombreuses *relations d'inclusion* lient les différentes classes de graphes entre elles. Parmi les classes de graphes que nous venons de citer on vérifie aisément que tout graphe d'intervalles est également un graphe d'intervalles circulaires ou encore un graphe triangulé, et qu'un graphe d'intervalles propres est également un graphe d'intervalles. La connaissance des inclusions entre les classes de graphes est intéressante lorsque l'on s'intéresse à la complexité de problèmes ou à la conception d'algorithmes restreints à certaines classes de graphes : un algorithme résolvant un problème sur une classe donnée résout aussi le problème sur toutes ses sous-classes. Par ailleurs, si on a un algorithme résolvant un problème sur une classe donnée, il peut être intéressant d'étudier ce qu'il se passe pour l'une de ses superclasses. Nous donnons dans la figure 1.7 quelques relations d'inclusion entre des classes de graphes.

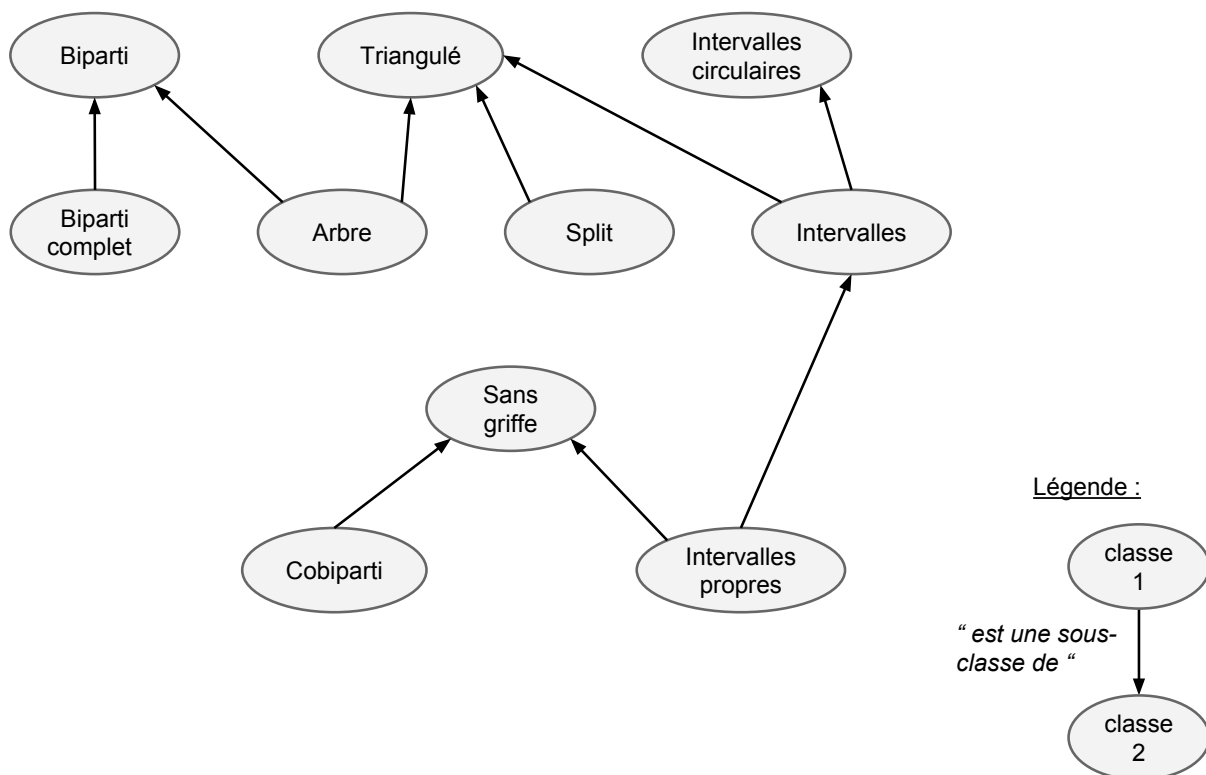


FIGURE 1.7 – Quelques inclusions entre quelques classes de graphes

Pour plus d'information concernant les classes de graphes, le lecteur pourra se référer à l'ouvrage « Algorithmic Graph Theory and Perfect Graphs » de *Golumbic* [Gol04] ainsi qu'à l'ouvrage « Graph Classes : A Survey » de *Brandstädt et al.* [BLS99]. Le site internet graphclasses.org en liste de nombreuses, ainsi que les relations les liant, et la complexité de divers problèmes sur ces classes (en date du 26 mars 2015, le site liste 1511 classes de graphes, ainsi que 179111 inclusions).

1.2 Conception d'algorithmes et analyse de leur efficacité

Nous supposons le lecteur familier avec les notions élémentaires de calculabilité et complexité. Nous le référons à l'ouvrage « Introduction à la Calculabilité » de *Wolper* [Wol91], à l'ouvrage « Computers and Intractability : A Guide to the Theory of NP-Completeness » de *Garey et Johnson* [GJ79] ainsi qu'à l'ouvrage « Computational Complexity » de *Papadimitriou* [Pap07].

1.2.1 Complexité des algorithmes : notation asymptotique

Lors de l'analyse de la complexité d'un algorithme, on utilise des notations afin de borner asymptotiquement le temps d'exécution d'un algorithme. Les fonctions que nous décrivons maintenant sont des fonctions de $\mathbb{R}^+ \rightarrow \mathbb{R}^+$.

En notant $T(n)$ le temps d'exécution d'un algorithme dont l'entrée est de taille n une fonction $f(n)$ est un *majorant asymptotique* de $T(n)$, que l'on note $\mathcal{O}(f(n))$, s'il existe deux constantes strictement positives c et n_0 telles que pour tout $n \geq n_0$ on ait $T(n) \leq c \cdot f(n)$. On dit alors que $T(n) \in \mathcal{O}(f(n))$.

De manière similaire une fonction $g(n)$ est un *minorant asymptotique* de $T(n)$, noté $\Omega(g(n))$, s'il existe deux constantes strictement positives c et n_0 telles que pour tout $n \geq n_0$ on ait $T(n) \geq c \cdot g(n)$. On dit alors que $T(n) \in \Omega(g(n))$.

Notons que si $T(n) \in \mathcal{O}(f(n))$ et $T(n) \in \Omega(f(n))$, alors on dit que $T(n) \in \Theta(f(n))$.

La fonction $T(n)$ est *asymptotiquement négligeable* devant une fonction $h(n)$ si pour toute constante strictement positive c il existe une constante strictement positive n_0 telle que pour tout $n \geq n_0$ on ait $T(n) \leq c \cdot h(n)$. On dit alors que $T(n) \in o(h(n))$. Notons que si $h(n)$ ne s'annule pas en $+\infty$ on a $\frac{T(n)}{h(n)} \rightarrow 0$ lorsque $n \rightarrow +\infty$.

À ces notations conventionnelles nous ajoutons la notation $\mathcal{O}^*$: étant donnée une fonction $f(n)$, $T(n) \in \mathcal{O}^*(f(n))$ s'il existe un polynôme *poly* tel que $T(n) \in \mathcal{O}(f(n) \cdot \text{poly}(n))$.

1.2.2 Problèmes de décision : les classes P et NP

Un problème de décision est un problème dont la réponse est soit « oui », soit « non ». Un problème de décision est dans P si on peut le résoudre en temps polynomial en fonction de la taille de son entrée. Il admet donc un algorithme le résolvant dont le temps d'exécution est en $\mathcal{O}(n^c)$, n étant la taille de l'entrée et c étant une constante. La classe NP est l'ensemble des problèmes pour

lesquels on peut vérifier la validité d'un certificat pour ce problème (c'est-à-dire une solution acceptante de ce problème) en temps polynomial.

Problèmes NP-difficiles, NP-complets et réductions

Un problème est *NP*-difficile si on peut réduire tout problème *NP* à lui via une *réduction polynomiale*. Un problème est *NP*-complet s'il est *NP*-difficile et qu'il appartient à *NP*. Les problèmes *NP*-complets sont les problèmes les plus « durs à résoudre » parmi les problèmes de *NP*.

Cook et Levin montrent indépendamment en 1971 que le problème 3-SAT, défini par :

Problème 3-SAT

- entrée.** Une formule logique en forme normale conjonctive et dont chaque clause contient au plus 3 variables
- sortie.** Existe-t-il une affectation de valeurs pour les variables rendant la formule vraie ?

est *NP*-complet [Coo71].

Montrer qu'un problème est *NP*-complet se fait généralement en deux étapes : on montre tout d'abord qu'il appartient à *NP*, en montrant qu'un certificat pour ce problème se vérifie en temps polynomial. On vérifie ensuite qu'un problème connu pour être *NP*-complet se réduit polynomialement à notre problème. Rappelons qu'un problème *A* se réduit polynomialement à un problème *B* s'il existe un algorithme prenant en entrée une instance $\mathcal{I}_A$ de *A* et la transformant en une instance équivalente $\mathcal{I}_B$ pour le problème *B*. Les deux instances sont équivalentes si la réponse du problème *A* sur l'instance $\mathcal{I}_A$ est la même que celle du problème *B* sur l'instance $\mathcal{I}_B$. De plus, l'algorithme de réduction doit être de complexité polynomiale en temps en fonction de la taille de l'instance en entrée.

La question $P=NP$

Un des grands problèmes ouverts en informatique depuis ces 50 dernières années est de savoir si $P = NP$, ou autrement dit s'il est possible de résoudre le problème 3-SAT ou tout autre problème *NP*-complet en temps polynomial. Ainsi, montrer qu'il existe un algorithme déterministe polynomial résolvant un problème *NP*-complet permettrait de résoudre tout problème de *NP* en temps polynomial, impliquant par conséquent que $P=NP$. La question « $P = NP?$ » est l'un des sept problèmes du millénaire, problèmes qui furent posés par l'institut *Clay* en 2000. L'institut offre en échange de la résolution d'un de ces problèmes 1000000 \$¹. Parmi ces problèmes, le problème

1. Pour l'anecdote, parmi ces sept problèmes, seule la conjecture de Poincaré a été à ce jour résolue par *Perelman* qui refusa la médaille Fields ainsi que la somme offerte.

$P = NP$ se distingue des autres notamment par la simplicité de son énoncé : existe-t-il un algorithme pour résoudre un problème NP -complet en temps polynomial ? Cette question demeure néanmoins ouverte, bien qu'il soit admis par un certain nombre d'informaticiens que P est différent de NP .

1.2.3 Problèmes d'optimisation et d'énumération

Il existe des problèmes d'autre nature que les problèmes de décision dont nous avons précédemment parlé. Certains problèmes ont pour but de calculer une solution qui est optimale selon certains critères : ce sont les **problèmes d'optimisation**. D'autres ont pour but de lister un ensemble de solutions admises par un problème respectant certaines propriétés : ce sont les **problèmes d'énumération**. Le temps d'exécution d'un algorithme conçu dans le but de résoudre un problème d'énumération a un temps d'exécution borné inférieurement par le nombre d'objets à énumérer.

Exemple : le cas du problème ENSEMBLE DOMINANT.

Soit le problème de décision **ENSEMBLE DOMINANT** :

Problème **ENSEMBLE DOMINANT**

entrée. Un graphe $G = (V, E)$; un entier (positif) k .

question. Existe-t-il un sous-ensemble de sommets $V' \subseteq V$, de taille au plus k , et tel que tout sommet de V soit dans V' , soit possède au moins un voisin dans V' ?

La question d'optimisation correspondant à ce problème consiste à rechercher la taille du plus petit ensemble dominant parmi tous les ensembles dominants valides de G . Le problème correspondant est le problème **ENSEMBLE DOMINANT MINIMUM**, défini par :

Problème **ENSEMBLE DOMINANT MINIMUM**

entrée. Un graphe $G = (V, E)$.

sortie. La taille d'un plus petit ensemble dominant de G .

Il existe un certain lien entre ces deux problèmes du point de vue de la complexité : soit (G, k) une entrée pour **ENSEMBLE DOMINANT**. **ENSEMBLE DOMINANT** pour (G, k) est vrai si et seulement si la sortie d'**ENSEMBLE DOMINANT MINIMUM** pour G est inférieure ou égale à k . Réciproquement $n+1$ appels à un algorithme résolvant **ENSEMBLE DOMINANT** suffisent à résoudre **ENSEMBLE DOMINANT MINIMUM**.

Un problème d'énumération correspondant à **ENSEMBLE DOMINANT** consiste à énumérer les ensembles dominants de G qui sont minimaux par inclusion (aucun ensemble dominant n'est strictement inclus dans l'un de ces ensembles). Ce problème, que nous appellerons **ENSEMBLES DOMINANTS MINIMAUX**, est défini de la manière suivante :

Problème **ENSEMBLES DOMINANTS MINIMAUX**

entrée. Un graphe $G = (V, E)$.

sortie. Tous les ensembles dominants de G qui sont minimaux.

Remarquons qu'à partir d'un algorithme résolvant ce problème on en déduit facilement une solution pour les deux problèmes précédents : un algorithme pour **ENSEMBLE DOMINANT** retournera oui sur une entrée (G, k) si et seulement s'il existe parmi les ensembles dominants minimaux énumérés un ensemble de taille au plus k . De même, tout ensemble dominant minimum étant également un ensemble dominant minimal, on déduit également d'un algorithme résolvant **ENSEMBLES DOMINANTS MINIMAUX** un algorithme pour **ENSEMBLE DOMINANT MINIMUM**. Notons finalement que tout algorithme résolvant **ENSEMBLES DOMINANTS MINIMAUX** sera exponentiel en n car il existe des graphes ayant une quantité exponentielle d'ensembles dominants minimaux. Cela arrivera même pour des classes de graphes pour lesquelles le problème **ENSEMBLE DOMINANT** est polynomial, comme les graphes d'intervalles (cf chapitre 3).

1.2.4 L'Hypothèse de Temps Exponentiel

L'*Hypothèse du Temps Exponentiel* (*ETH*) conjecture qu'on ne peut pas résoudre 3-SAT en temps sous-exponentiel par rapport au nombre de variables.

Hypothèse 1.1. *Impagliazzo et Paturi [IP01]*

Pour tout $k \geq 3$, k -SAT n'admet pas d'algorithme en temps $2^{o(n)}$, n étant le nombre de variables de la formule.

Impagliazzo, Paturi et Zane montrent dans [IPZ01] que l'on peut résoudre 3-SAT en temps $2^{o(n)}$ si et seulement si on peut le résoudre en $2^{o(m)}$, m étant le nombre de clauses. Ceci a ouvert la voie à une série de résultats montrant que sous l'*ETH* un certain nombre de problèmes de graphes ne peuvent pas être résolus en temps $2^{o(n)}$, n étant le nombre de sommets du graphe. C'est le cas du problème du problème 3-COLORATION, mais aussi du problème **ENSEMBLE DOMINANT** même pour des graphes de degré maximum 6 [FKW04]. Ce type de résultat s'obtient typiquement par des réductions linéaires, ou plus généralement par des familles de réductions sous-exponentielles (*SERF*). Nous utiliserons le résultat sus-cité de *Fomin et al.* pour montrer que, sous l'*ETH*, le pro-

blème **ENSEMBLE CONNEXE TROPICAL MINIMUM** ne peut pas être résolu en temps sous-exponentiel pour la classe des arbres.

1.3 Résoudre un problème difficile

Donnons avant tout plus de détails concernant la notion de *problèmes difficiles* : Nous qualifions de *problème difficile* un problème n'admettant pas (a priori pour certain d'entre eux) d'algorithmes exacts les résolvant en temps polynomial. Distinguons deux catégories de problèmes difficiles. La première concerne les problèmes *NP*-difficiles de décision ou d'optimisation (dont le problème de décision associé est *NP*-difficile). La seconde concerne les problèmes d'énumération pour lesquels le nombre d'objets à énumérer est exponentiel en n .

Nous nous intéressons à la conception d'algorithmes exacts afin de résoudre un problème difficile en temps le plus raisonnable possible en fonction de n , le nombre de sommets du graphe en entrée. La raison pour laquelle nous avons choisi n comme paramètre est que les problèmes abordés ici peuvent être trivialement résolus par un algorithme de recherche exhaustive en temps $\mathcal{O}^*(2^n)$ (pour les problèmes **ENSEMBLE CONNEXE TROPICAL MINIMUM** et **ENSEMBLES DOMINANTS MINIMAUX**) et de $\mathcal{O}^*(3^n)$ (pour **DOMINATION ROMAINE FAIBLE MINIMUM** sur les graphes quelconques). Le but sera alors d'obtenir un algorithme de complexité meilleure que la recherche exhaustive.

1.3.1 Différentes techniques algorithmiques

Programmation Dynamique

Intuitivement, l'idée de cette méthode est de mémoriser les calculs effectués durant l'exécution de l'algorithme afin de pouvoir calculer la solution d'un problème en réutilisant les solutions précédemment calculées des sous-problèmes. On fixe à l'avance l'ordre dans lequel on traite les sous-problèmes, cet algorithme ne sera donc généralement pas récursif. Pour calculer la solution d'une instance, nous nous servons des résultats d'instances plus petites. Par recomposition de solutions précédemment calculées et mémorisées, on en déduit la solution du problème.

C'est une méthode généralement associée aux problèmes d'optimisation. On peut caractériser le développement d'une méthode de programmation dynamique en 4 étapes :

- **Caractériser** la structure d'une solution optimale ;
- **Définir** récursivement la valeur d'une solution optimale ;
- **Calculer** la valeur d'une solution optimale de manière ascendante ;
- **Construire** une solution optimale à partir des informations calculées.

Le principal avantage quant à l'utilisation de cette méthode algorithmique est un gain en temps conséquent, par rapport à l'utilisation d'une méthode de type *Diviser pour Régner*. Alors que dans *Diviser pour Régner*, nous n'allons pas conserver les résultats des sous-instances, nous obligeant de calculer à chaque fois ces sous-instances, en *Programmation Dynamique*, chaque sous-instance du problème qui sera rencontrée ne sera calculée qu'une seule et unique fois. Le gain de temps se fait néanmoins au prix de l'espace nécessaire afin de stocker les calculs intermédiaires.

Brancher et Réduire

La technique *Brancher et Réduire* est un paradigme de programmation basé sur la technique de branchement. Typiquement les algorithmes utilisant ce paradigme contiennent deux types de règles :

- **Règle de réduction.** On distingue deux types de règles de réduction : les premières ont pour but de simplifier l'instance du problème (notamment en réduisant la taille de l'instance) sans que cela n'influe sur le résultat calculé par l'algorithme. Les secondes, souvent qualifiées de *règles d'arrêt*, servent à stopper l'algorithme (lorsqu'une solution a été calculée, ou lorsque le sous-arbre courant des appels n'admet pas de solution acceptable).
- **Règle de branchement.** Utilisée pour résoudre une instance du problème en *résolvant récursivement* des sous-instances du problème.

La description d'un algorithme basé sur ce paradigme de programmation est relativement simple à donner, et consiste en la description des règles le composant (cf l'algorithme `StableMax` de la sous-section 1.3.2). Sa correction est aussi généralement simple à montrer, et est basée sur la correction de ses règles de branchement et de réduction. Finalement, l'analyse de la complexité d'un tel algorithme est la partie la plus délicate.

Notons que la plupart des algorithmes décrits dans cette thèse sont basés sur ce paradigme de programmation. Bien que cette technique soit ancienne, c'est une technique puissante permettant de produire des algorithmes exacts exponentiels efficaces.

1.3.2 Analyse du temps d'exécution

Lors de la conception d'un algorithme, il est important de pouvoir **analyser son temps d'exécution**, c'est-à-dire le nombre d'opérations que cet algorithme effectue en fonction d'un paramètre comme la taille de son entrée, pour avoir une idée de son efficacité, et ainsi pouvoir le comparer à d'autres algorithmes. Dans le cas des algorithmes itératifs, leur temps d'exécution se déduit

généralement assez facilement à partir de leur structure. Pour les algorithmes récursifs de branchement un tel raisonnement n'est pas applicable, et il nous faut une technique plus sophistiquée pour les analyser et ainsi établir une borne sur leur temps d'exécution.

Cette section est dédiée à l'analyse du temps d'exécution d'un algorithme exact exponentiel basé sur le paradigme de programmation *Brancher et Réduire*. Nous présenterons tout d'abord l'analyse standard de la complexité d'un tel algorithme, puis une analyse non standard, via la technique *Mesurer pour Conquérir* introduite par Fomin et al. dans [FGK09].

Les analyses que nous décrivons maintenant sont basées sur celles décrites dans l'ouvrage « Exact Exponential Algorithms » de Fomin et Kratsch [FK10], auquel le lecteur pourra se référer, notamment concernant les preuves de correction de ces techniques, que nous ne détaillerons pas ici.

Afin d'illustrer cet analyse, considérons le problème d'optimisation suivant :

Problème STABLE MAXIMUM

entrée. Un graphe G

sortie. La taille d'un plus grand sous-ensemble de sommets qui sont deux-à-deux non-adjacents.

Ainsi que l'algorithme de branchement suivant afin de résoudre **STABLE MAXIMUM** :

Algorithme 1 : $\text{StableMax}(G = (V, E))$

Entrées : Un graphe $G = (V, E)$.

Sorties : La taille d'un plus grand stable de G .

si $\exists v \in V$ avec $d(v) = 0$ **alors**

└ **retourner** $1 + \text{StableMax}(G[V \setminus \{v\}])$

si $\exists v \in V$ avec $d(v) = 1$ **alors**

└ **retourner** $1 + \text{StableMax}(G[V \setminus N[v]])$

si $\Delta(G) \geq 3$ **alors**

└ choisir un sommet v de degré maximum dans G
└ **retourner** $\max(1 + \text{StableMax}(G[V \setminus N[v]]), \text{StableMax}(G[V \setminus \{v\}]))$

si $\Delta(G) \leq 2$ **alors**

└ calculer en temps polynomial la cardinalité $s(G)$ d'un stable maximum pour G
└ **retourner** $s(G)$

Analyse standard

Remarquons tout d'abord qu'à chaque appel récursif de l'algorithme `StableMax` on modifie le graphe G (en supprimant un ou plusieurs sommets).

L'ensemble des appels récursifs effectués par l'algorithme forme un arbre. La racine de cet arbre correspond au premier appel. Chaque nœud (appel) a 0, 1 ou 2 fils. Les feuilles correspondent à la règle d'arrêt (cas où $\Delta(G) \leq 2$). Les nœuds avec un seul fils correspondent aux règles de réduction (cas où $d(v) = 0$ ou $d(v) = 1$). Les autres nœuds ont exactement deux fils, et correspondent à l'unique règle de branchement de l'algorithme ($\Delta(G) \geq 3$ et G n'a pas de sommet de degré 0 ou 1). Notons $T(n)$ le nombre maximum de feuilles de l'arbre des appels lorsque le graphe en entrée a n sommets. Il est facile de se convaincre que la complexité en temps de l'algorithme est $\mathcal{O}^*(T(n))$, car la profondeur de l'arbre est polynomiale (ce qui fait que l'arbre a $\mathcal{O}^*(T(n))$ nœuds et que chaque appel a un coût polynomial. Estimons maintenant le nombre $T(n)$ de feuilles de l'arbre des appels récursifs via l'étude de récurrences.

Typiquement, étant donnée une *règle de branchement* de l'algorithme, on définit une récurrence de la forme $T(n) \leq \sum_{i=1}^k T(n - r_i)$, k étant le nombre de branches de la règle de branchement et r_i étant le nombre de sommets supprimés de l'instance en entrée du sous-problème correspondant à la branche. Dans l'exemple de l'algorithme `StableMax`, il n'y a qu'une seule règle de branchement, qui est appliquée lorsque $\Delta(G) \geq 3$ et lorsque G ne contient pas de sommets de degré 0 ou 1 ; la récurrence associée est $T(n) \leq T(n - (d(v) + 1)) + T(n - 1)$ avec $d(v) \geq 3$. Le pire cas est lorsque $d(v) = 3$. On peut donc écrire $T(n) \leq T(n - 4) + T(n - 1)$. Pour résoudre des récurrences de type $T(n) \leq \sum_{i=1}^k T(n - r_i)$, on sait, par des méthodes d'analyse mathématique, qu'elles satisfont $T(n) \in \mathcal{O}^*(c^n)$ où c est l'unique racine réelle positive du polynôme caractéristique $\sum_{i=1}^k \alpha^{-r_i} - 1$.

Pour la récurrence précédente, le polynôme caractéristique est $\alpha^{-4} + \alpha^{-1} - 1$. La seule racine réelle positive est $c < 1.3803$. On en déduit que le temps d'exécution de l'algorithme `StableMax` est $\mathcal{O}^*(1.3803^n)$.

La recherche de la *racine positive* de la récurrence peut s'effectuer par exemple avec le logiciel *Sage* disponible à l'adresse <http://www.sagemath.org>.

La commande suivante :

```
sage : find_root(x^-4 + x^-1 - 1 == 0, 1, 2)
```

nous retourne la racine positive du polynôme caractéristique associé à notre récurrence, soit $c < 1.3803$.

On note par $(r_1, r_2, \dots, r_k)$ le **vecteur de branchement** d'une récurrence de la forme $T(n) \leq \sum_{i=1}^k T(n-r_i)$. La racine positive du polynôme caractéristique associé est appelée **facteur de branchement**, et sera notée $\tau(r_1, r_2, \dots, r_k)$.

Le vecteur de branchement associé à notre récurrence est $(1, 4)$, et $\tau(1, 4) \leq 1.3803$.

Ceci montre comment analyser des algorithmes ayant une seule règle de branchement. Lorsque les algorithmes ont plusieurs règles de branchement, nous considérerons chaque vecteur de branchement, et nous notons c le plus grand des facteurs de branchement. Il a été montré que le temps d'exécution de l'algorithme est $\mathcal{O}^*(c^n)$.

Mesurer pour Conquérir

Remarquons tout d'abord que dans l'algorithme `StableMax`, le temps d'exécution est obtenu à partir de la racine du polynôme caractéristique associé à l'unique règle de branchement, c'est-à-dire le cas où le graphe est de degré maximum 3 ou plus (et lorsque ce graphe ne contient pas de sommet de degré 0 ou 1, au quel cas on applique les deux premières règles de réduction). Une question naturelle se pose alors : la borne de $\mathcal{O}^*(1.3803^n)$ est elle serrée ? Nous verrons qu'au contraire, elle est fortement surestimée.

Prenons un exemple tout simple : soit G un graphe avec $\Delta(G) \geq 3$ et ne contenant que des sommets de degré au moins 2. Si $\Delta(G) \geq 4$ nous gagnerons au moins 5 sommets sur l'une des branches. Si $\Delta(G) = 3$, les voisins de v auront un degré au plus égal à 2 sur la branche où l'on supprime v , on ne branchera donc pas dessus, et ils n'interviendront pas dans d'autres branchements. Dans les deux cas, on gagne plus que ce que suggérait notre analyse.

L'analyse standard que nous avons décrite précédemment ne considère que le nombre de sommets présents dans le graphe, mais ne considère pas que lors d'un branchement, le degré de certains sommets diminue. L'idée de l'analyse non standard *Mesurer pour Conquérir* est la suivante : au lieu de ne considérer dans la récurrence que le nombre de sommets présents dans le graphe, on prend également en compte le degré des sommets du graphe pour obtenir une borne plus précise.

Nous décrivons maintenant cette approche appliquée à l'algorithme `StableMax`.

Soit n_2 le nombre de sommets de G de degré 2, n_3 le nombre de sommets de G de degré 3 et $n_{\geq 4}$ le nombre de sommets de G de degré 4 ou plus. Soient β_2, β_3 et $\beta_{\geq 4} \in [0, 1]$ des constantes que nous calculerons ultérieurement. On définit la mesure $\mu(G)$ du graphe G par $\mu(G) = \beta_2 \cdot n_2 + \beta_3 \cdot n_3 + \beta_{\geq 4} \cdot n_{\geq 4}$. Notons que $\mu(G) \leq n$.

On refait maintenant l'analyse de l'algorithme en notant $T(\mu)$ le temps d'exécution de l'algorithme sur un graphe de mesure $\mu(G)$. Soit v le sommet choisi par l'algorithme, et notons d_v^2 le nombre de voisins de v de degré 2, d_v^3 le nombre de voisins de v de degré 3, d_v^4 le nombre de

voisins de v de degré 4 et $d_v^{\geq 5}$ le nombre de voisins de v de degré 5 ou plus. On distingue le cas où v est de degré 3, de degré 4 et de degré 5 ou plus :

v est de degré 3. Étant donné que G ne contient que des sommets de degré 2 ou plus et que v a été choisi comme étant un sommet de degré maximum dans G , si $d(v) = 3$ alors tout sommet de G est de degré 2 ou 3.

- Dans la première branche du branchement de l'algorithme on supprime v (qui est de degré 3) et ses voisins (qui sont de degré 2 ou 3). La mesure de $G[V - N[v]]$ diminue d'au moins $\beta_3 + d_v^2\beta_2 + d_v^3\beta_3$.
- Dans la seconde branche on ne supprime que v . On diminue alors de 1 le degré de ses voisins. Remarquons que ses voisins de degré 2 seront supprimés tout de suite, par l'application des règles de réduction. Le nouveau graphe, après application des réductions, aura une mesure diminuée d'au moins $\beta_3 + d_v^2\beta_2 + d_v^3(\beta_3 - \beta_2)$.

On obtient la récurrence suivante :

$$T(\mu) \leq T(\mu - \beta_3 - d_v^2\beta_2 - d_v^3\beta_3) + T(\mu - \beta_3 - d_v^2\beta_2 - d_v^3\beta_3 + d_v^3\beta_2).$$

v est de degré 4. Pour les mêmes arguments que le cas précédent, v n'a que des voisins de degré 2, 3 et 4.

- Dans la première branche on supprime v et ses voisins.
- Dans la seconde branche on ne supprime que v . On diminue alors de 1 le degré de ses voisins.

La récurrence obtenue est la suivante :

$$T(\mu) \leq T(\mu - \beta_{\geq 4} - d_v^2\beta_2 - d_v^3\beta_3 - d_v^4\beta_{\geq 4}) + T(\mu - \beta_{\geq 4} - d_v^2\beta_2 - d_v^3\beta_3 + d_v^3\beta_2 - d_v^4\beta_{\geq 4} + d_v^4\beta_3)$$

v est de degré 5 ou plus. Pour les mêmes arguments que le cas précédent, v n'a que des voisins de degré 2 ou plus.

- Dans la première branche on supprime v et ses voisins.
- Dans la seconde branche on ne supprime que v . Diminue de 1 le degré de ses voisins. On ne peut rien déduire quant aux sommets de degré 5 ou plus, qui n'interviennent donc pas dans la récurrence.

La récurrence obtenue est la suivante :

$$T(\mu) \leq T(\mu - \beta_{\geq 4} - d_v^2\beta_2 - d_v^3\beta_3 - d_v^4\beta_{\geq 4} - d_v^{\geq 5}\beta_{\geq 4}) + T(\mu - \beta_{\geq 4} - d_v^2\beta_2 - d_v^3\beta_3 + d_v^3\beta_2 - d_v^4\beta_{\geq 4} + d_v^4\beta_3)$$

Finalement, il faut calculer les valeurs $\beta_2, \beta_3, \beta_{\geq 4}$ minimisant le plus grand des facteurs de branchement parmi toutes les récurrences précédemment décrites.

Pour ces récurrences le facteur de branchement obtenu est 1.2905 (pour les valeurs $\beta_2 = 0.5967$, $\beta_3 = 0.9287$ et $\beta_{\geq 4} = 1$). Finalement nous en déduisons que l'algorithme *StableMax* s'exécute en temps $\mathcal{O}^*(1.2905^n)$, améliorant donc l'analyse précédente qui était de $\mathcal{O}^*(1.3803^n)$.

Cette nouvelle mesure nous a permis d'obtenir une mesure *plus fine* de son temps d'exécution. L'algorithme n'a pas été modifié ; seule la mesure a été améliorée ! Il n'y a cependant aucune garantie que la nouvelle borne supérieure soit serrée ; une analyse plus fine ou encore une autre mesure non standard permettrait peut-être d'obtenir une borne encore plus précise. Une comparaison avec la borne inférieure sur le temps d'exécution de l'algorithme (que l'on obtient notamment en construisant une instance sur laquelle l'algorithme s'exécute en un certain nombre d'étapes) donne des indices sur la précision de cette analyse.

L'analyse *Mesurer pour Conquérir* a été appliquée dans une série de papiers afin d'améliorer l'analyse du temps d'exécution de certains algorithmes :

- Dans [FK10], Fomin et Kratsch améliorent l'analyse de plusieurs algorithmes :
 - Ils améliorent la borne sur le temps d'exécution d'un algorithme calculant un *stable maximum* dans un graphe (l'exemple que nous venons de décrire) en la réduisant de $\mathcal{O}(1.3803^n)$ à $\mathcal{O}(1.2905^n)$;
 - Ils améliorent la borne sur le temps d'exécution d'un algorithme calculant un *ensemble dominant minimum* dans un graphe en la réduisant de $\mathcal{O}(1.9052^n)$ à $\mathcal{O}(1.5259^n)$;
 - Ils améliorent la borne sur le temps d'exécution d'un algorithme calculant un *ensemble coupe-cycle minimum* dans un graphe en la réduisant de $\mathcal{O}(1.8907^n)$ à $\mathcal{O}(1.8899^n)$;
- Dans [FGK09], Fomin, Grandoni et Kratsch améliorent l'analyse de deux algorithmes :
 - Ils améliorent la borne d'exécution de l'algorithme de Grandoni décrit dans [Gra04, Gra06] calculant un *ensemble dominant minimum* en temps $\mathcal{O}(1.8026^n)$ et en espace exponentiel en la réduisant à $\mathcal{O}(1.5137^n)$.
 - Ils améliorent également la borne sur le temps d'exécution d'un algorithme calculant un *stable maximum* dans un graphe en la réduisant de $\mathcal{O}(1.2269^n)$ à $\mathcal{O}(1.2202^n)$;

D'autres exemples d'améliorations des bornes supérieures sur le temps d'exécution d'algorithmes sont données dans [FKK⁺09, BvR11, FGK05, FGPS08, BvR08, NvDvR09, FGK06].

1.4 Quelques problèmes classiques

Nous présentons dans cette section un ensemble de problèmes *NP*-complets fondamentaux sur les graphes, auxquels nous ferons référence tout au long de cette thèse.

Ensemble Dominant

Rappelons le problème **ENSEMBLE DOMINANT** :

Problème **ENSEMBLE DOMINANT**

entrée. Un graphe $G = (V, E)$; un entier k .

question. Existe-t-il un ensemble dominant de taille au plus k pour G ?

La version d'optimisation de ce problème consiste à calculer la taille du plus petit ensemble dominant d'un graphe. Un problème d'énumération associé consiste à lister tous les ensembles dominants minimaux qu'un graphe possède. Nous étudions ce problème d'énumération dans le chapitre 3.

De nombreuses variantes d'ensemble dominant existent. Un ensemble de variantes concerne la structure qu'un tel ensemble doit respecter : l'ensemble dominant doit être une clique, être connexe, ou encore être un stable.

Un autre type de variante concerne les problèmes de domination pour lesquels on cherche à construire une fonction de pondération pour les sommets du graphe, afin de dominer le graphe selon certaines règles.

Un exemple de telle variante est le problème **DOMINATION ROMAINE** :

Problème **DOMINATION ROMAINE**

entrée. Un graphe $G = (V, E)$; un entier k .

question. Existe-t-il une fonction $f : V \rightarrow \{0, 1, 2\}$ telle que tout sommet $v \in V$ tel que $f(v) = 0$ a au moins un voisin v' tel que $f(v') = 2$, et telle que $\sum_{v \in V} f(v) \leq k$?

Une variante similaire est le problème **DOMINATION ROMAINE FAIBLE** :

Problème **DOMINATION ROMAINE FAIBLE**

entrée. Un graphe $G = (V, E)$; un entier k .

question. Existe-t-il une fonction $f : V \rightarrow \{0, 1, 2\}$ telle que :

- pour tout sommet $v \in V$ tel que $f(v) = 0$ il existe $u \in N[v]$ tel que $f(u) \geq 1$ et l'ensemble $D = \{x : f_{u \rightarrow v}(x) \geq 1\}$ est un ensemble dominant de G , la fonction $f_{u \rightarrow v}$ étant définie par :

$$f_{u \rightarrow v}(x) = \begin{cases} 1 & \text{si } x = v; \\ f(u) - 1 & \text{si } x = u; \\ f(x) & \text{si } x \notin \{u, v\}. \end{cases}$$

- $\sum_{v \in V} f(v) \leq k$?

Nous étudions le problème d'optimisation associé à ce problème, dont le but est de construire une fonction de domination romaine faible de poids minimum, dans le chapitre 4.

Clique et Stable

Le problème **CLIQUE** est défini de la manière suivante :

Problème **CLIQUE**

entrée. Un graphe $G = (V, E)$; un entier k .

question. Existe-t-il un sous-ensemble de sommets V' tel que les sommets de V' soient deux-à-deux adjacents, et tel que $|V'| \geq k$?

Le problème **STABLE** est défini de la manière suivante :

Problème **STABLE**

entrée. Un graphe $G = (V, E)$; un entier k .

question. Existe-t-il un sous-ensemble de sommets V' tel que les sommets de V' soient deux-à-deux non adjacents, et tel que $|V'| \geq k$?

Remarquons que les notions de clique et de stable sont en quelque sorte complémentaires. Un graphe G admet une clique de taille k si et seulement si son complémentaire $\overline{G}$ admet un stable de taille k .

Les problèmes d'optimisation associés à ces deux problèmes consistent à rechercher la taille d'une

plus grande clique (respectivement d'un plus grand stable). Un problème d'énumération consiste à lister les cliques maximales (respectivement les stables maximaux) qu'un graphe possède.

Couverture de Sommets

Le problème **COUVERTURE DE SOMMETS** est défini de la manière suivante :

Problème **COUVERTURE DE SOMMETS**

entrée. Un graphe $G = (V, E)$; un entier k .

question. Existe-t-il un sous-ensemble de sommets V' tel que toute arête de E a au moins une extrémité dans V' et tel que $|V'| \leq k$?

La version d'optimisation du problème consiste à calculer la plus petite couverture de sommets que possède un graphe. Un problème d'énumération associé consiste à énumérer toutes les couvertures de sommets minimales qu'un graphe possède. On notera que si un ensemble V' est une couverture de sommets, alors l'ensemble $V \setminus V'$ est un stable. Par conséquent, G admet une couverture de sommets de taille k si et seulement si G admet un stable de taille $n - k$.

Ensemble Coupe-Cycle

Le problème **ENSEMBLE COUPE-CYCLE** est défini de la manière suivante :

Problème **ENSEMBLE COUPE-CYCLE**

entrée. Un graphe $G = (V, E)$; un entier k .

question. Existe-t-il un ensemble de sommets V' de taille au plus k tel que le graphe induit par $G[V \setminus V']$ soit une forêt ?

La version optimisation consiste à trouver le plus petit ensemble coupe-cycle qu'un graphe peut avoir. La version d'énumération consiste à énumérer tous les ensembles coupe-cycles minimaux qu'un graphe possède.

Recherche d'ensemble connexe tropical minimum

Sommaire

2.1	Introduction	30
2.1.1	Détection de motifs dans les graphes colorés	30
2.1.2	État de l'art	31
2.1.3	Résultats du chapitre	32
2.1.4	Preliminaires	33
2.2	Un algorithme exact pour les graphes quelconques	34
2.2.1	Un algorithme par recherche exhaustive	35
2.2.2	Un algorithme via Arbre de Steiner	35
2.2.3	Un algorithme via Ensemble Dominant Connexe Bleu et Rouge	39
2.2.4	Combinaison des trois approches	42
2.3	Non-existence d'un algorithme sous-exponentiel pour les arbres	44
2.4	Un algorithme exact pour les arbres	45
2.4.1	Instances et sous-problèmes	46
2.4.2	Description de l'algorithme	47
2.5	Conclusion	58

2.1 Introduction

Ce chapitre étudie la recherche de structures particulières dans les graphes colorés, les ensembles connexes tropicaux de cardinalité minimum. Étant donné un graphe et une coloration de ses sommets (qui n'est pas nécessairement une coloration propre), un ensemble connexe tropical est un sous-ensemble de sommets contenant toutes les couleurs du graphe, tel que le graphe induit par ce sous-ensemble soit connexe.

Nous donnerons une preuve de la non-existence d'un algorithme sous-exponentiel en n pour ce problème, y compris pour les arbres sous l'*Hypothèse de Temps Exponentiel*, un algorithme exact calculant un ensemble connexe tropical minimum pour les graphes colorés quelconques et un algorithme exact plus rapide calculant un ensemble connexe tropical minimum pour les arbres colorés.

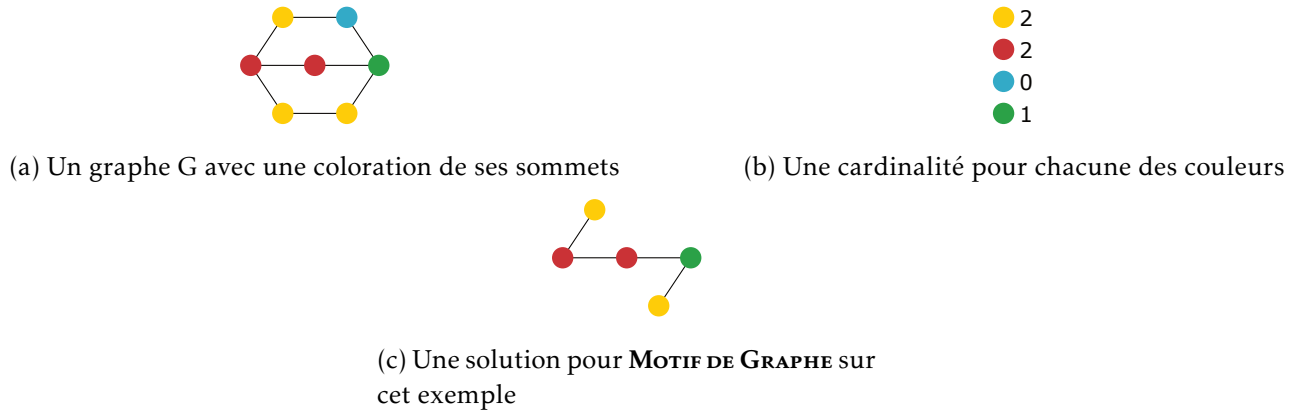
Les résultats de ce chapitre ont été établis en collaboration avec *Mathieu Chapelle, Manfred Cochefert, Dieter Kratsch* et *Mathieu Liedloff* et ont fait l'objet d'un article présenté à la conférence *IPEC 2014* [CCK⁺14].

2.1.1 Détection de motifs dans les graphes colorés

Les problèmes portant sur les graphes colorés ont été largement étudiés. Citons notamment le problème **MOTIF DE GRAPHE** qui fut introduit en 1994 par *Mc Morris et al.* [MWW94]. Ce problème a des applications en biologie et en réseaux métaboliques [FLS06, IKSS06]. Dans **MOTIF DE GRAPHE**, étant donné un graphe coloré $G = (V, E)$ on cherche à savoir s'il existe un sous-ensemble de sommets $S \subseteq V$ connexe tel qu'il y ait une bijection entre S et un multi-ensemble de couleurs faisant partie de l'entrée. De manière analogue, la question est de savoir si un graphe dont les sommets sont colorés et auquel est associé un vecteur de multiplicités sur les couleurs de ses sommets admet un ensemble de sommets connexe S tel que chaque couleur apparaisse dans S avec la multiplicité requise. Notons que la taille de la solution est donnée par l'entrée (elle est égale à la somme des multiplicités).

Nous donnons à la figure 2.1 une solution à ce problème pour un graphe coloré et un ensemble de multiplicités en entrée.

Parmi les variantes étudiées de ce problème, citons le problème **MOTIF MAXIMUM** pour lequel, étant donné un graphe G coloré et un motif $\mathcal{M}$ (c'est-à-dire un multi-ensemble de couleurs) on recherche un sous-ensemble de sommets S tel que $G[S]$ est connexe, l'ensemble des couleurs de S est inclus dans $\mathcal{M}$ et S est de cardinalité maximum [DFV11]. D'autres variantes de **MOTIF DE**

FIGURE 2.1 – **MOTIF DE GRAPHE** sur un exemple

GRAPHE ont été étudiées dans [BvBF⁺11, FFHV07, FFHV11, GS10, FLS06].

Nous nous intéressons dans ce chapitre à une autre variante de **MOTIF DE GRAPHE** définie par *Angles d'Auriac et al.*, le problème **ENSEMBLE CONNEXE TROPICAL MINIMUM** pour lequel, étant donné un graphe coloré, on recherche un sous-ensemble de sommets de cardinalité minimum contenant chacune des couleurs du graphe d'origine, et tel que le graphe induit par ce sous-ensemble est connexe [AdCH⁺14]. Plus formellement, le problème **ENSEMBLE CONNEXE TROPICAL MINIMUM** est le suivant :

Problème ENSEMBLE CONNEXE TROPICAL MINIMUM

- entrée.** Un graphe $G = (V, E)$, un ensemble de couleurs $C \subseteq \mathbb{N}$ et une fonction de coloration $c : V \rightarrow C$.
- sortie.** Un sous-ensemble de taille minimum $S \subseteq V$ tel que $G[S]$ est connexe et S contient au moins un sommet de chaque couleur dans C .

On donne à la figure 2.2 une solution de ce problème étant donné un graphe en entrée et une coloration des sommets de ce graphe.

2.1.2 État de l'art

Fellows et al. ont montré que **MOTIF DE GRAPHE** est *NP*-complet, même si le multi-ensemble de couleurs est un ensemble, et si le graphe est un arbre de degré maximum 3. Ils ont aussi montré que ce problème est *NP*-complet même si le multi-ensemble contient uniquement deux couleurs

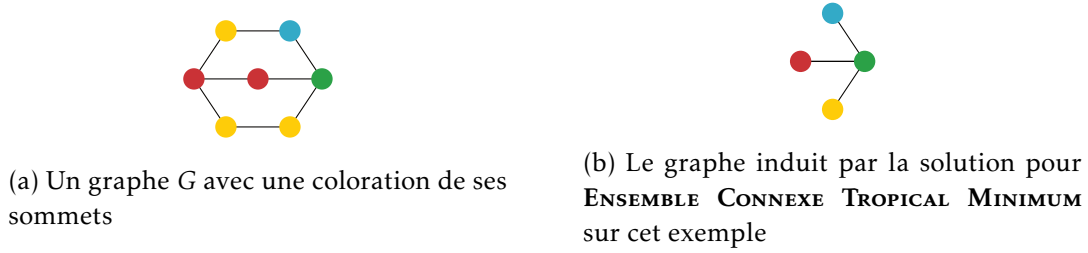


FIGURE 2.2 – ENSEMBLE CONNEXE TROPICAL MINIMUM sur un exemple

et si le graphe est biparti de degré maximum 4 [FFHV07].

Parmi les variantes étudiées de **MOTIF DE GRAPHE**, à notre connaissance la plupart des résultats sont des preuves de *NP*-difficultés et des algorithmes *FPT*¹ [BvBF⁺11, FFHV07, FFHV11, GS10, FLS06], à l'exception du problème **MOTIF MAXIMUM** pour lequel Dondi et al. donnent dans [DFV11] un algorithme exact exponentiel. Ils donnent un algorithme résolvant **MOTIF MAXIMUM** en temps $\mathcal{O}^*(1.62^n)$ lorsque le graphe en entrée est un arbre, et réduisent cette complexité en temps à $\mathcal{O}^*(1.33^n)$ si le motif est un ensemble de couleurs.

Angles d'Auriac et al. montrent dans [AdCH⁺14] que le problème de décision associé à **ENSEMBLE CONNEXE TROPICAL MINIMUM** est *NP*-complet, même lorsque l'entrée est restreinte aux arbres de hauteur au plus 3. Cette réduction est obtenue à partir du problème **ENSEMBLE DOMINANT**. Ils ont aussi montré que ce problème reste *NP*-complet lorsque l'entrée est un graphe d'intervalles, via une réduction à partir de **COUVERTURE DE SOMMETS**. Via une réduction à partir de **COUVERTURE DE SOMMETS** sur les graphes quelconques ils ont montré qu' **ENSEMBLE CONNEXE TROPICAL** est *NP*-complet lorsque le graphe en entrée est un graphe split. Ils donnent ensuite un algorithme résolvant **ENSEMBLE CONNEXE TROPICAL** en temps $\mathcal{O}(n^2 \times m \times 8^{|C|})$ pour les graphes à n sommets, m arêtes et colorés par $|C|$ couleurs différentes. Finalement, ils donnent des conditions suffisantes pour qu'un graphe coloré possède des *ensembles connexes tropicaux* dont chaque couleur n'apparaisse qu'une seule fois, et étudient cette question lorsque le graphe en entrée est généré aléatoirement (sur le modèle $G_{n,p}$ d'Erdős-Rényi).

2.1.3 Résultats du chapitre

Nous verrons tout d'abord en section 2.2 le premier résultat de ce chapitre : nous donnerons un algorithme exact exponentiel par rapport au nombre de sommets du graphe en entrée calculant un ensemble connexe tropical de cardinalité minimum en temps $\mathcal{O}^*(1.5359^n)$ et en espace polynomial. Cet algorithme est une combinaison de trois techniques : une approche en force

1. Étant donné un problème muni d'un paramètre k , un algorithme pour ce problème est *FPT* (pour Fixed Parameter Tractable) s'il est de complexité $f(k) \cdot n^{\mathcal{O}(1)}$, f étant une fonction calculable et n étant la taille de l'entrée.

brute, une approche basée sur une réduction vers **ARBRE DE STEINER** dans laquelle nous utilisons l'algorithme *FPT* de Nederlof [Ned12] et une approche basée sur une réduction vers **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE** dans laquelle nous utilisons l'algorithme de Abu-Khzam et al. [AKLM11] pour résoudre ce problème.

Nous verrons ensuite en section 2.3 qu'il n'existe pas d'algorithme sous-exponentiel pour résoudre **ENSEMBLE CONNEXE TROPICAL** sur les arbres sous l'*Hypothèse de Temps Exponentiel*; cette démonstration étant basée sur une réduction d'**ENSEMBLE CONNEXE TROPICAL** à partir d'**ENSEMBLE DOMINANT**. De nombreux problèmes *NP*-difficiles peuvent être résolus en temps polynomial lorsque l'entrée est restreinte aux arbres; ils admettent souvent un algorithme *FPT* paramétrés par la largeur arborescente [Cou90]. Des problèmes *NP*-difficiles dont l'entrée est restreinte aux graphes planaires ou de genre borné sont souvent résolubles par des algorithmes sous-exponentiels paramétrés [DH08, FT04]. Citons par exemple que le problème **ENSEMBLE DOMINANT**, dont le meilleur algorithme connu résout ce problème en temps $\mathcal{O}(1.4689^n)$ [Iwa11] se résout en temps $\mathcal{O}(2^{5.043\sqrt{n}}n + n^4)$ [FT04] lorsque le graphe en entrée est un graphe planaire. Plus généralement, Lipton et Tarjan ont montré dans [LT80] que de nombreux problèmes sur les graphes planaires peuvent être résolus en temps sous-exponentiel. Ainsi, si un problème ne peut pas être résolu en temps sous-exponentiel sur des arbres, quel type d'algorithme exponentiel *rapide* peut-on espérer obtenir? C'est notre principale motivation pour la conception d'algorithmes exponentiels pour ce problème.

Nous verrons en section 2.4 le second résultat de ce chapitre, concernant un algorithme exact exponentiel en la taille de l'entrée calculant un ensemble connexe tropical de cardinalité minimum en temps $\mathcal{O}^*(1.2721^n)$ et en espace polynomial lorsque le graphe en entrée est un arbre. Cet algorithme est un algorithme de branchement utilisant les propriétés structurelles des arbres.

2.1.4 Préliminaires

Les graphes considérés dans ce chapitre sont non-orientés et simples. Soit $G = (V, E)$ un graphe, $C \subseteq \mathbb{N}$ un ensemble de couleurs et $c : V \rightarrow C$ une coloration de G (qui n'est pas nécessairement une coloration propre). On désigne par (G, c) un graphe dont les sommets sont colorés, et $C = \{c(v) \mid v \in V\}$ est l'ensemble des couleurs de (G, c) . Étant donné un sous-ensemble de sommets S d'un graphe coloré (G, c) , $c(S) = \{c(v) \mid v \in S\}$ est l'ensemble de couleurs de S . Les graphes de ce chapitre seront associés à une coloration de leurs sommets; on notera souvent G un tel graphe au lieu de (G, c) .

Définition 2.1. *Ensemble connexe tropical*

S est un ensemble connexe tropical pour G si et seulement si $c(S) = C$.

La fonction $l_1(G)$ désigne le nombre de couleurs apparaissant une seule fois dans un graphe coloré (G, c) , et $l_2(G)$ le nombre de couleurs apparaissant au moins deux fois dans (G, c) . On a donc $l_1(G) + l_2(G) = |C|$.

Une composante connexe P d'un graphe non connexe G admet un ensemble connexe tropical si et seulement si toutes les couleurs de G apparaissent dans P . Un graphe non connexe dont aucune des composantes connexes ne contient toutes ses couleurs n'admet donc pas d'ensemble connexe tropical. Désormais, on ne considère que les graphes connexes.

Soit $T = (V, E)$ un arbre et (T, c) une instance d'ENSEMBLE CONNEXE TROPICAL MINIMUM. Dans le reste de ce chapitre T désigne un arbre enraciné en un sommet particulier $r \in V$. Pour un sommet donné v , $T(v)$ désigne le sous-arbre de T enraciné en v , et $|T(v)|$ son nombre de sommets. On note par $\text{chem}(v, r) = (x_0, x_1, \dots, x_k)$ l'ensemble des sommets du chemin dans l'arbre T du sommet $v = x_0$ vers la racine $r = x_k$ de T . Par $d(v, w)$ on désigne la longueur du chemin de v vers w dans T .

Soit $S \subseteq V$ un sous-ensemble connexe de T et $v \in V$. On note $d(v, S)$ la distance de v vers S dans T , définie par $d(v, S) = \min_{s \in S} d(v, s)$. Autrement dit, $d(v, S)$ désigne la distance minimum entre v et un sommet de S . Notons que $d(v, S) = 0$ pour tout $v \in S$, et que $d(v, S) = 1$ pour tout $v \in N(S)$.

Un sommet $v \in V$ est appelé *feuille* de l'arbre enraciné T si $|T(v)| = 1$, autrement v est appelé *nœud interne* de T . Finalement, étant donné un sommet $v \in V \setminus \{r\}$, on désigne par $p(v)$ le père de v dans l'arbre enraciné T , et étant donné un nœud interne w , on désigne par $s_1(w), \dots, s_k(w)$, $k \geq 1$ les fils de w dans T .

Le lecteur pourra se référer à la section 1.1 du chapitre 1 pour les définitions et notations complémentaires aux termes que nous venons de définir, et concernant les graphes en général. Le lecteur pourra également se référer à la section 1.2 de ce même chapitre ainsi qu'au livre « Exact Exponential Algorithms » de Fomin et Kratsch (voir chapitre 2 dans [FK10]) pour plus de précisions concernant les techniques algorithmiques utilisées ainsi que sur leur analyse.

2.2 Un algorithme exact pour les graphes quelconques

Cette section est dédiée à la conception et à l'analyse d'un algorithme exact pour ENSEMBLE CONNEXE TROPICAL. Un algorithme par recherche exhaustive résout ce problème en temps $\mathcal{O}^*(2^n)$, étant donné un graphe à n sommets. On vérifie pour chaque sous-ensemble de sommets (il y en a 2^n) s'il est connexe et tropical (s'il contient toutes les couleurs de G) en temps polynomial. Enfin, on retourne un sous-ensemble valide qui est de cardinalité minimum.

En utilisant un algorithme en force brute, un algorithme basé sur une réduction vers le problème ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE et un algorithme basé sur une réduction vers le

problème **ARBRE DE STEINER** et en combinant ces algorithmes, on conçoit un algorithme calculant un ensemble connexe tropical minimum en temps $\mathcal{O}^*(1.5359^n)$.

2.2.1 Un algorithme par recherche exhaustive

Soit (G, c) une instance d'**ENSEMBLE CONNEXE TROPICAL**.

Soient $V_{=1}$ les sommets de V dont la couleur n'apparaît qu'une seule fois dans G , et $V_{\geq 2}$ les sommets dont la couleur apparaît au moins deux fois dans G . Notons que $V = V_{=1} \cup V_{\geq 2}$. L'algorithme commence par calculer l'ensemble $V_{=1}$, forcés à être dans n'importe quel ensemble tropical, puis il énumère tous les ensembles $W \subseteq V_{\geq 2}$. Pour chaque $W \subseteq V_{\geq 2}$ il vérifie si $W \cup V_{=1}$ contient chacune des couleurs de G , et si $W \cup V_{=1}$ est connexe, chacune de ces vérifications se faisant en temps polynomial. Il mémorise alors W s'il est valide et s'il est de cardinalité minimum parmi les ensembles valides précédemment calculés. À la fin de son exécution, il retourne cet ensemble minimum.

En notant $l_1(G)$ le nombre de couleurs apparaissant exactement une fois dans G il y a $n - l_1(G)$ sommets dont la couleur apparaît au moins deux fois. Notre algorithme va énumérer chaque sous-ensemble de ces sommets (il y en a $2^{n-l_1(G)}$) et vérifier pour chacun d'entre eux en temps polynomial s'il est connexe et tropical. On en déduit le lemme suivant :

Lemme 2.2.

*L'algorithme en force brute résout **ENSEMBLE CONNEXE TROPICAL** en temps $\mathcal{O}^*(2^{n-l_1(G)})$, étant donné un graphe G à n sommets, $l_1(G)$ étant le nombre de couleurs apparaissant exactement une fois dans G .*

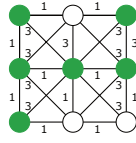
2.2.2 Un algorithme via Arbre de Steiner

Ce deuxième algorithme est basé sur une réduction vers le problème **ARBRE DE STEINER**. Le problème **ARBRE DE STEINER** est le suivant :

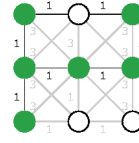
Problème **ARBRE DE STEINER**

- | |
|---|
| <div style="border-left: 1px solid black; padding-left: 10px;"> <p>entrée. Un graphe $G = (V, E)$, une fonction de pondération $w : E \rightarrow \mathbb{N}$, un ensemble de terminaux $K \subseteq V$.</p> <p>sortie. Un sous-arbre connexe $T = (V', E')$ de G, avec $V' \subseteq V$ et $E' \subseteq E$, tel que $K \subseteq V'$ et tel que $\sum_{e \in E'} w(e)$ est minimum.</p> </div> |
|---|

Le principe de cet algorithme est de construire à partir d'une instance $\mathcal{I}$ en entrée d'**ENSEMBLE CONNEXE TROPICAL** une instance $\mathcal{I}'$ telle que l'on puisse déduire d'une solution d'**ARBRE DE STEINER** sur $\mathcal{I}'$ une solution d'**ENSEMBLE CONNEXE TROPICAL** sur $\mathcal{I}$. Cette construction se fait en temps polynomial.



(a) Un graphe G avec un ensemble de terminaux (les sommets verts), et une pondération des arêtes.



(b) Un arbre de poids minimum couvrant tous les terminaux.

FIGURE 2.3 – **ARBRE DE STEINER** sur un exemple

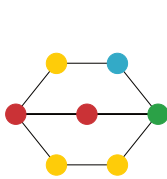
Ce problème est proche du problème **ARBRE COUVRANT MINIMUM**, sauf qu'on ne cherche à couvrir qu'un sous-ensemble de sommets $K \subseteq V$.

On donne un exemple d'**ARBRE DE STEINER** à la figure 2.3.

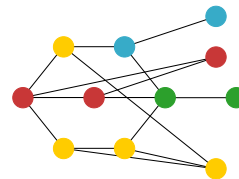
ARBRE DE STEINER est un problème connu pour être *NP*-complet appartenant à la liste des 21 problèmes *NP*-complets de Karp [Kar72].

Nederlof donne dans [Ned12] un algorithme *FPT* résolvant **ARBRE DE STEINER** en temps $\mathcal{O}(W \cdot 2^{|K|})$, W étant le poids maximum parmi les arêtes du graphe en entrée, et K étant l'ensemble des terminaux. Notons que Fomin et al. donnent dans [FGK⁺13] un algorithme en $\mathcal{O}(1.59^n)$ et en espace polynomial, ainsi qu'un algorithme en $\mathcal{O}(1.36^n)$ nécessitant un espace exponentiel.

Description de la réduction



(a) Un graphe coloré (G, c) .



(b) le graphe G' obtenu à partir de la première réduction.

FIGURE 2.4 – Illustration de la réduction décrite dans la section 2.2.2.

On définit maintenant une construction qui sera utilisée par nos algorithmes pour réduire **ENSEMBLE CONNEXE TROPICAL** à **ARBRE DE STEINER** (nous utiliserons cette même réduction pour

construire la réduction d'ENSEMBLE CONNEXE TROPICAL vers ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE dans la section 2.2.3).

Soit $G = (V, E)$ un graphe, c une coloration de G et C l'ensemble de couleurs de G . On définit tout d'abord le graphe $G' = (R' \cup B', E')$ par :

- $R' = V$;
- $B' = \{x_i \mid i \in C\}$;
- $E' = E \cup \{(v, x_i) \mid v \text{ est coloré en } i \text{ dans } G\}$.

Observons que cette construction peut être faite en temps polynomial.

Étant donné un graphe G à n sommets dont les sommets sont colorés par $|C|$ couleurs, le graphe G' obtenu par l'application de cette réduction est un graphe à $n + |C|$ sommets ; en effet, on ajoute un sommet dans G' pour chaque sommet de G , ainsi qu'un sommet pour chaque couleur de G .

Le problème ENSEMBLE CONNEXE TROPICAL se réduit à ARBRE DE STEINER de la manière suivante. Soit (G, c) une instance d'ENSEMBLE CONNEXE TROPICAL. On construit une instance (G', w, K) pour ARBRE DE STEINER, où $G' = (R' \cup B', E')$ est le graphe construit via la construction décrite précédemment et l'ensemble de terminaux $K = B'$. Notons que $|K| = |B'| = |C|$.

La fonction de pondération des arêtes est définie de la manière suivante :

- $w(e) = n = |V|$ pour tout $e \in E(G')$ incidente à un sommet de B' ;
- $w(e) = 1$ pour toute arête $e \in E$.

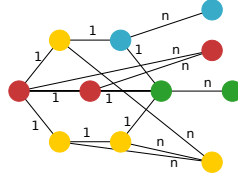


FIGURE 2.5 – La pondération du graphe G' construit dans la figure 2.4 pour ARBRE DE STEINER.

Lemme 2.3. [Réduction d'ENSEMBLE CONNEXE TROPICAL à ARBRE DE STEINER]

Le graphe dont les sommets sont colorés (G, c) admet un ensemble connexe tropical de taille k si et seulement si (G', w, B') admet un arbre de Steiner de poids $k - 1 + |C| \cdot n$.

Démonstration.



(a) Un graphe G et un ensemble connexe tropical de taille minimum (les sommets encadrés).

(b) Le graphe G' et l'ensemble de sommets correspondant à l'ensemble connexe tropical de G (les sommets encadrés).

FIGURE 2.6 – Illustration de la réduction d'ENSEMBLE CONNEXE TROPICAL à ARBRE DE STEINER. Les sommets encadrés dans G' sont les images des sommets encadrés de G ; l'ensemble des sommets encadrés dans G est un ensemble connexe tropical minimum si et seulement si l'ensemble des sommets encadrés dans G' est un arbre de Steiner de poids minimum.

Commençons tout d'abord par montrer que si (G, c) admet un ensemble connexe tropical de taille k , alors (G', w, B') admet un arbre de Steiner de poids $k - 1 + |C| \cdot n$.

Soit S un ensemble connexe tropical de (G, c) tel que $|S| = k$. L'ensemble des arêtes d'un arbre de Steiner $\tilde{T}$ pour (G', w, B') peut donc être obtenu en prenant tout d'abord toutes les $k - 1$ arêtes d'un arbre couvrant de $G[S] = G'[S]$; ces arêtes ont toutes pour poids 1. Alors pour tout terminal $x_i \in B'$ avec i une couleur de C , on choisit une arête $(v', x_i) \in E'$, v étant coloré par i dans G . Une telle arête existe pour tout $x_i \in B'$ étant donné que S est tropical dans G , et ces arêtes ont toutes pour poids n . L'arbre de Steiner $\tilde{T}$ a donc pour poids $k - 1 + |C| \cdot n = k'$.

Montrons maintenant que si (G', w, B') admet un arbre de Steiner de poids k' alors (G, c) admet un ensemble connexe tropical de taille k .

Soit $\tilde{E}$ l'ensemble des arêtes de l'arbre de Steiner $\tilde{T}$ de (G', w, B') ayant un poids de k' . Par construction, B' est un stable de G' , et donc $\tilde{E}$ contient pour chaque sommet de $x_i \in B'$ une arête incidente à x_i . Il y a au moins $|B'| = |C|$ arêtes de poids n ; étant donné la valeur de k' , il y a précisément $|C|$ arêtes de poids n dans $\tilde{E}$. Soit S l'ensemble des sommets de R' incidents aux arêtes de poids n dans $\tilde{E}$. Par construction de G' , S est tropical dans G . L'arbre de Steiner $\tilde{T}$ contient $k - 1$ arêtes dans $G'[R'] = G$ connectant les sommets de S . Par conséquent l'ensemble des sommets dans l'arbre de Steiner à l'intérieur de R' est connexe, contient S , et est par conséquent tropical. Étant donné que l'arbre de Steiner dans $G'[R']$ a $k - 1$ arêtes, l'ensemble connexe tropical contient k sommets. $\square$

En utilisant le résultat du lemme 2.3 ainsi que l'algorithme de Nederlof [Ned12] résolvant ARBRE DE STEINER en temps $O(W \cdot 2^k)$ (k étant le nombre de terminaux du graphe et W étant le poids maximum parmi les poids des arêtes du graphe), on en déduit le lemme suivant :

Lemme 2.4.

L'algorithme utilisant **ARBRE DE STEINER** résout **ENSEMBLE CONNEXE TROPICAL** en temps $\mathcal{O}^*(2^{|C|})$, où $|C|$ désigne le nombre de couleurs dans G .

2.2.3 Un algorithme via Ensemble Dominant Connexe Bleu et Rouge

Le troisième algorithme est basé sur une réduction vers le problème **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE**. Le principe de cet algorithme est similaire à l'algorithme précédent : à partir d'une instance pour **ENSEMBLE CONNEXE TROPICAL** on construit une instance pour **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE**, et on déduit d'une solution pour le second problème une solution pour le problème initial.

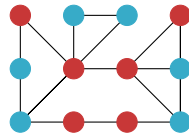
Le problème **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE** est le suivant :

Problème ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE

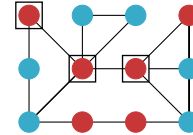
entrée. Un graphe $G = (R \cup B, E)$ dont les sommets sont colorés soit en rouge (les sommets de l'ensemble R), soit en bleu (les sommets de l'ensemble B).

question. Trouver le plus petit sous-ensemble $S \subseteq R$ de sommets rouges tel que $G[S]$ est connexe, et chaque sommet de B a au moins un voisin dans S , autrement dit $B \subseteq N(S)$.

L'entrée de ce problème est un graphe dont les sommets sont partitionnés en deux ensembles : l'ensemble des sommets B et l'ensemble des sommets R . On cherche alors le plus petit sous-ensemble connexe de sommets $S \subseteq R$ tel que tout sommet de B a au moins un voisin dans S . On donne dans la figure 2.7 un exemple de solution pour ce problème, étant donné un graphe en entrée.



(a) Un graphe dont les sommets sont partitionnés en deux ensembles B et R .



(b) Une solution pour ce graphe (les sommets encadrés).

FIGURE 2.7 – **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE** sur un exemple.

Ce problème a été montré comme étant *NP*-complet par Downey et al. [DFVW99]. Abu-Khzam et al. montrent que ce problème peut être résolu en temps $\mathcal{O}^*(1.3645^n)$ et en espace polynomial [AKLM11].

Description de la réduction

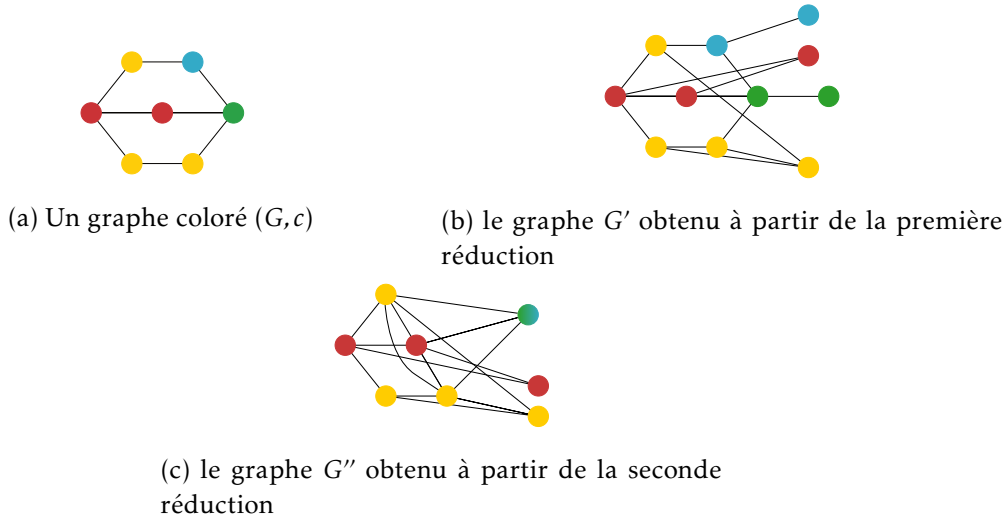


FIGURE 2.8 – Illustration de la réduction de G'' à partir de G' décrite dans la section 2.2.3

La réduction suivante utilise la réduction décrite précédemment : à partir d'une instance (G, c) pour **ENSEMBLE CONNEXE TROPICAL** on construit un nouveau graphe G' puis, comme pour **ARBRE DE STEINER**, à partir de ce graphe on construit un graphe $G'' = (R'' \cup B'', E'')$ en modifiant $G' = (R' \cup B', E')$:

- Initialement, on a $R'' = R'$, $B'' = B'$ et $E'' = E'$;
- Pour tout sommet $v \in V$ dont la couleur apparaît exactement une fois dans G , on déplace le sommet correspondant v' de R'' vers B'' , et on supprime le sommet x_i de B'' , où $c(v) = i$ dans G ;
- À l'issue de cette étape, soient $B_1, \dots, B_p$ les composantes du sous-graphe induit dans $G''[B'']$ par les sommets qui ont été déplacés vers B'' . Pour chaque $i = 1, 2, \dots, p$, contracter la composante B_i dans $G''[B'']$ de manière à ce qu'il ne reste plus qu'un sommet que nous appelons b_i ;

On notera qu'une fois toutes les contractions effectuées, B'' est maintenant un stable de G'' .

- Pour finir, pour chaque b_i , $1 \leq i \leq p$, on ajoute des arêtes entre les sommets voisins de b_i de manière à former une clique. Le graphe résultant est $G'' = (R'' \cup B'', E'')$ et est utilisé pour réduire **ENSEMBLE CONNEXE TROPICAL** vers **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE**.

Notons que cette construction peut être faite en temps polynomial. Exprimons maintenant le nombre de sommets d'un graphe G'' obtenu à partir d'un graphe G à n sommets en appliquant les deux réductions précédemment décrites. Rappelons que $l_1(G)$ désigne le nombre de couleurs n'apparaissant qu'une seule fois dans G et $l_2(G)$ le nombre de couleurs apparaissant au moins deux fois. Nous avons précédemment montré que le graphe obtenu à l'issue de la première réduction contient $n + |C|$ sommets, soit $n + l_1(G) + l_2(G)$ sommets. Dans la seconde réduction, on contracte chaque sommet correspondant à une couleur n'apparaissant qu'une seule fois dans G en un seul sommet. Le graphe G'' ainsi obtenu contient au plus $n + l_2(G) + 1$ sommets.

ENSEMBLE CONNEXE TROPICAL peut être réduit à **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE** de la manière suivante. Soit (G, c) une instance d'**ENSEMBLE CONNEXE TROPICAL**. On construit une instance $G'' = (R'' \cup B'', E'')$ d'**ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE** où G'' est le graphe via la construction décrite précédemment, R'' est l'ensemble des sommets rouges et B'' est l'ensemble des sommets bleus.

Lemme 2.5.

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

Démonstration.

Soit S un ensemble connexe tropical de G de taille k . Soit U l'ensemble (qui peut être vide) de sommets ayant une couleur apparaissant une seule fois dans G . Clairement $U \subseteq S$. On affirme que $D = S \setminus U$ est un ensemble dominant connexe bleu et rouge de G'' . Par construction, $B_1, B_2, \dots, B_p$ sont les composantes connexes de $G[U]$, et pour chaque $i \in \{1, 2, \dots, p\}$, l'ensemble connexe B_i de G est contracté en un sommet b_i dans G'' . L'ensemble S étant connexe dans G , l'ensemble $(S \setminus U) \cup \{b_1, b_2, \dots, b_p\}$ est connexe dans G'' vu que l'on contracte les ensembles connexes dans G . Lorsque l'on supprime $\{b_1, b_2, \dots, b_p\}$ de $(S \setminus U) \cup \{b_1, b_2, \dots, b_p\}$ on obtient D .

On considère maintenant les sous-graphes induits dans G'' . Étant donné que par construction de G'' pour tout b_i le voisinage de b_i forme une clique dans G'' , la suppression d'un sommet b_i ne peut pas déconnecter l'ensemble D . Donc D est connexe dans G'' . Finalement, par construction, $D \subseteq R''$ et $N(D) \cap B''$ contient tous les sommets $x_i \in B''$ avec une couleur i apparaissant au moins deux fois dans G . Vu que S est connexe dans G et $\{b_1, b_2, \dots, b_p\} \subseteq B''$ est un stable de G'' , tout sommet de $\{b_1, b_2, \dots, b_p\}$ a un voisin dans D . Ainsi D est un ensemble dominant bleu et rouge dans G'' . Pour résumer, D est un ensemble dominant connexe bleu et rouge de G'' tel que

$$|D| = |S| - |U| = k - l_1(G) = k'.$$

Inversement, soit $D \subseteq R''$ un ensemble dominant bleu et rouge de G'' tel que $|D| = k' = k - l_1(G)$. Par construction de G'' et étant donné que D est un ensemble dominant bleu et rouge, pour tout $i = 1, 2, \dots, p$, l'ensemble D contient au moins un sommet de $N(b_i) \cap R''$ dans G'' . Par conséquent $D \cup \bigcup_{i=1}^p B_i$ est un ensemble connexe dans G' , et également dans G . Ceci est dû au fait que pour tout i , le sommet b_i est obtenu en contractant l'ensemble connexe B_i . Cependant $D \cup \bigcup_{i=1}^p B_i$ est tropical car il contient tous les sommets de G avec une couleur apparaissant une seule fois dans G , et D domine $B'' \setminus \bigcup_{i=1}^p B_i$, c'est-à-dire D contient un sommet de chaque couleurs apparaissant au moins deux fois. Par conséquent $D \cup \bigcup_{i=1}^p B_i$ est un ensemble connexe tropical dans G de taille $k' + l_1(G) = k$. $\square$

Rappelons que le graphe G'' obtenu à partir de G par l'application successive des deux réductions possède $n + l_2(G) + 1$ sommets. En utilisant le résultat du lemme 2.5 ainsi que l'algorithme de Abu-Khzam et al. [AKLM11] résolvant **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE** en temps $\mathcal{O}(1.36443^n)$, on obtient le lemme suivant :

Lemme 2.6.

*L'algorithme utilisant **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE** résout **ENSEMBLE CONNEXE TROPICAL** en temps $\mathcal{O}(1.36443^{n+l_2(G)})$ (où $l_2(G)$ est le nombre de couleurs apparaissant au moins deux fois dans G).*

2.2.4 Combinaison des trois approches

Nous sommes maintenant prêts à décrire notre algorithme exact pour **ENSEMBLE CONNEXE TROPICAL MINIMUM** sur les graphes quelconques, en combinant les algorithmes obtenu via la réduction à **ARBRE DE STEINER**, via la réduction à **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE**, et via l'approche en force brute, selon la valeur de $l_1(G)$ (voir la figure 2.9). Soit (G, c) une instance d'**ENSEMBLE CONNEXE TROPICAL MINIMUM**.

- Si $l_1(G) < 0.23814 \cdot n$:
On réduit vers **ARBRE DE STEINER** et on utilise l'algorithme de Nederlof [Ned12] sur (G', w, K) ; on obtient un algorithme résolvant **ENSEMBLE CONNEXE TROPICAL MINIMUM** en temps $\mathcal{O}(2^{|C|})$.
Dans ce cas, $|C| = l_1(G) + l_2(G) < 0.23814 \cdot n + \frac{1-0.23814}{2} \cdot n = 0.61907 \cdot n$.

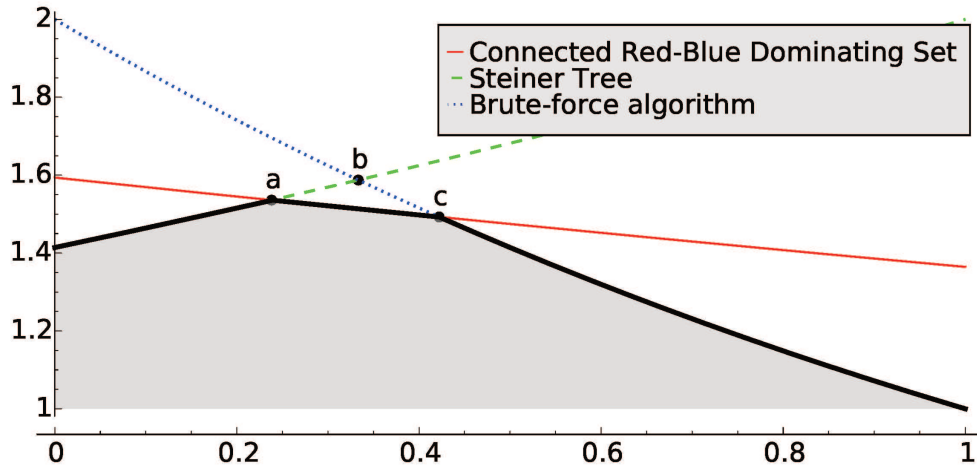


FIGURE 2.9 – Courbes représentant le temps d'exécution de chacune des trois techniques selon le taux de sommets dont la couleur n'apparaît qu'une fois dans le graphe

- Si $0.23814 \cdot n \leq l_1(G) \leq 0.42218 \cdot n$:
On réduit vers **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE** et on utilise l'algorithme de *Abu-Khizam et al.* [AKLM11] sur G'' ; on obtient un algorithme résolvant **ENSEMBLE CONNEXE TROPICAL MINIMUM** en temps $\mathcal{O}(1.36443^{n+l_2(G)})$.
Dans ce cas, $n + l_2(G) \leq n + \frac{1}{2} \cdot n - \frac{1}{2} \cdot l_1(G) \leq 1.38093 \cdot n$.
- Si $0.42218 \cdot n < l_1(G)$:
On utilise l'algorithme en force brute sur (G, c) ; on obtient un algorithme résolvant **ENSEMBLE CONNEXE TROPICAL MINIMUM** en temps $\mathcal{O}(2^{n-l_1(G)})$.
Dans ce cas, $n - l_1(G) \leq (1 - 0.42218) \cdot n = 0.57782 \cdot n$.

On établit un algorithme exponentiel n'ayant besoin que d'un espace polynomial. On en déduit le théorème suivant :

Théorème 2.7. [Algorithme exact pour **ENSEMBLE CONNEXE TROPICAL MINIMUM** sur des graphes quelconques]

ENSEMBLE CONNEXE TROPICAL MINIMUM peut être résolu en temps $\mathcal{O}(1.5359^n)$ et en espace polynomial pour les graphes à n sommets.

L'algorithme en force brute n'améliore pas la borne supérieure sur le temps d'exécution de l'algorithme. Notons cependant qu'il est meilleur que les deux autres approches lorsque $l_1(G) > 0.42218 \cdot n$.

2.3 Non-existence d'un algorithme sous-exponentiel pour les arbres

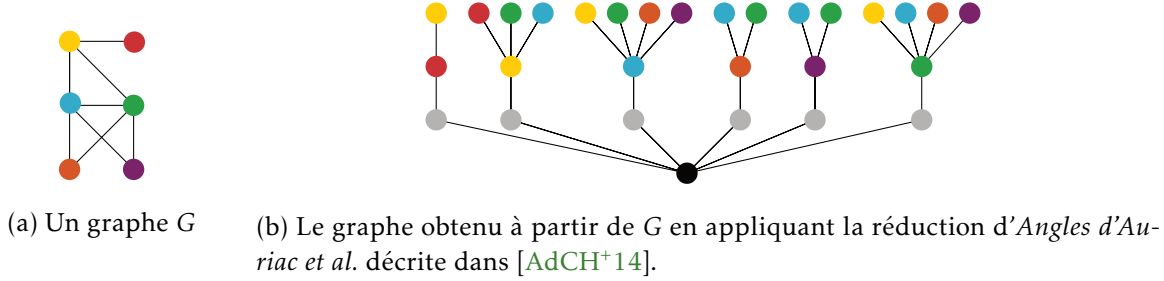


FIGURE 2.10 – La réduction d'ENSEMBLE DOMINANT vers ENSEMBLE CONNEXE TROPICAL sur un exemple

Dans cette section, on montre qu'ENSEMBLE CONNEXE TROPICAL n'admet pas d'algorithme sous-exponentiel même si le graphe en entrée est un arbre de hauteur au plus trois, à moins que l'Hypothèse de Temps Exponentiel (abrégée en *ETH*) ne soit fausse. Nous rappelons l'Hypothèse de Temps Exponentiel, énoncée par Impagliazzo et Paturi [IP01].

Hypothèse 2.8. [Hypothèse de Temps Exponentiel, Impagliazzo et Paturi [IP01]]

Pour tout $k \geq 3$, k -SAT n'admet pas d'algorithme sous-exponentiel en fonction du nombre de variables.

Il est considéré comme improbable que l'*ETH* soit fausse. Plus de détails concernant l'*ETH* sont donnés dans la sous-section 1.2.4 du chapitre 1.

La non-existence d'un algorithme sous-exponentiel pour ENSEMBLE CONNEXE TROPICAL sous l'*ETH* souligne l'importance de la conception d'un algorithme efficace s'exécutant en temps $\mathcal{O}(c^n)$ (pour tout graphe à n sommets et pour une constante c inférieure ou égale à 2). Le résultat est établi via une réduction d'ENSEMBLE DOMINANT vers ENSEMBLE CONNEXE TROPICAL décrite dans [AdCH⁺14], et la preuve qu'ENSEMBLE DOMINANT n'admet pas d'algorithme sous-exponentiel dans les graphes de degré maximum 6 donnée dans [FKW04].

Théorème 2.9.

Sous l'Hypothèse de Temps Exponentiel le problème ENSEMBLE CONNEXE TROPICAL n'admet pas d'algorithme en temps sous-exponentiel dans les arbres de hauteur au plus 3.

Démonstration.

Angles d'Auriac et al. montrent dans [AdCH⁺14] qu'ENSEMBLE CONNEXE TROPICAL est NP-complet même sur les arbres de hauteur au plus 3, par une réduction à partir d'ENSEMBLE DOMINANT. Par souci d'exhaustivité, on rappelle la construction utilisée dans leur preuve. Étant donné un graphe $G = (V, E)$ une instance en entrée d'ENSEMBLE DOMINANT, on construit un arbre coloré (T, c) de la manière suivante :

Considérons une coloration arbitraire $\sigma : V \rightarrow \mathbb{N}$ des sommets de G , telle que chaque sommet est coloré avec une couleur différente. Initialement, T contient un seul sommet r coloré en une couleur σ_r non-utilisée par la coloration de G . Puis, pour chaque sommet $v \in V(G)$, on ajoute à T une étoile dont le centre est coloré $\sigma(v)$, avec une feuille colorée en $\sigma(u)$ pour tout voisin u de v dans G , et une feuille supplémentaire colorée σ_0 , avec $\sigma_0 \neq \sigma(v_i) \forall v_i \in V$, reliée au sommet r de T . Nous illustrons cette réduction dans la figure 2.10. Ils montrent finalement que G contient un ensemble dominant de taille au plus k si et seulement si (T, c) contient un ensemble connexe tropical de taille au plus $k + n + 1$.

On se réfère maintenant à Fomin et al. [FKW04] qui ont prouvé qu'ENSEMBLE DOMINANT n'admet pas d'algorithme en temps sous-exponentiel pour des graphes de degré maximum 6 sous l'ETH.

Étant donnée une instance G du problème ENSEMBLE DOMINANT telle que G est de degré maximum 6, en appliquant la réduction que nous venons de décrire sur G nous obtenons un arbre coloré (T, c) qui est une instance du problème ENSEMBLE CONNEXE TROPICAL telle que T est un arbre de hauteur au plus 3 avec au plus $8n + 1 = \mathcal{O}(n)$ sommets. Par conséquent, d'un algorithme en temps sous-exponentiel pour ENSEMBLE CONNEXE TROPICAL sur les arbres de hauteur maximum 3 on obtient un algorithme en temps sous-exponentiel pour ENSEMBLE DOMINANT pour des graphes de degré maximum 6, ce qui contredit l'ETH. L'existence d'un tel algorithme est donc impossible à moins que l'ETH ne soit fausse. $\square$

Le fait qu'il n'y ait pas d'algorithme sous-exponentiel pour ENSEMBLE CONNEXE TROPICAL dans les arbres à moins que l'Hypothèse de Temps Exponentiel ne soit fausse justifie l'intérêt de construire des algorithmes modérément exponentiels pour ENSEMBLE CONNEXE TROPICAL aussi bien pour les arbres que pour les graphes en général.

2.4 Un algorithme exact pour les arbres

Le résultat principal de cette section est un algorithme calculant un ensemble connexe tropical minimum sur les arbres en temps $\mathcal{O}(1.2721^n)$. Cet algorithme utilise le paradigme *Brancher et Réduire*, dont le principe ainsi que les techniques d'analyse d'un tel algorithme sont présentés en section 1.2 du chapitre 1.

2.4.1 Instances et sous-problèmes

Comme mentionné précédemment, les algorithmes de branchement résolvent récursivement les sous-problèmes par l'application de règles de branchement et de réduction.

Étant donné un arbre (enraciné) $T = (V, E)$ avec une coloration c et un ensemble de couleurs C comme instance d'**ENSEMBLE CONNEXE TROPICAL MINIMUM**, T , c et C sont globaux pour notre algorithme récursif.

On définit maintenant une instance d'un sous-problème comme une 3-partition (S, F, D) de V . Étant donnée une instance (S, F, D) la tâche est de trouver un ensemble connexe tropical S^* de (T, c) tel que $S \subseteq S^*$, $S^* \cap D = \emptyset$ et la taille de S^* est minimum. On appelle un tel ensemble de sommet S^* une solution de (S, F, D) . Une instance (S, F, D) est finalement définie de la manière suivante :

- S est l'ensemble des sommets *sélectionnés* (*Selected vertices*), c'est-à-dire les sommets ayant déjà été choisis comme étant un sous-ensemble d'une solution ;
- F est l'ensemble des sommets *libres* (*Free vertices*), c'est-à-dire aucune décision n'a été prise quant au fait qu'ils appartiennent à S ou à D ;
- D est l'ensemble des sommets *rejetés* (*Discarded vertices*), c'est-à-dire les sommets ne pouvant appartenir à aucune solution en cours de construction.

La construction de notre algorithme implique qu'à chaque étape l'instance (S, F, D) satisfait les invariants suivants :

- la racine r de T appartient à S ;
On choisit initialement un sommet r de l'arbre à l'ensemble S , et on désigne r comme étant la racine de T . Étant donné qu'à chaque étape l'algorithme ne fait qu'ajouter des sommets à S , r appartient donc à S à chaque étape de l'algorithme.
- S et $S \cup F$ sont des ensembles connexes de T ;
L'ensemble que nous souhaitons obtenir à l'issue de l'algorithme est connexe. Les règles de branchement et de réduction garantissent que S reste connexe à chaque étape de l'algorithme. Si $S \cup F$ n'est pas connexe, les sommets de F qui ne sont pas dans la même composante connexe que r ne peuvent pas appartenir à l'ensemble connexe tropical en cours de construction ; ils appartiennent donc à D .
- pour tout $v \in D$, tout sommet de $T(v)$ appartient à D ;
 T étant un arbre, $v \in \text{chem}(v', r)$ pour chaque sommet $v' \in T(v)$. Si $v \in D$, v' ne peut appartenir à aucun ensemble connexe tropical de $V(T) \cap D$. Il appartient donc à D .

Finalement pour toute instance (S, F, D) , T' est le sous-arbre induit par $S \cup F$ et $C' \subseteq C$ est l'ensemble des couleurs qui n'appartiennent pas à S : $C' = C \setminus c(S)$.

L'ensemble que l'algorithme construit doit contenir toutes les couleurs du graphe d'origine. C' est donc l'ensemble des couleurs qui devront par la suite être ajoutés à la composante connexe afin

qu'elle contienne chaque couleur du graphe d'origine.

2.4.2 Description de l'algorithme

Les deux procédures suivantes sont utilisées dans les règles de branchement et de réduction de l'algorithme. Lorsqu'elles sont appliquées sur une instance (S, F, D) , elles permettent d'obtenir de nouvelles instances et de nouveaux sous-problèmes.

Soit un sommet libre $v \in F$. Notons que sélectionner v implique de déplacer les sommets libres de $chem(v, r)$ vers S , car sinon aucun super-ensemble de $S \cup \{v\}$ ne pourrait être connexe, et pour les mêmes raisons, rejeter v implique de rejeter tous les sommets de $T(v)$, et donc de les déplacer vers D .

ADD(v) on sélectionne tous les sommets libres de $chem(v, r)$: on déplace tout sommet de $chem(v, r)$ appartenant à F vers S et on supprime dans C' toutes les couleurs des sommets de $chem(v, r)$;

RMV(v) supprimer de F tous les sommets de $T(v)$, et les ajouter dans D , tandis que C' reste inchangé.

Soit $X \subseteq F$. Pour simplifier les notations, $ADD(X)$ désigne l'application de $ADD(v)$ sur tous les sommets $v \in X$. De même, $RMV(X)$ désigne l'application de $RMV(v)$ sur tous les sommets $v \in X$. Il n'y a pas d'importance concernant l'ordre des sommets selon lequel on applique la procédure.

Nous décrivons maintenant notre algorithme de branchement calculant un ensemble tropical connexe minimum sur les arbres.

Soit $T = (V, E)$ l'arbre en entrée, c la fonction de coloration de ses sommets, et C l'ensemble des couleurs de T . Pour chaque sommet $r \in V$, on considère l'arbre T enraciné en r et on applique l'algorithme de branchement sur l'instance $(S, F, D) = (\{r\}, V \setminus \{r\}, \emptyset)$. Notons que la racine appartient toujours à S . Un ensemble connexe tropical minimum de (T, c) est donc une solution de $(\{r\}, V \setminus \{r\}, \emptyset)$ de taille minimum.

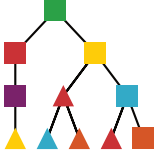
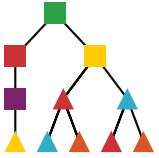
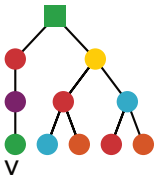
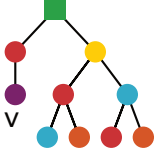
Soit (S, F, D) une instance quelconque. Cette instance correspondant à l'instance initiale (dans ce cas, $F = V \setminus \{r\}$), ou est obtenue à l'issue d'une suite d'appels récursifs. On donne ci-après la liste des règles de branchement et de réduction de notre algorithme. On associe de même à chaque règle de branchement son vecteur de branchement.

Nous associons à chacune des règles de branchement et de réduction une illustration. Les sommets de cette figure sont de trois formes différentes, selon l'ensemble dont ils font partie : les sommets « carrés » font partie de l'ensemble S des sommets *sélectionnés* ; les sommets « ronds »

font partie de l'ensemble F des sommets *libres* ; finalement, les sommets « triangles » font partie de l'ensemble D des sommets *rejetés*. Dans la plupart des figures, les sommets auront une couleur ; néanmoins, dans certaines figures les sommets sont noirs. Cela signifie que les couleurs des sommets n'ont aucune importance dans les branchements et réductions illustrés.

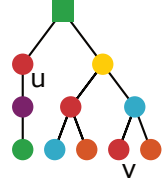
Règles de réduction de l'algorithme

Les règles sont listées dans l'ordre où elles sont appliquées, c'est-à-dire qu'une règle ne peut être appliquée à une instance que si les règles précédentes ne sont pas applicables. La correction de ces règles de réduction est donnée dans le lemme 2.10.

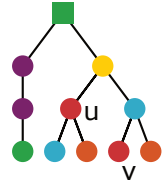
Règle de réduction 0.1. Si $C' = \emptyset$, alors STOP : S est un ensemble connexe tropical de T.	
Règle de réduction 0.2. Si $F = \emptyset$, alors STOP : il n'y a pas de solution pour cette instance.	
Règle de réduction 1. S'il existe $v \in F$, v est une feuille de T', tel que $c(v) \notin C'$, alors RMV(v).	
Règle de réduction 2. S'il existe $v \in F$ tel que $c(v) \in C'$ et pour tout $u \in F \setminus \{v\}$, $c(v) \neq c(u)$, alors ADD(v).	

Règle de réduction 3.

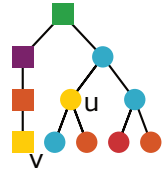
S'il existe $v \in F$, v est une feuille de T' , tel qu'il existe $u \in F \setminus \{v\}$ tel que $c(u) = c(v)$ et $d(u, S) = 1$, alors $\text{RMV}(v)$.

**Règle de réduction 4.**

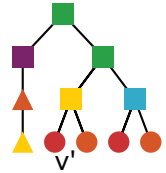
S'il existe $v \in F$, v est une feuille de T' , tel qu'il existe $u \in F \setminus \{v\}$ avec $c(u) = c(v)$ tel que $d(u, \text{chem}(v, r)) \leq 1$, alors $\text{RMV}(v)$.

**Règle de réduction 5.**

S'il existe $u \in F$, tel qu'il existe une feuille v de T' , $v \in S$, avec $c(u) = c(v)$, alors $\text{RMV}(u)$.

**Règle de réduction 6.**

Si tous les sommets de F sont des feuilles dans T' , alors $\text{ADD}(v)$, pour n'importe quel $v \in F$.



Nous montrons dans le lemme suivant la correction de ces règles de réduction :

Lemme 2.10.

Les règles de réduction de cet algorithme sont correctes.

Démonstration.

Soit (S, F, D) une instance d'un sous-problème et $C' = C \setminus c(S)$.

On désigne par S^* une solution de (S, F, D) . S^* si elle existe, est donc un ensemble connexe tropical satisfaisant $S \subseteq S^*$ et $D \cap S^* = \emptyset$ et est de taille minimum parmi tous les autres ensembles valides.

règle de réduction 0.1.

Soit $C' = \emptyset$. Étant donnée les propriétés sur les instances, $G[S]$ est connexe, et vu que $C' = \emptyset$, S est tropical. Par conséquent S est un ensemble connexe tropical de T et S est une solution

de (S, F, D) .

règle de réduction 0.2.

Soit $C' \neq \emptyset$ et $F = \emptyset$. Étant données les propriétés sur les instances, $G[S]$ est un ensemble connexe de T . Cependant, étant donné que $C' \neq \emptyset$, l'ensemble S n'est pas tropical. (S, F, D) n'a donc pas de solution, étant donné que S ne peut pas être étendu.

règle de réduction 1.

Soit $v \in F$ une feuille de $T' = T[S \cup F]$ telle que $c(v) \notin C'$. Soit $v' \in S$ avec $c(v) = c(v')$. Supposons que S^* est une solution de (S, F, D) . Supposons que $v \in S^*$. Étant donné que $S \subseteq S^*$ et $v' \in S$, nous avons aussi $v' \in S^*$. Clairement $S^* \setminus \{v\}$ est tropical et un ensemble connexe. Donc S^* n'est pas une solution de (S, F, D) , ce qui est une contradiction.

On peut donc sans problème rejeter v et appliquer $\text{RMV}(v)$.

règle de réduction 2.

Soit $v \in F$ l'unique sommet dans F de couleur $c(v) \in C'$. Vu que toute solution S^* de (S, F, D) doit contenir v pour être tropicale, on peut sans risque appliquer $\text{ADD}(v)$.

règle de réduction 3.

Soit $v \in F$ une feuille $T' = T[S \cup F]$, et soit $u \in F$ tel que $c(u) = c(v)$ et $d(u, S) = 1$. Supposons que $v \in S^*$ pour une solution S^* de (S, F, D) . Si S^* contient u , alors S^* n'est pas minimum, étant donné que $S^* \setminus \{v\}$ est tropical et connexe. Si S^* ne contient pas u , alors $(S^* \setminus \{v\}) \cup \{u\}$ est lui aussi connexe et tropical. Par conséquent, s'il y a une solution contenant v , il existe aussi une solution ne le contenant pas.

On peut par conséquent appliquer sans risque $\text{RMV}(v)$.

règle de réduction 4.

Soit $v \in F$ une feuille de T' et $u \in F$ avec $c(u) = c(v)$ et $d(u, \text{chem}(v, r)) \leq 1$. Soit S^* une solution contenant v . Si $u \in \text{chem}(v, r)$, alors $S^* \setminus \{v\}$ est tropical et connexe, et donc S^* n'est pas une solution, contradiction. Si $u \notin \text{chem}(v, r)$, alors il existe un $x \in \text{chem}(v, r)$, $x \neq v$, tel que $(x, u) \in E$. Cependant, étant donné que S^* est connexe, $v \in S^*$ implique $x \in S^*$. Donc $(S^* \setminus \{v\}) \cup \{u\}$ est tropical et connexe.

Il existe donc une solution ne contenant pas v et on peut donc appliquer sans risque $\text{RMV}(v)$.

règle de réduction 5.

Soit $u \in F$, v une feuille de T' , et $v \in S$ avec $c(u) = c(v)$. Soit S^* une solution de (S, F, D) contenant un sommet $x \in T'(u)$. S^* contenant x , il contient aussi u (car S^* est connexe). Étant donné que $c(u) = c(v)$, on remarque que $S^* \setminus \{v\}$ est aussi une solution, ce qui contredit le fait

que S^* est une solution minimale. Donc $u \notin S^*$ et on peut donc sans risque appliquer $\text{RMV}(u)$.

règle de réduction 6.

Quand cette règle est appliquée, tout sous-ensemble de feuilles de T' de couleurs deux à deux différentes peuvent être ajoutés à S pour obtenir une solution S^* . On peut donc sans risque ajouter une des feuilles de T' à S .

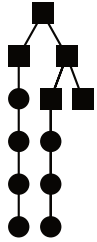
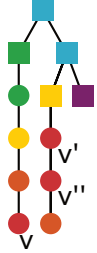
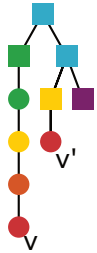
□

Règles de branchement de l'algorithme

Chacune des règles de branchement crée deux sous-problèmes. On écrit $\langle O_1 \parallel O_2 \rangle$ pour exprimer le fait que l'algorithme branche en deux sous-instances, où l'ensemble des opérations O_i est appliqué à l'instance (S, F, D) pour obtenir la i^{eme} sous-instance.

À chacune des règles de branchement on indique le vecteur de branchement correspondant.

La correction de ces règles de branchement est donnée par le lemme 2.11.

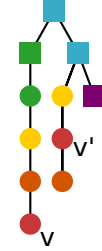
Règle de branchement 1. S'il existe un sommet libre $v \in F$, v est une feuille de T' avec $d(v, S) \geq 4$, alors l'algorithme distingue trois cas :	
Cas 1.a. S'il existe deux nœuds internes v' et v'' dans F satisfaisant $c(v) = c(v') = c(v'')$: $\langle \{\text{ADD}(v), \text{RMV}(\{v', v''\})\} \parallel \text{RMV}(v) \rangle$ vecteur de branchement : $(7, 1)$	
Cas 1.b. S'il existe une feuille $v' \in F$ de T' satisfaisant $c(v) = c(v')$: $\langle \{\text{ADD}(p(v)), \text{RMV}(v')\} \parallel \text{RMV}(p(v)) \rangle$ vecteur de branchement : $(4, 2)$	

Cas 1.c.

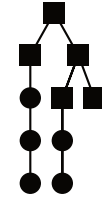
S'il existe un unique nœud interne $v' \in F$ de T' satisfaisant $c(v) = c(v')$:

$\langle \{ \text{ADD}(v), \text{RMV}(v') \} \parallel \{ \text{RMV}(v), \text{ADD}(v') \} \rangle$

vecteur de branchement : $(6, 3)$

**Règle de branchement 2.**

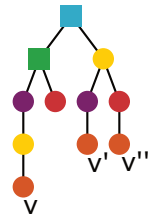
S'il existe un sommet $v \in F$ tel que v est une feuille de T' et $d(v, S) = 3$, alors l'algorithme distingue quatre cas :

**Cas 2.a.**

S'il existe deux feuilles $v', v'' \in F$ de T' satisfaisant $c(v) = c(v') = c(v'')$:

$\langle \{ \text{ADD}(p(v)), \text{RMV}(\{v', v''\}) \} \parallel \text{RMV}(p(v)) \rangle$

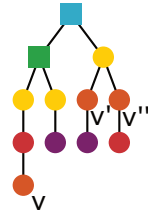
vecteur de branchement : $(4, 2)$

**Cas 2.b.**

S'il existe deux nœuds internes $v', v'' \in F$ de T' satisfaisant $c(v) = c(v') = c(v'')$:

$\langle \{ \text{ADD}(v), \text{RMV}(\{v', v''\}) \} \parallel \text{RMV}(v) \rangle$

vecteur de branchement : $(7, 1)$

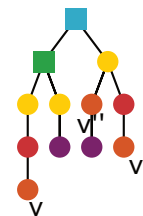


Cas 2.c.

S'il existe une feuille $v' \in F$ et un nœud interne $v'' \in F$ de T' satisfaisant $c(v) = c(v') = c(v'')$:

$\langle \{ \text{ADD}(v''), \text{RMV}(\{v, v'\}) \} \parallel \text{RMV}(v'') \rangle$

vecteur de branchement : (4, 2)

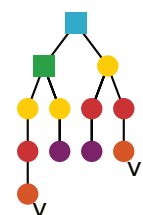


Cas 2.d.

S'il existe un unique sommet $v' \in F$ de T' satisfaisant $c(v) = c(v')$:

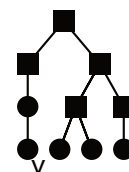
$\langle \text{ADD}(v), \text{RMV}(v') \rangle \parallel \langle \text{RMV}(v), \text{ADD}(v') \rangle$

vecteur de branchement : **(4,3)**



Règle de branchement 3.

S'il existe un sommet libre $v \in F$, v est une feuille de T' avec $d(v, S) = 2$, alors l'algorithme distingue deux cas. :

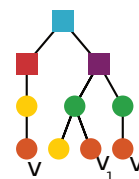


Cas 3.a.

S'il existe $k \geq 2$ sommets $v_1, \dots, v_k \in F$ de T' satisfaisant $c(v) = c(v_1) = \dots = c(v_k)$:

$\langle \{\text{ADD}(v), \text{RMV}(\{v_1, \dots, v_k\})\} \parallel \text{RMV}(p(v)) \rangle$

vecteur de branchement : $(4, 2)$

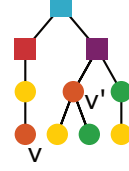


Cas 3.b.

S'il existe un unique sommet $v' \in F$ de T' satisfaisant $c(v) = c(v')$:

$\langle \{ADD(v), RMV(v')\} \parallel \{RMV(p(v)), ADD(v')\} \rangle$

vecteur de branchement : (3,4)



Nous montrons dans le lemme 2.11 la correction de ces règles de branchement.

Lemme 2.11.

Les règles de branchement de cet algorithme sont correctes.

Démonstration.

Commençons par quelques commentaires à propos des règles de branchement de l'algorithme.

Tout d'abord la **règle de branchement 1**, la **règle de branchement 2** et la **règle de branchement 3** font une analyse de cas couvrant tous les cas dans lesquels il y a un sommet libre v tel que $d(v, S) \geq 2$. Si aucune d'entre elles ne peut être appliquée alors tous les sommets libres sont adjacents à S et dans ce cas des règles de réduction peuvent être appliquées.

Considérons maintenant les neuf règles de branchement de notre algorithme. Chacune d'entre elles branche soit en sélectionnant x , soit en rejetant x , étant donné un sommet x approprié. Un branchement est donc obtenu par l'application de la procédure $ADD(x)$ et la seconde par la procédure $RMV(x)$. Clairement une telle règle de branchement est correcte. À chaque branchement on applique également si possible les règles de réduction, ce qui garantit la correction des règles de branchement.

règle de branchement 1.**cas 1.a.**

Dans la première branche, supprimer v' et v'' de F après avoir ajouté v à S est justifié par la **règle de réduction 5**.

cas 1.b.

Dans la première branche, après avoir ajouté $p(v)$ à S , la distance de v à S est de 1. Par conséquent, supprimer v' après avoir ajouté $p(v)$ à S est justifié par la **règle de réduction 3**.

cas 1.c.

Dans la première branche, la suppression de v' de F après avoir ajouté v à S est justifié par la **règle de réduction 5**.

Dans la seconde branche, la suppression de v de F fait de v' l'unique sommet de sa couleur, donc ajouter v' à S après avoir supprimé v de F est justifié par la **règle de réduction 2**.

règle de branchement 2.**cas 2.a.**

Dans la première branche, après avoir ajouté $p(v)$ à S , la distance de v à S est de 1. La suppression de v' et de v'' après l'ajout de $p(v)$ à S est donc justifié par la **règle de réduction 3**.

cas 2.b.

Dans la première branche, la suppression de v' et de v'' de F après l'ajout de v à S est justifié par la **règle de réduction 5**.

cas 2.c.

Dans la première branche, après l'ajout de v'' à S , la couleur de v et de v' n'appartient désormais plus à C' , la suppression de v et de v' de F après l'ajout de v'' à S est justifié par la **règle de réduction 1**.

cas 2.d.

Dans la première branche, la suppression de v' de F après avoir ajouté v à S est justifié par la **règle de réduction 5**.

Dans la seconde branche, v' devient l'unique sommet de sa couleur après la suppression de v de F , donc l'ajout de v' à S après la suppression de v de F est justifié par la **règle de réduction 2**.

règle de branchement 3.**cas 3.a.**

Dans la première branche, $v_1, \dots, v_k$ sont des feuilles de T' (car si ils étaient à distance 1, on aurait appliqué la **règle de réduction 3** pour supprimer v).

Observons néanmoins qu'après l'ajout de $p(v)$ à S , v est à distance 1 de S . Par conséquent la suppression de $v_1, \dots, v_k$ de F après l'ajout de $p(v)$ à S est justifié par la **règle**

de réduction 3.

Après la suppression de $v_1, \dots, v_k$ de F , v devient l'unique sommet de sa couleur, donc ajouter v à S est justifié par la **règle de réduction 2**.

cas 3.b.

Les opérations successives de la première branche sont justifiées de manière similaire au **cas 3.a**.

Dans la seconde branche, on observe que la suppression de $p(v)$ de F supprime aussi v de F , donc v' est alors l'unique sommet de sa couleur, et donc ajouter v' à S est justifié par la **règle de réduction 2**.

□

On établit finalement le théorème suivant :

Théorème 2.12. [*Temps d'exécution de l'algorithme résolvant ENSEMBLE CONNEXE TROPICAL MINIMUM sur les arbres*]

L'algorithme de branchement calcule un ensemble connexe tropical minimum d'un arbre en temps $\mathcal{O}(1.2721^n)$.

Démonstration.

Nous avons montré dans les lemmes 2.10 et 2.11 la correction des règles de branchement et de réduction de l'algorithme.

Pour établir le temps d'exécution de notre algorithme nous établissons maintenant la correction des vecteurs de branchement précédemment annoncés. On se réfère aux préliminaires concernant l'analyse du temps d'exécution d'un algorithme de branchement et de réduction présentés à la section 1.2 du chapitre 1.

Quelques commentaires généraux concernant l'analyse du temps d'exécution de l'algorithme : Pour analyser son temps d'exécution et la diminution de la taille d'un sous-problème lors d'un branchement, on mesure la taille d'une instance (S, F, D) par $|F|$, c'est-à-dire le nombre de sommets libres.

L'argument typique est le suivant. Chaque fois que $\text{ADD}(x)$ est appliquée on déplace $d(v, S)$ sommets de F à S ; la taille de l'instance diminue donc de $d(v, S)$. Chaque fois que $\text{RMV}(x)$ est appliquée on déplace tous les sommets de $T'(x)$ de F vers D ; on diminue donc la taille de l'instance de $|T'(x)|$. Notons que $\text{RMV}(x)$ implique une diminution d'au moins 2 si v est un nœud interne dans T' .

Chaque fois qu'une règle de branchement est appliquée, et qu'aucune règle de réduction ne s'applique, toutes les feuilles $v \in F$ de T' satisfont $d(v, S) \geq 2$.

Analyse du temps :

Déterminons maintenant pour chaque règle de branchement le *facteur de branchement* associé. On notera que l'application d'une règle de réduction diminue la taille de l'instance, à l'exception de la **règle de réduction 0.1** et de la **règle de réduction 0.2** qui respectivement retourne une solution et stoppe la récurrence en cours.

cas 1.a.

Afin de déterminer le *facteur de branchement* de ce cas, on distingue deux cas : dans le premier $v'' \in T'(v')$ (de manière similaire si $v' \in T'(v'')$) alors le *facteur de branchement* est $\tau(d(v, S) + |T'(v')|, |T'(v)|)$. Notons que $|T'(v')| \geq 3$ étant donné que v'' est un nœud interne de T' , donc le *facteur de branchement* est $\tau(7, 1)$.

Dans le second cas, le *facteur de branchement* est $\tau(d(v, S) + |T'(v')| + |T'(v'')|, |T'(v)|) \leq \tau(8, 1)$.

Par conséquent le plus grand *facteur de branchement* est celui du premier cas, c'est-à-dire $\tau(7, 1) \leq 1.2555$.

cas 1.b.

Ce cas a pour *facteur de branchement* $\tau(d(p(v), S) + |T'(v')|, |T'(p(v))|) \leq \tau(4, 2) \leq 1.2721$.

cas 1.c.

Ce cas a pour *facteur de branchement* $\tau(d(v, S) + |T'(v')|, |T'(v)| + d(v', S)) \leq \tau(6, 3) \leq 1.1740$. Notons que $d(v', S) \geq 2$ puisque la **règle de réduction 3** ne s'applique pas.

cas 2.a.

Ce cas a pour *facteur de branchement* $\tau(d(p(v), S) + |T'(v')| + |T'(v'')|, |T'(p(v))|) \leq \tau(4, 2) \leq 1.2721$

cas 2.b.

Ce cas a pour *facteur de branchement* $\tau(d(v, S) + |T'(v')| + |T'(v'')|, |T'(v)|) \leq \tau(7, 1) \leq 1.2555$.

Notons que $d(v', S) \geq 2$ et $d(v'', S) \geq 2$ puisque la **règle de réduction 3** ne s'applique pas.

De plus aucune feuille de T' dans F ne peut être à distance 4 ou plus de S , car sinon l'un des cas de la **règle de branchement 1** aurait été appliquée.

Par conséquent, étant donné que v' et v'' sont des nœuds internes, v' ne peut appartenir à $T'(v'')$, et v'' ne peut appartenir à $T'(v')$.

cas 2.c.

Ce cas a pour *facteur de branchement* $\tau(d(v'', S) + |T'(v)| + |T'(v')|, |T'(v'')|)$. Notons que $d(v'', S) \geq 2$ puisque la **règle de réduction 3** ne s'applique pas. Le *facteur de branchement* est donc d'au plus $\tau(4, 2) \leq 1.2721$.

cas 2.d.

Ce cas a pour *facteur de branchement* $\tau(d(v, S) + |T'(v')|, |T'(v)| + d(v', S)) \leq \tau(4, 3) \leq 1.2208$. Rappelons que comme $d(v', S) \geq 2$, la **règle de réduction 3** ne s'applique pas.

cas 3.a.

Ce cas a pour *facteur de branchement* $\tau(d(p(v), S) + 1 + |T'(v_1)| + \dots + |T'(v_k)|, |T'(p(v))|) \leq \tau(4, 2) \leq 1.2721$.

cas 3.b.

Ce cas a pour *facteur de branchement* $\tau(d(p(v), S) + 1 + |T'(v')|, |T'(p(v))| + d(v', S)) \leq \tau(3, 4) \leq 1.2208$.

Le calcul de tous les nombres de branchements montre que le plus grand d'entre eux est $\tau(2, 4) \leq 1.2721$.

Par conséquent lors de l'exécution de l'algorithme de branchement sur une instance $(\{v\}, V \setminus \{v\}, \emptyset)$ il s'exécute en temps $\mathcal{O}^*(1.2721^n)$ et on fait cet appel pour tout $v \in V$, et par conséquent le temps d'exécution global est $\mathcal{O}^*(1.2721^n)$.

Finalement précisons qu'à chaque fois que la **règle de réduction 0.1** est appliquée on trouve un ensemble connexe tropical. En gardant le plus petit ensemble connexe tropical trouvé récursivement, on résout **ENSEMBLE CONNEXE TROPICAL** pour tout arbre coloré (T, c) . $\square$

2.5 Conclusion

Nous avons étudié dans ce chapitre un problème dont l'objet est la recherche de structures particulières dans des graphes colorés, le problème **ENSEMBLE CONNEXE TROPICAL**. Ce problème est une variante du problème **MOTIF DE GRAPHE** défini par Mc Morris et al. [MWW94]. Les travaux sur ce problème ont été initiés par Angles d'Auriac et al. [AdCH⁺14].

Nous avons donné dans la section 2.2 un algorithme qui résout **ENSEMBLE CONNEXE TROPICAL** en temps $\mathcal{O}^*(1.5359^n)$ et utilisant un espace polynomial sur un graphe quelconque. Cet algorithme est une combinaison de trois techniques : un algorithme en force brute, un algorithme basé sur une réduction vers **ARBRE DE STEINER** utilisant l'algorithme de *Nederlof* pour résoudre **ARBRE DE STEINER** [Ned12], et un algorithme basé sur une réduction vers **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE** utilisant l'algorithme de *Abu-Khzam et al.* pour résoudre **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE** [AKLM11].

Nous avons montré en section 2.3 qu'**ENSEMBLE CONNEXE TROPICAL** n'admet pas d'algorithme sous-exponentiel, même lorsque l'entrée est restreinte aux arbres de hauteur plus 3 sous l'*Hypothèse de Temps Exponentiel*. Cette démonstration est basée sur la réduction d'**ENSEMBLE DOMINANT** vers **ENSEMBLE CONNEXE TROPICAL** d'*Angles d'Auriac et al.* [AdCH⁺14] montrant la *NP*-complétude de **ENSEMBLE CONNEXE TROPICAL**, ainsi que sur la preuve de *Fomin et al.* qu'**ENSEMBLE DOMINANT** n'admet pas d'algorithme sous-exponentiel pour des graphes de degré maximum 6 sous l'*Hypothèse de Temps Exponentiel* [FKW04].

Finalement dans la section 2.4 nous nous sommes intéressés au problème lorsque l'entrée est restreinte aux arbres, et avons conçu un algorithme résolvant **ENSEMBLE CONNEXE TROPICAL** en temps $\mathcal{O}^*(1.2721^n)$ et utilisant un espace polynomial. Cet algorithme est un algorithme de branchement et de réduction comprenant huit règles de réductions et trois règles de branchements contenant un ensemble de sous-cas.

Pour ce qui est de la suite de ce travail, il serait intéressant de concevoir un algorithme pour résoudre **ENSEMBLE CONNEXE TROPICAL** pour des graphes quelconques qui soit indépendant d'autres algorithmes comme **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE** et **ARBRE DE STEINER**. En effet, le fait que l'on ait transformé l'instance, bien que polynomialement, fait que l'on a augmenté la taille de l'instance pour pouvoir appliquer ces techniques. On peut donc espérer qu'il existe un algorithme, typiquement de branchement, pour résoudre **ENSEMBLE CONNEXE TROPICAL** avec un temps d'exécution meilleur que le temps d'exécution de l'algorithme décrit dans ce chapitre.

Concernant notre algorithme sur les arbres, il existe des techniques permettant d'analyser plus finement les règles de branchement, comme la technique *Mesurer pour Conquérir* décrite dans le chapitre 1. Il serait de même intéressant de modifier notre algorithme afin qu'il énumère tous les ensembles connexes tropicaux.

Finalement, il serait intéressant d'aborder d'autres classes de graphes ; en effet, bien que ce problème ait été montré comme étant *NP*-complet sur les arbres et sur les graphes split, il existe peut-être d'autres classes de graphes sur lesquelles il est possible de résoudre dans un temps polynomial **ENSEMBLE CONNEXE TROPICAL**.

En se basant sur notre algorithme de branchement et de réduction de la section 2.4 on peut éventuellement résoudre **ENSEMBLE CONNEXE TROPICAL** sur d'autres classes de graphes, en utilisant les caractéristiques particulières de cette classe, comme sur les graphes planaires, splits ou encore sur les graphes bipartis.

3 Énumération des ensembles dominants minimaux dans quelques classes de graphes

Sommaire

3.1	Introduction	62
3.1.1	Énumération d'ensembles dominants minimaux	62
3.1.2	État de l'art	63
3.1.3	Résultats du chapitre	66
3.1.4	Preliminaires	66
3.2	Énumération dans les graphes splits	67
3.2.1	L'algorithme	68
3.2.2	Correction de l'algorithme et borne combinatoire	71
3.3	Énumération dans les graphes cobipartis	75
3.3.1	L'algorithme	75
3.3.2	Correction de l'algorithme et borne combinatoire	78
3.4	Énumération dans les graphes d'intervalles	81
3.4.1	Graphes partiels et bons triplets	81
3.4.2	L'algorithme	82
3.4.3	Correction de l'algorithme et borne combinatoire	82
3.5	Conclusion	87

3.1 Introduction

Dans ce chapitre nous nous intéressons à un problème classique de graphes, l'énumération d'ensembles dominants minimaux dans les graphes. Étant donné un graphe, un ensemble dominant est un sous-ensemble de sommets tel que chaque sommet du graphe est soit dans cet ensemble, soit adjacent à au moins un sommet de cet ensemble. Un ensemble dominant est minimal s'il ne contient aucun sous-ensemble de sommets qui soit lui-même un ensemble dominant.

Nous donnons dans ce chapitre un algorithme énumérant tous les ensembles dominants minimaux lorsque le graphe en entrée est un *graphe split*, un algorithme lorsque le graphe en entrée est un *graphe cobiparti*, et un algorithme lorsque le graphe en entrée est un *graphe d'intervalles*. Nous améliorons la meilleure borne supérieure connue sur le nombre d'ensembles dominants minimaux dans ces trois classes de graphes, et nous montrons que les bornes obtenues pour les graphes d'intervalles et pour les graphes splits sont les meilleures bornes possibles.

Les résultats de ce chapitre ont été établis en collaboration avec *Jean-François Couturier et Mathieu Liedloff* et ont fait l'objet d'un article présenté à la conférence *ICGT 2014* et publié dans le journal *Theoretical Computer Science* [CLL15].

3.1.1 Énumération d'ensembles dominants minimaux

L'énumération d'objets combinatoires est une problématique classique en théorie des graphes, pour laquelle on trouve un intérêt grandissant depuis les années soixante ; on trouve de nombreux articles dans la littérature à ce sujet.

Une question majeure est d'établir le nombre maximum d'objets satisfaisant certaines propriétés dans un graphe (ou hypergraphe) à n sommets. *Moon et Moser* ont par exemple montré dans [MM65] un fameux résultat établissant que le nombre maximum de stables maximaux dans un graphe à n sommets est de $3^{n/3}$.

C'est l'un des premiers résultats concernant l'établissement d'une borne sur le nombre d'objets combinatoires dans un graphe. Depuis l'avènement des algorithmes exacts exponentiels, de nombreuses bornes de cette nature ont été établies, sur les *stables maximaux*, les *séparateurs minimaux*, les *cliques maximales potentielles*, les *transversaux minimaux dans les hypergraphes*, les *ensembles coupe-cycles minimaux* ou encore les *ensembles dominants minimaux* [MM65, FV10, FGPR08, FHK⁺11, LT08]. Pour ces quatre derniers exemples, les bornes ne sont cependant pas serrées.

Ce qui est remarquable, c'est qu'à l'exception du premier exemple, ces bornes sont établies non pas par des techniques combinatoires, mais en proposant un algorithme exponentiel d'énumération et en analysant le temps d'exécution de cet algorithme.

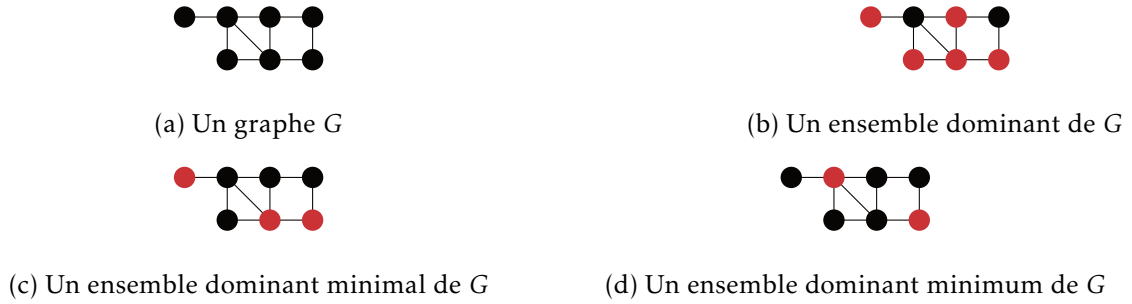


FIGURE 3.1 – Exemple d'ensembles dominants d'un graphe

On se concentre dans ce chapitre sur les ensembles dominants minimaux. Étant donné un graphe $G = (V, E)$, un sous-ensemble de sommets $D \subseteq V$ est un ensemble dominant si tout sommet $v \in V$ est soit dans D , soit adjacent à au moins un sommet de D . Autrement dit, en notant $N[D] = \bigcup_{v \in D} N[v]$, D est un ensemble dominant pour G si $N[D] = V$.

Parmi tous les ensembles dominants d'un graphe, certains sont triviaux : V est un ensemble dominant de G car il contient tous les sommets de V . Nous avons donc choisi d'énumérer des ensembles dominants particuliers, les ensembles dominants minimaux. Étant donné un sous-ensemble de sommets D cet ensemble est dominant minimal si c'est un ensemble dominant, et s'il n'existe pas de sous-ensemble $D' \subset D$ qui soit un ensemble dominant. L'ensemble dominant D est minimum s'il n'existe pas de plus petit ensemble dominant que D pour G . Nous donnons dans la figure 3.1 un exemple d'ensemble dominant, d'ensemble dominant minimal et d'ensemble dominant minimum.

Dans cette partie, nous avons cherché à énumérer les ensembles dominants minimaux dans certaines classes de graphes : les graphes cobipartis, splits et d'intervalles. L'énumération des ensembles dominant attire l'intérêt dans la littérature et est utile pour résoudre d'autres problèmes, comme le calcul du *nombre domatique* d'un graphe (le nombre maximum de partition en ensembles dominants que possède un graphe) [FGPS08].

3.1.2 État de l'art

Dans sa version de décision le problème **ENSEMBLE DOMINANT** consiste à chercher s'il existe, étant donné un graphe G et un entier k , un ensemble dominant pour G qui soit de taille au plus k . Fomin et al. ont montré dans [FGPS08] qu'un graphe a $\mathcal{O}(1.7159^n)$ ensembles dominants minimaux et qu'il existe des graphes avec au moins 1.5705^n ensembles dominants minimaux. Récemment, l'énumération d'ensembles dominants minimaux a été étudiée pour de nombreuses classes de graphes. Un résumé des résultats connus à ce jour est donné dans le tableau 3.1.

classe de graphe	borne inférieure	ref.	borne supérieure	ref.
général	$15^{n/6}$	[FGPS08]	1.7159^n	[FGPS08]
arbres	1.4167^n	[Krz13]	1.4656^n	[Krz13]
cordaux	$3^{n/3}$	[CHvHK13]	1.6181^n	[CHvHK13]
cobipartis	$4^{n/5}$	[CHvHK12]	1.5875^n	[CHvHK12]
splits	$3^{n/3}$	[CHvHK13]	1.4656^n	[CHvHK13]
intervalles propres	$3^{n/3}$	[CHvHK13]	1.4656^n	[CHvHK13]
cographes	$15^{n/6}$	[CHvHK13]	$15^{n/6}$	[CHvHK13]
trivialement parfaits	$3^{n/3}$	[CHvHK13]	$3^{n/3}$	[CHvHK13]
seuils	$\omega(G)$	[CHvHK13]	$\omega(G)$	[CHvHK13]
chaines	$\lfloor n/2 \rfloor + m$	[CHvHK13]	$\lfloor n/2 \rfloor + m$	[CHvHK13]

TABLE 3.1 – Bornes supérieures et inférieures sur le nombre maximum d’ensembles dominants minimaux. Notons que $3^{n/3} \approx 1.4423^n$, $4^{n/5} \approx 1.3195^n$ et $15^{n/6} \approx 1.5704^n$.

Afin d’établir nos nouvelles bornes supérieures, on donne des algorithmes en temps exponentiel pour énumérer tous les ensembles dominants minimaux dans les classes de graphes considérées, et on analyse leur temps d’exécution au pire cas. Mentionnons que l’énumération des ensembles dominants minimaux a aussi été étudiée sous l’aspect des algorithmes à délai polynomial ainsi que via les algorithmes en temps polynomial en la sortie (voir [KLMN11, KLMN12, KLM⁺13]).

Pour conclure cet état de l’art, intéressons-nous plus particulièrement à l’établissement de bornes inférieures sur le nombre maximum d’ensembles dominants maximaux sur les classes de graphes cobipartis, splits et d’intervalles.

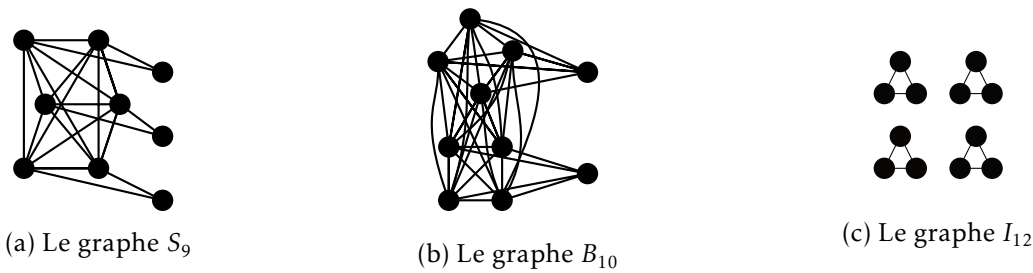


FIGURE 3.2 – Graphes pour lesquels les bornes inférieures données dans les théorèmes 3.1, 3.2 et 3.3 sont atteintes.

Théorème 3.1. [Borne inférieure pour les graphes splits, Couturier et al. [CHvHK13]]
 Il existe des graphes splits à n sommets avec $3^{n/3}$ ensembles dominants minimaux.

Démonstration.

Notons S_n la famille de graphes avec $n = 3k$ sommets composés d'une clique C de taille $2k$ et d'un stable I de taille k , tels que chaque sommet de I a exactement deux voisins dans C et tels que tout sommet de C a exactement un voisin dans I . Les sommets de I n'ont entre eux aucun voisin commun. Un exemple d'un graphe de cette famille est donné dans la figure 3.2a. On vérifie facilement tout d'abord qu'un tel graphe est un graphe split : ses sommets sont partitionnés en deux ensembles disjoints C et I qui sont respectivement une clique et un stable.

Vérifions maintenant qu'un tel graphe a au moins $3^{n/3}$ ensembles dominants minimaux. Pour dominer chaque sommet de I , il faut ajouter exactement un sommet parmi lui-même et son voisinage. Il y a exactement $k = n/3$ sommets dans I , par conséquent il existe $3^{n/3}$ ensembles dominants minimaux pour un tel graphe. $\square$

Théorème 3.2. [Borne inférieure pour les graphes cobipartis, Couturier et al. [CHvHK12]]
 Il existe des graphes cobipartis à n sommets avec $4^{n/5}$ ensembles dominants minimaux.

Démonstration.

Afin d'obtenir une borne inférieure, on définit la famille de graphe B_n avec $n = 5k$ sommets composés de deux cliques X et Y , telles que $|X| = k$ et $|Y| = 4k$. Chaque sommet de X est adjacent à quatre sommets de Y , et chaque sommet de Y est adjacent à exactement un sommet de X . Un exemple de graphe de cette famille est donné dans la figure 3.2b.

En considérant les ensembles dominants minimaux de la forme $\{x, y\}$, avec $x \in X$ et $y \in Y$, les graphes cobipartis en ont $\mathcal{O}(n^2)$, chaque sommet de X (respectivement Y) dominant tous les sommets de la clique X (respectivement Y). Concernant les ensembles dominants minimaux qui sont des sous-ensembles de Y , les graphes de B_n en ont exactement $4^{n/5} \leq 1.3195^n$: afin de dominer chaque sommet de X , il faut ajouter l'un de ses quatre voisins à l'ensemble dominant. Finalement, concernant les ensembles dominants qui sont des sous-ensembles de X , il n'y en a qu'un seul : chaque sommet de Y n'ayant qu'un seul voisin dans X , les k sommets de X doivent appartenir à l'ensemble dominant minimal pour dominer tous les sommets de Y . $\square$

Théorème 3.3. [Borne inférieure pour les graphes d'intervalles, Couturier et al. [CHvHK13]]
 Il existe des graphes d'intervalles propres à n sommets avec $3^{n/3}$ ensembles dominants minimaux.

Démonstration.

Afin d'obtenir une borne inférieure pour les graphes d'intervalles, on définit la famille de graphes I_n avec $n = 3k$ sommets composés d'une collection de k triangles (des cliques de taille 3). Nous illustrons un tel graphe dans la figure 3.2c. Étant donné un triangle, ajouter un sommet à l'ensemble dominant permet de dominer l'ensemble du triangle. Il y a trois choix possibles pour dominer chacun des triangles, il y a par conséquent $3^{n/3}$ ensembles dominants minimaux pour dominer ce graphe d'intervalles, ce qui constitue la borne. $\square$

3.1.3 Résultats du chapitre

Nous verrons tout d'abord en section 3.2 que tout graphe split admet $\mathcal{O}^*(3^{n/3})$ ensembles dominants minimaux. Cette borne est établie via un algorithme de branchement qui, étant donné un graphe split en entrée, énumère tous les ensembles dominants minimaux de ce graphe, la borne supérieure étant déduite de l'analyse du temps d'exécution de cet algorithme. Ce résultat améliore la borne de $\mathcal{O}(1.4656^n)$ établie par Couturier et al. dans [CHvHK13].

Le théorème 3.1 montre que cette borne est la meilleure borne possible, ce qui résout la conjecture de Couturier et al. proposée dans [CHvHK13].

Nous donnerons ensuite en section 3.3 une borne supérieure de $\mathcal{O}(n^2 + 2 \cdot 1.4511^n)$ sur le nombre d'ensembles dominants minimaux lorsque le graphe en entrée est un graphe cobiparti. Nous établirons cette borne en donnant un algorithme de branchement énumérant chaque ensemble dominant minimal du graphe en entrée, puis en déduisant cette borne de l'analyse du temps d'exécution de cet algorithme. Ce résultat améliore la borne de $\mathcal{O}(1.5875^n)$ établie par Couturier et al. dans [CHvHK12].

Finalement, nous donnerons en section 3.4 une borne supérieure de $\mathcal{O}^*(3^{n/3})$ sur le nombre d'ensembles dominants minimaux lorsque le graphe en entrée est un graphe d'intervalles. Cet algorithme énumère, étant donné un graphe d'intervalles en entrée, tous les ensembles dominants minimaux de ce graphe. Nous montrons ensuite que le nombre d'ensembles dominants minimaux énumérés par notre algorithme est de $\mathcal{O}^*(3^{n/3})$, ce qui améliore la précédente borne établie par Couturier et al. dans [CHvHK13] de $\mathcal{O}(1.4656^n)$ ensembles dominants minimaux pour les graphes d'intervalles propres et l'étend à la superclasse des graphes d'intervalles.

Combiné avec le théorème 3.3, nous montrerons que cette borne est la meilleure borne possible, ce qui résout la conjecture de Couturier et al. proposée dans [CHvHK13].

3.1.4 Préliminaires

Un ensemble D est un **ensemble dominant** d'un graphe $G = (V, E)$ si $N[D] = V$. Étant donné un sommet v et un ensemble X , on note $N_X(v) = N(v) \cup X$ et $d_X(v) = |N_X(v)|$.

Nous définissons maintenant une notion assez simple, mais qui sera essentielle tout au long de ce chapitre :

Définition 3.4. *[Voisin privé]*

Étant donné un ensemble D , un sommet $x \in D$ et un sommet $v \in V$ (où éventuellement $x = v$), v est un **voisin privé** de x si $v \notin N[D \setminus \{x\}]$.

Étant donné un ensemble dominant D , si cet ensemble contient un sommet x n'ayant aucun voisin privé, alors tous ses voisins (lui y compris) ont au moins un autre voisin que x dans D , et par conséquent $D \setminus \{x\}$ est aussi un ensemble dominant. Réciproquement, si chaque sommet x de D a un voisin privé, ce sommet n'est pas dominé par $D - \{v\}$; D est donc un ensemble dominant minimal.

On en déduit la propriété suivante :

Propriété 3.5.

Un ensemble dominant D est minimal si chaque sommet de D a un voisin privé.

Deux graphes $G = (V_G, E_G)$ et $H = (V_H, E_H)$ sont **isomorphes**, que l'on note $G \cong H$, s'il existe une bijection f entre V_G et V_H telle que $\{u, v\} \in E_G$ si et seulement si $\{f(u), f(v)\} \in E_H$.

Pour énumérer les ensembles dominants minimaux d'un graphe nous avons conçu des algorithmes de branchement. Un tel algorithme est généralement composé d'un ensemble de règles d'arrêt, de règles de réduction et de règles de branchement. Ces dernières sont typiquement responsables du facteur exponentiel dans l'expression du temps d'exécution. On réfère le lecteur à la section 1.2 du chapitre 1 ainsi qu'au livre de Fomin et Kratsch (voir le chapitre 2 de [FK10]) pour de plus amples détails sur ces algorithmes, ainsi que sur leur analyse.

3.2 Énumération dans les graphes splits

Rappelons qu'un graphe split est un graphe dont les sommets peuvent être partitionnés en une clique et un stable.

Dans cette section nous montrerons que les graphes splits ont au plus $3^{n/3}$ ensembles dominants minimaux qui peuvent être énumérés en temps $\mathcal{O}^*(3^{n/3}) = \mathcal{O}(1.4423^n)$. Nous établirons cette borne en concevant un algorithme qui, étant donné un graphe split, énumère tous les ensembles dominants minimaux pour ce graphe. Finalement, nous déduirons de son temps d'exécution une borne supérieure sur le nombre d'ensembles dominants minimaux qu'un tel graphe peut avoir.

3.2.1 L'algorithme

Notre algorithme énumérant les ensembles dominants minimaux est une collection de règles de branchement, que nous décrirons par la suite. Une règle ne sera appliquée que si les règles précédentes ne sont pas applicables.

On note DS l'ensemble dominant minimal en cours de construction (initialement vide), et v est un sommet de la clique C avec le plus grand nombre de voisins dans le stable I . Typiquement, lors de l'ajout de v dans DS , on supprimera également son voisinage dans I : ces derniers étant dominés, il n'est plus nécessaire de les conserver dans la branche en cours. Lorsque l'on choisira de rejeter v , on ne supprimera alors que v , ses voisins dans I n'étant pas encore dominés.

L'algorithme construit un ensemble dominant I qui est minimal. Un tel ensemble est un ensemble dominant minimal sauf si cet ensemble est I est si l'un des sommets de C n'a aucun voisin dans I .

Aux règles d'arrêt et de réductions que nous décrivons nous associons une *illustration* : les sommets « bleus » représentent les sommets appartenant à C , la clique du graphe et les sommets « rouges » représentent les sommets appartenant à l'ensemble stable I . Bien que C soit une clique, nous ne représentons pas dans sur les figures les arêtes entre deux sommets de C par souci de clarté.

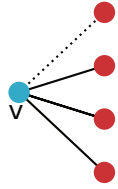
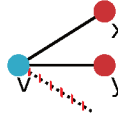
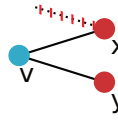
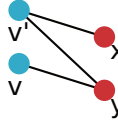
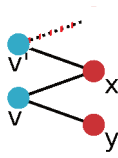
Règle d'arrêt de l'algorithme

Nous donnons tout d'abord une règle d'arrêt pour notre algorithme. Cette règle est appelée lorsque le graphe ne contient pas (ou plus) d'arête : dans ce cas, elle construit un ensemble dominant en temps polynomial pour le graphe en entrée contenant l'ensemble DS en cours de construction, ainsi que quelques autres sommets. La correction de cette règle de réduction est donnée dans le lemme 3.7.

Règle d'arrêt. Le graphe ne contient plus d'arêtes : on ajoute tous les sommets restants dans I à DS. Si $DS \cap C$ n'est pas vide alors les sommets restants dans C sont supprimés et $DS \cup I$ est retourné, sinon pour chaque sommet v de C on retourne $DS \cup I \cup \{v\}$.	
---	---

Règles de branchement de l'algorithme

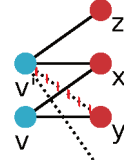
Chacune des règles de branchement crée un ensemble de sous-problèmes. Pour chacune des règles de branchement on indique le vecteur de branchement correspondant. La correction de ces règles de branchement ainsi de leur vecteur de branchement est donnée dans le lemme 3.6.

Règle de branchement 1. $d_I(v) \geq 3$: — soit on ajoute v à DS ; — soit on rejette v. Le vecteur de branchement est $(4, 1)$, et $\tau(4, 1) < 1.3803$.	
Règle de branchement 2. $d_I(v) = 2$; notons $N_I(v) = \{x, y\}$. Sans perte de généralité, supposons que $d(x) \leq d(y)$:	
Cas 2.1. $d(x) = 1$: — Soit on ajoute x à DS ; — Soit on rejette x. Le vecteur de branchement est $(2, 3)$, et $\tau(2, 3) < 1.3248$.	
Cas 2.2. $d(x) = 2$; notons v' un autre voisin de x. On distingue plusieurs sous-cas :	
Cas 2.2.1. $d_I(v') = 1$: — Soit on ajoute v à DS ; — Soit on ajoute v' à DS ; — Soit on ajoute x à DS. Le vecteur de branchement est $(4, 3, 3)$, et $\tau(4, 3, 3) < 1.3954$.	

Cas 2.2.2. $d_I(v') \geq 2$ et $y \notin N(v')$; notons z l'autre voisin de v' dans I :

- Soit on ajoute v à DS et x est son voisin privé ;
- Soit on ajoute v à DS et y est son voisin privé ;
- Soit on rejette v et on ajoute v' à DS ;
- Soit on ajoute x à DS .

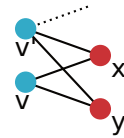
Le vecteur de branchement est $(4, 6, 4, 3)$, et $\tau(4, 6, 4, 3) < 1.4064$.



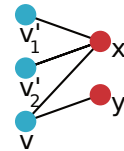
Cas 2.2.3. $d_I(v') \geq 2$ et $y \in N(v')$:

- Soit on ajoute v à DS ;
- Soit on ajoute v' à DS ;
- Soit on ajoute x à DS .

Le vecteur de branchement est $(4, 4, 3)$, et $\tau(4, 4, 3) < 1.3533$.



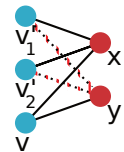
Cas 2.3. $d(x) = 3$; notons v, v'_1 et v'_2 ses voisins. Sans perte de généralité on a $d(v'_1) \leq d(v'_2)$. On distingue les sous-cas suivants :



Cas 2.3.1. $y \notin N(v'_1) \cup N(v'_2)$:

- Soit on ajoute v à DS et x est son voisin privé ;
- Soit on ajoute v à DS , y est son voisin privé et $v'_1 \in DS$;
- Soit $v \in DS$, y est son voisin privé et $v'_2 \in DS$;
- Soit on rejette v .

Le vecteur de branchement est $(5, 7, 8, 1)$, et $\tau(5, 7, 8, 1) < 1.4331$.

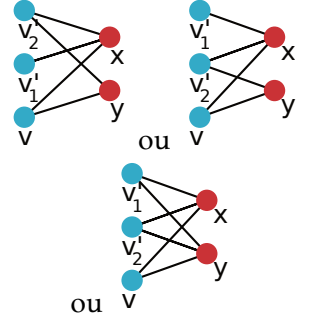


Cas 2.3.2. $y \in N(v'_1) \cup N(v'_2)$:

- Soit on ajoute x à DS et x est son voisin privé ;
- Soit on ajoute v à DS , y est son voisin privé et $v'_i \in DS$;
- Soit on rejette v .

Si $y \notin N(v'_1) \cap N(v'_2)$, **le vecteur de branchement est** $(5, 7, 1)$, **et** $\tau(5, 7, 1) < 1.3971$.

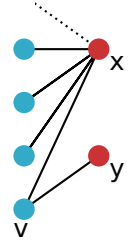
Si $y \in N(v'_1) \cap N(v'_2)$, **le vecteur de branchement est** $(5, 1)$, **et** $\tau(5, 1) < 1.3248$.



Cas 2.4. $d(x) \geq 4$:

- Soit on ajoute v à DS et x est son voisin privé ;
- Soit on ajoute v à DS et y est son voisin privé ;
- Soit on rejette v .

Le vecteur de branchement est $(6, 6, 1)$, **et** $\tau(6, 6, 1) < 1.3881$.



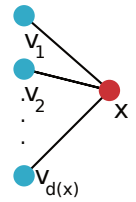
Notons qu'à ce stade, $d_I(v) \leq 1$ pour tout $v \in C$.

Règle de branchement 3. $x \in I$ de degré maximum ; notons

$v_1, \dots, v_{d(x)}$ ses voisins :

- Soit on ajoute x à DS ;
- Soit on rejette x et on ajoute v_i à DS pour $1 \leq i \leq d(x)$;

Le vecteur de branchement est $(d(x)+1, d(x)+1, \dots, d(x)+1)$ **de longueur** $d(x)+1$, **et** $\tau(d(x)+1, d(x)+1, \dots, d(x)+1) \leq 3^{1/3} < 1.4423$ **(le pire cas est atteint pour** $d(x) = 2$ **)**.



3.2.2 Correction de l'algorithme et borne combinatoire

Nous montrons maintenant que l'algorithme que nous venons de décrire énumère bien tous les ensembles dominants minimaux, étant donné un graphe split.

Lemme 3.6. *Les règles de branchement de l'algorithme sont correctes.*

Démonstration. Observons tout d'abord que la correction de l'algorithme est simple à montrer, chacune des règles de branchement considérant toutes les possibilités pour dominer un sommet, et pour prendre ou rejeter un sommet de l'ensemble dominant. Nous donnons pour chacune des règles de branchement le vecteur de branchement associé ainsi que son facteur de branchement. Nous montrons maintenant que ces règles couvrent bien la totalité des cas de figure possible, et que les vecteurs de branchements sont corrects.

règle de branchement 1.

Dans le cas où on ajoute v à DS , v et ses voisins dans I sont dominés et peuvent donc être supprimés. Le sommet v ayant au moins 3 voisins dans I , au moins 4 sommets sont supprimés dans ce cas.

Dans le cas où on rejette v , on ne domine aucun sommet. On peut néanmoins supprimer v , étant donné que v appartient à C , le prochain sommet de C que l'on ajoutera à DS dominera par la même occasion v . Nous traitons le cas où aucun sommet de C n'est ajouté à DS dans la **règle d'arrêt**. Nous supprimons exactement 1 sommet.

Note. Après l'application de cette règle de branchement, dans l'instance tout sommet de C a au plus 2 voisins dans I .

règle de branchement 2 :

cas 2.1. le sommet v étant le seul voisin de x , on a deux possibilités pour dominer x :

- Dans le premier cas, on ajoute x à DS et, par minimalité de DS , v ne peut pas appartenir à l'ensemble dominant. On peut donc supprimer x et v , soit exactement 2 sommets.
- Dans le second cas, x n'ayant que v comme voisin, on ajoute v à DS . Par minimalité de DS et étant donné que y est dominé par v , on le supprime (on rappelle que C est une clique, et que $v \in C$). On supprime donc exactement 3 sommets.

cas 2.2.

cas 2.2.1. x ayant deux voisins : v , et un autre sommet noté v' , on a trois possibilités pour dominer x :

- Dans le cas où on ajoute v à DS , x étant le seul voisin de v' dans I , il n'y a pas d'ensemble dominant minimal contenant v' . On supprime donc au moins 4 sommets (v , x , y et v').
- Dans le cas où on ajoute v' à DS , on supprime x , v et v' , soit exactement 3 sommets. Notons que le cas où on ajoute v à DS a été précédemment considéré, donc on sait que v ne peut pas être ajouté à DS (et v est dominé par v').

- Dans le cas où on ajoute x à DS , on supprime v , v' et x , soit exactement 3 sommets.

cas 2.2.2. On distingue ici quatre cas pour dominer x :

- Dans le cas où on ajoute v à DS et x est le voisin privé de v , x étant le voisin privé de v , le sommet v' ne peut pas appartenir à DS et on le supprime donc de l'instance. On ajoute v à DS et on supprime son voisinage (contenant x et y). On supprime donc 4 sommets (v , v' , x et y).
- Dans le cas où on ajoute v à DS et y est le voisin privé de v , étant donné que x n'est pas le voisin privé de v , v' appartient donc à DS . Rappelons aussi que $d(x) \leq d(y)$. Le sommet x ayant deux voisins, y en a au moins deux. On supprime donc x , y , v' , z et $N(y)$, soit au moins 6 sommets.
- Dans le cas où on rejette v et on ajoute v' à DS , x , z , v et v' sont supprimés, soit 4 sommets.
- Dans le cas où on rejette v et v' et on ajoute x à DS , ces 3 sommets sont supprimés.

cas 2.2.3. Il y a trois possibilités pour dominer x :

- Dans le cas où on ajoute v à DS , v' ne peut pas être ajouté à DS par minimalité de DS ; on le supprime donc. Par conséquent, on supprime 4 sommets (v , v' , x et y).
- Dans le cas où on ajoute v' à DS , par minimalité de DS v ne peut être ajouté à DS . On supprime également 4 sommets.
- Dans le cas où on ajoute x à DS : v et v' n'appartenant pas à DS , x appartient à DS et $N[x]$ est supprimé, soit 3 sommets.

cas 2.3.

cas 2.3.1.

- Dans le cas où on ajoute v à DS et x est son voisin privé, les sommets v , x et y sont dominés par DS et sont donc supprimés. De même, v'_1 et v'_2 ne peuvent appartenir à DS (car x est le voisin privé de v). On supprime donc 5 sommets : v , $N_I(v)$ et $N(x)$.
- Dans le cas où on ajoute v à DS avec y son voisin privé et $v'_1 \in DS$, on a $d_I(v'_1) \geq 2$ (si $d_I(v'_1) = 1$ on n'applique pas cette branche étant donné qu'il n'est pas minimal d'ajouter v'_1 et v à DS). On supprime v , v'_1 , x , y , $N(y)$ (car v est son voisin privé) et $N_I(v'_1)$, soit au moins 7 sommets.
- Dans le cas où on ajoute v à DS avec y son voisin privé et $v'_2 \in DS$, pour la même raison que la branche précédente on suppose que $d_I(v'_2) \geq 2$ sinon on l'ignore. On supprime donc v , v'_1 , v'_2 , x , y , $N(y)$, et $N_I(v'_2)$, soit au moins 8 sommets.
- Finalement on peut rejeter v : dans ce cas, on supprime uniquement v .

cas 2.3.2.

- Dans le cas où on ajoute v à DS et x est son voisin privé, on supprime 5 sommets : v , $N_I(v)$ et $N(x)$.
- Dans le cas où on ajoute v à DS , y est son voisin privé et $v'_i \in DS$, le sommet y ne pouvant pas être un voisin privé de v , on n'applique pas ce branchement si $y \in N(v'_1) \cap N(v'_2)$. Soit i tel que $y \notin N(v'_i)$. Si $d_I(v'_i) = 1$ alors on n'applique pas ce branchement étant donné qu'il n'est pas minimal d'ajouter à la fois v'_1 et v à DS . Sinon on supprime v , v'_i , x , y , $N(y)$ (étant donné que v est son voisin privé) et $N_I(v'_i)$; soit au moins 7 sommets;
- Finalement on peut rejeter v : dans ce cas on supprime uniquement v ;

cas 2.4.

- Dans le cas où on ajoute v à DS et x est son voisin privé, on supprime v , x , y et $N(x)$, soit au moins 6 sommets.
- Dans le cas où on ajoute v à DS et y est son voisin privé, on supprime v , x , y et $N(y)$, soit au moins 6 sommets;
- Dans le cas où on rejette v , on supprime uniquement v .

Observations. Certains ensembles dominants peuvent être générés plus d'une fois par l'algorithme : on remarque que le cas où x et y sont uniquement dominés par v est traité par les deux premiers branchements, et donc un même ensemble dominant peut être généré à deux étapes différentes de l'algorithme.

Remarquons également que si les règles précédentes ne sont plus applicables, alors tout sommet $v \in C$ a au plus un voisin dans I .

règle de branchement 3.

Dans ce branchement nous considérons un sommet de I de degré maximum, et notons $v_1, \dots, v_{d(x)}$ ses voisins. Observons qu'à partir du moment où x est dominé, tous ses voisins sont immédiatement supprimés. Pour dominer x , nous avons $d(x) + 1$ possibilités : soit on choisit x , soit on choisit l'un de ses $d(x)$ voisins, et pour chacun des cas on supprime $d(x) + 1$ sommets. La solution au pire cas du vecteur de branchement est obtenue pour $d(x) = 2$. Par conséquent, $\tau(d(x) + 1, d(x) + 1, \dots, d(x) + 1) \leq 3^{1/3} < 1.4423$.

Notons que ce cas correspond à la construction établie par Couturier et al. [CHvHK13] et rappelée en section 3.1 pour établir la borne inférieure sur le nombre d'ensembles dominants minimaux pour les graphes splits. □

Nous montrons maintenant la correction de la règle d'arrêt :

Lemme 3.7. *La règle d'arrêt de l'algorithme est correcte.*

Démonstration. On doit ajouter tous les sommets restants de I dans DS : par définition, étant donné $v_i \in I$, tout voisin de v_i appartient à C . Ainsi, si v_i n'a pas été supprimé, alors aucun de ses voisins n'est dans DS , et par conséquent la seule solution pour dominer v_i est de l'ajouter à DS .

Si $DS \cap C$ n'est pas vide, alors $DS \cap C$ est un ensemble dominant pour $G[C]$, et on peut donc supprimer les sommets restants dans C .

Si $DS \cap C$ est vide, tout sommet restant dans C est un ensemble dominant minimal pour C (qui rappelons-le est une clique) alors $DS \cup \{v_c\}$ est un ensemble dominant minimal $\forall v_c \in C \setminus DS$. On retourne donc chacun de ces ensembles dominants minimaux. $\square$

De l'algorithme et des lemmes 3.6 et 3.7 on déduit le théorème suivant :

Théorème 3.8. *Les graphes splits ont au plus $3^{n/3}$ ensembles dominants minimaux qui peuvent être énumérés en temps $\mathcal{O}^*(3^{n/3}) = \mathcal{O}(1.4423^n)$.*

Couturier et al. montrent dans [CHvHK13], comme nous le rappelons dans le théorème 3.1, qu'il existe des graphes splits avec $\Omega^*(3^{n/3})$ ensembles dominants minimaux. Notre borne est donc serrée.

3.3 Énumération dans les graphes cobipartis

Un graphe $G = (V, E)$ est cobiparti si son ensemble de sommets V peut être partitionné en deux ensembles X et Y tel que $G[X]$ et $G[Y]$ induisent des graphes complets, c'est-à-dire que X et Y sont deux cliques. Nous montrons que les graphes cobipartis ont au plus $n^2 + 2 \cdot 1.4510..^n = \mathcal{O}(1.4511^n)$ ensembles dominants minimaux. Afin d'établir cette borne nous donnons un algorithme qui, étant donné un graphe cobiparti, énumère tous ses ensembles dominants minimaux en temps $\mathcal{O}(1.4511^n)$, ce qui améliore la borne de $\mathcal{O}(1.5875^n)$ donnée par Couturier et al. dans [CHvHK12].

3.3.1 L'algorithme

Observons tout d'abord que tout ensemble dominant minimal DS d'un graphe cobiparti respecte exactement l'une des trois conditions suivantes :

- Soit $DS \cap Y$ et $DS \cap X$ ne sont pas vides. Dans ce cas, $|DS| = 2$ car chaque ensemble dominant contenant un sommet de X et un sommet de Y est minimal. Il y a au plus $|X| \cdot |Y| = \mathcal{O}(n^2)$ ensembles dominants comme ceci pouvant être facilement énumérés ;

- Soit $DS \subseteq Y$;
- Soit $DS \subseteq X$.

Notre algorithme se concentre sur l'énumération des ensembles dominants minimaux qui sont des sous-ensembles de X . Clairement, les sous-ensembles de Y peuvent aussi être énumérés par notre algorithme. Il suffit en effet d'exécuter l'algorithme en renommant Y et X , par X et Y respectivement.

On suppose également que les ensembles X et Y sont non-vides. Clairement, si X et Y sont tout-deux vides, l'ensemble $DS = \emptyset$ est l'unique ensemble dominant minimal de $G = (\emptyset, \emptyset)$. Si X est vide, aucun ensemble dominant strictement inclus dans X ne peut dominer Y . Si Y est vide, tout sommet de X est un ensemble dominant minimal de G .

Décrivons maintenant notre algorithme de manière similaire à celui de la section 3.2 pour les graphes splits. Nous décrivons tout d'abord quelques règles de réduction, et ensuite les règles de branchement. Encore une fois, une règle de branchement est appliquée si aucune règle de réduction et aucune règle de branchement apparaissant avant ne peut être appliquée.

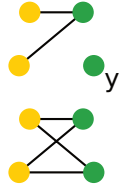
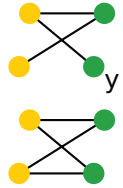
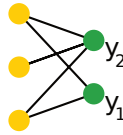
Nous associons à chacune des règles de réduction et de branchement une *illustration* : les sommets « jaunes » représentent les sommets appartenant à l'ensemble X et les sommets « verts » représentent les sommets appartenant à l'ensemble Y . Bien que X (respectivement Y) soit une clique, nous ne représentons pas les arêtes entre deux sommets de X (respectivement Y) dans les illustrations par souci de clarté.

On note DS l'ensemble dominant minimal en cours de construction par notre algorithme. Notons que DS est initialement vide.

Règles de réduction de l'algorithme

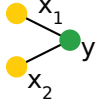
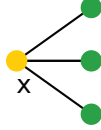
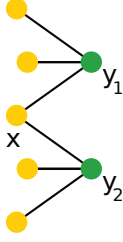
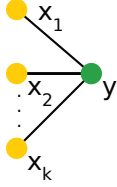
La correction de ces règles de réduction est donnée dans le lemme 3.9.

Règle de réduction 1. $\exists x \in X$ tel que $d_Y(x) = 0$: On supprime x.	
---	--

Règle de réduction 2. $\exists y \in Y$ tel que $d_X(y) = 0$: On stoppe le branchement en cours.	
Règle de réduction 3. $\exists y \in Y$ tel que $d_X(y) = 1$: On ajoute l'unique voisin de y dans X à DS.	
Règle de réduction 4. $\exists y_1, y_2 \in Y$ tel que $N_X(y_1) \subseteq N_X(y_2)$: On supprime y_2.	
Règle de réduction 5. G est vide : On retourne l'ensemble dominant courant DS.	

Règles de branchement de l'algorithme

Décrivons maintenant les règles de branchement de notre algorithme, dont nous donnons la correction dans le lemme 3.10.

Règle de branchement 1. $\exists y \in Y$ tel que $d_X(y) = 2$. On définit $N_X(y) = \{x_1, x_2\}$: — Soit on ajoute x_1 à DS ; — Soit on rejette x_1 et on ajoute x_2 à DS. Le vecteur de branchement est $(2, 3)$, et $\tau(2, 3) < 1.3258$.	
Règle de branchement 2. $\exists x \in X$ tel que $d_Y(x) \geq 3$: — Soit on ajoute x à DS ; — Soit on rejette x. Le vecteur de branchement est $(4, 1)$, et $\tau(4, 1) < 1.3803$.	
Règle de branchement 3. $\exists x \in X$ tel que $d_Y(x) = 2$. On définit $N_Y(x) = \{y_1, y_2\}$: — Soit on ajoute x à DS et y_1 est son voisin privé ; — Soit on ajoute x à DS et y_2 est son voisin privé ; — Soit on rejette x. Le vecteur de branchement est $(5, 5, 1)$, et $\tau(5, 5, 1) < 1.4511$.	
Règle de branchement 4. $y \in Y$ et $N_X(y) = \{x_1, x_2, \dots, x_k\}$: — On branche sur l'un des k voisins de y. Le vecteur de branchement est $(k + 1, k + 1, \dots, k + 1)$ de longueur k, et $\tau(k + 1, k + 1, \dots, k + 1) < 1.3196$ (le pire cas est atteint pour $k = 4$).	

3.3.2 Correction de l'algorithme et borne combinatoire

Montrons tout d'abord que chacune des règles de réduction précédemment décrites sont correctes.

Lemme 3.9.

Les règles de réduction de l'algorithme sont correctes.

Démonstration.

La correction de ces règles de réduction est simple à montrer. Rappelons tout d'abord que X et Y sont deux cliques, et que l'on cherche à énumérer les ensembles dominants minimaux inclus dans X .

Règle de réduction 1. Si on ajoute x à DS , dès que l'on ajoutera un autre sommet de X à DS , le sommet x n'aura plus de voisin privé, par conséquent DS ne pourra être étendu à un ensemble dominant minimal du graphe en entier. Le sommet x étant adjacent à tous les autres sommets de X , l'ajout d'un sommet de X dominera par la même occasion x , et on peut donc le supprimer.

Règle de réduction 2. Tous les voisins de y sont dans Y , il n'existe donc pas de sommet dans X dominant y . Par conséquent, DS ne peut pas être étendu à un ensemble dominant du graphe en entier.

Règle de réduction 3. Si y n'a qu'un seul voisin dans X , ajouter ce voisin à DS est la seule possibilité pour obtenir un ensemble dominant du graphe contenant DS .

Règle de réduction 4. Pour dominer y_1 on doit choisir l'un de ses voisins et l'ajouter à l'ensemble dominant en cours de construction à une certaine étape de l'algorithme. Si le voisinage de y_2 est inclus dans celui de y_1 , alors tout sommet de X qui domine y_1 domine par la même occasion y_2 . On peut donc simplement supprimer y_2 .

Règle de réduction 5. Si le graphe est vide, alors tout sommet du graphe est soit dans DS , ou est dominé par DS , qui est donc un ensemble dominant minimal du graphe. On peut donc le retourner. La correction de cette dernière règle est finalisée par la correction des règles de branchement, montrant notamment que l'ensemble dominant construit par notre algorithme est bien minimal.

□

Nous montrons dans le prochain lemme la correction des règles de branchement de notre algorithme :

Lemme 3.10.

Les règles de branchement de l'algorithme sont correctes.

Démonstration.

Notons tout d'abord qu'après application des règles de réduction, les sommets de X ont au moins 1 voisin dans Y , et les sommets de Y ont au moins 2 voisins dans X .

Règle de branchement 1. Pour dominer y on doit ajouter l'un de ses voisins à DS . Soit on ajoute x_1 à DS , soit on ne l'ajoute pas et dans ce cas on doit ajouter x_2 à DS . Dans le premier cas, on supprime deux sommets (y et x_1), dans le second on supprime trois sommets (x_1 , x_2 et y).

Règle de branchement 2. Ici, soit on ajoute x à DS et on supprime ses voisins dans Y , donc au moins quatre sommets, soit on le rejette, et on le supprime.

Note. Les cas de branchement suivant ayant été atteints, les cas de branchement et les règles de réduction précédents impliquent que pour tout $y \in Y$, $d_X(y) \geq 3$ et pour tout $x \in X$, $d_Y(x) \leq 2$.

Règle de branchement 3. x a deux voisins dans Y ayant au moins trois voisins dans X . Dans le cas où on ajoute x à DS et y_1 est son voisin privé, y_1 ne peut avoir un autre voisin que x qui appartienne à DS . On supprime donc y_1 , y_2 ainsi que $N_X[y_1]$, soit au moins 5 sommets. Dans le cas où on ajoute x à DS et y_2 est son voisin privé, par des arguments similaires au branchement précédent on supprime 5 sommets. Finalement, si on rejette x on supprime 1 sommet (le sommet x).

Note. Maintenant, pour tout $y \in Y$, $d_X(y) \geq 3$ et pour tout $x \in X$, $d_Y(x) = 1$. En négligeant les arêtes de $G[Y]$ et $G[X]$, le graphe se réduit à une collection disjointe d'étoiles dont le centre est un sommet de Y et dont les feuilles appartiennent à X . Chaque ensemble dominant minimal doit précisément contenir une feuille de chacune des étoiles. On en déduit le dernier branchement :

Règle de branchement 4. Pour dominer y on doit choisir un sommet parmi ses k voisins dans X . Chaque sommet de x n'ayant qu'un seul voisin dans Y , on supprime y et ses k voisins dans chaque branche, et on obtient par conséquent le vecteur de branchement $(k+1, k+1, \dots, k+1)$ de longueur k . Finalement, par une analyse mathématique nous obtenons que le facteur de branchement est maximal pour $k = 4$. Remarquons que le graphe correspond à la construction pour établir la borne inférieure sur le nombre d'ensembles dominants minimaux pour les graphes cobipartis, établie par Couturier et al. dans [CHvHK12] et rappelée en section 3.1.

□

De l'algorithme ainsi que des lemmes 3.9 et 3.10 on en déduit le théorème suivant :

Théorème 3.11. *Les graphes cobipartis ont $\mathcal{O}(n^2 + 2 \cdot 1.4511^n)$ ensembles dominants minimaux qui peuvent être énumérés en temps $\mathcal{O}(1.4511^n)$.*

Le temps d'exécution de l'algorithme est obtenu par le plus grand facteur de branchement, c'est-à-dire $\tau(5,5,1) \leq 1.4511$. Rappelons que notre algorithme se restreint à l'énumération des ensembles dominants minimaux qui sont des sous-ensembles de X . En considérant les ensembles dominants minimaux qui sont des sous-ensembles de Y , et ceux contenant un sommet de X et un sommet de Y , le nombre d'ensembles dominants minimaux est d'au plus $n^2 + 2 \cdot 1.4511^n$.

3.4 Énumération dans les graphes d'intervalles

Les graphes d'intervalles sont des graphes pour lesquels on peut affecter à chaque sommet un intervalle de l'ensemble des réels de telle sorte que deux sommets sont adjacents si et seulement si les intervalles correspondant s'intersectent.

S'il existe une affectation d'intervalles pour laquelle aucun intervalle n'est proprement contenu dans un autre, ce graphe est un graphe d'intervalles propres.

Couturier et al. ont montré dans le théorème 3.3 que les graphes d'intervalles propre ont au plus 1.4656^n ensembles dominants minimaux [CHvHK13]. Nous améliorons dans cette section cette borne et montrons que tout graphe d'intervalles (non nécessairement propre) a au plus $3^{n/3}$ ensembles dominants minimaux, ce qui répond à la question ouverte soulevée dans [CHvHK12]. En particulier, nous donnons un algorithme énumérant tous les ensembles dominants minimaux en temps $\mathcal{O}^*(3^{n/3})$. Étant donné qu'il existe des graphes d'intervalles ayant $3^{n/3}$ ensembles dominants minimaux, notre résultat est le meilleur possible.

3.4.1 Graphes partiels et bons triplets

Nous définissons tout d'abord deux notions nécessaires à la compréhension de l'algorithme ainsi qu'à l'analyse de sa correction.

Définition 3.12. [Graphe partiel]

Étant donné un sommet $z \in V$, on définit le **graphe partiel** $G_{\leq z}$ comme le sous-graphe induit $G[S]$ où S est l'ensemble des sommets dont l'extrémité gauche est inférieure (ou égale) à $r(z)$. De manière analogue, on définit $G_{> z}$ comme étant le sous-graphe induit par les sommets dont l'extrémité gauche est supérieure à $r(z)$.

Un exemple de graphe partiel est donné dans la figure 3.3.

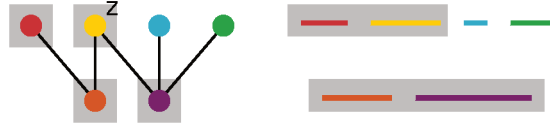


FIGURE 3.3 – Exemple de graphe partiel $G_{\leq z}$, étant donné un graphe d'intervalles G et un intervalle z . $G_{\leq z}$ est le graphe induit par les sommets « grisés ».

Définition 3.13. [Bon triplet]

Étant donné deux sommets $y, z \in V$ et un sous-ensemble $DS \subseteq V$, $\{y, z, DS\}$ est un **bon triplet** si :

1. $z \in DS$ et est le sommet de DS ayant la plus grande extrémité droite ;
2. DS est un ensemble dominant minimal du graphe partiel $G_{\leq z}$;
3. y est le voisin privé de z dans $G_{\leq z}$, au regard de DS , avec la plus petite extrémité droite.

3.4.2 L'algorithme

Étant donné un bon triplet $\{y, z, DS\}$, le sommet y est utilisé par notre algorithme pour garantir la propriété de *minimalité*. Il garantit en effet que le sommet z est nécessaire à l'ensemble dominant et que y est l'un de ses voisins privés. Le sommet z pouvant avoir plusieurs voisins privés, on garde uniquement celui avec la plus petite extrémité droite.

Nous supposons qu'un modèle d'intervalles normalisé $\mathcal{I}$ de $G = (V, E)$ fait partie de l'entrée ; c'est-à-dire $\cup_{v \in V} \{l(v), r(v)\} = \{1, 2, \dots, 2n\}$. L'**Algorithme 2** énumère tous les ensembles dominants minimaux d'un graphe d'intervalles donné en entrée.

La correction ainsi que l'analyse du temps d'exécution de cet algorithme, et l'établissement de la borne supérieure sur le nombre d'ensembles dominants minimaux sont donnés dans la section suivante.

3.4.3 Correction de l'algorithme et borne combinatoire

Les lemmes suivants assurent la correction de notre algorithme **EnumDomSetInterval**.

Dans le lemme 3.14 nous montrons que l'intervalle avec la plus petite extrémité droite est un voisin privé d'un sommet de tout ensemble dominant minimal pour G :

Lemme 3.14. Soit y l'intervalle de $\mathcal{I}$ tel que $r(y)$ est minimum. Dans tout ensemble dominant minimal DS , y est le voisin privé d'un sommet de DS .

Algorithme 2 : EnumDomSetInterval($G = (V, E), \mathcal{I}$)

Entrées : Un graphe d'intervalles et son modèle d'intervalles normalisé $\mathcal{I}$.

Sorties : Tous les ensembles dominants minimaux correspondant au graphe en entrée.

Initialisation :

Soit y l'intervalle de $\mathcal{I}$ ayant la plus petite extrémité droite

Soit $\mathcal{L} \leftarrow \emptyset$

pour chaque $z \in N[y]$ **faire**

 Ajouter le triplet $\{y, z, \{z\}\}$ à $\mathcal{L}$

Boucle « tant que » :

tant que $\mathcal{L}$ *n'est pas vide* **faire**

 Extraire un triplet $\{y, z, DS\}$ de $\mathcal{L}$

 Soit y' le sommet tel que $r(y') = \min_{x \in \mathcal{I}} \{r(x) \mid l(x) > r(z)\}$

si il n'existe pas de tel sommet y' **alors**

DS est un ensemble dominant minimal

sinon si $N[y'] \setminus N[y]$ *est vide* **alors**

 on oublie le triplet $\{y, z, DS\}$

sinon

pour chaque $z' \in (N[y'] \setminus N[y])$ **faire**

 On ajoute le triplet $\{y', z', DS \cup \{z'\}\}$ à $\mathcal{L}$

Démonstration.

Soit $DS_y = N[y] \cap DS$. DS étant un ensemble dominant, DS_y n'est pas vide. Si $DS_y = \{z\}$, alors y est un voisin privé de z (éventuellement $z = y$). Supposons que $|DS_y| \geq 2$ et soit z et z' deux sommets de DS_y . Sans perte de généralité, on suppose que $r(z') < r(z)$. L'intervalle y étant l'intervalle de $\mathcal{I}$ avec l'extrémité droite la plus petite, on a $N[z'] \subseteq N[z]$, ce qui contredit le fait que DS est un ensemble dominant minimal, car $DS \setminus \{z'\}$ est lui aussi un ensemble dominant. $\square$

Le lemme suivant montre comment étendre un bon triplet à partir du prochain sommet non-dominé y' :

Lemme 3.15. Soit $\{y, z, DS\}$ un bon triplet. Soit y' un sommet tel que $r(y') = \min_{x \in \mathcal{I}} \{r(x) \mid l(x) > r(z)\}$ et supposons qu'un tel sommet y' existe. Dans tout ensemble dominant minimal DS' , tel que $DS \subseteq DS'$, le sommet y' est le voisin privé d'un sommet de DS' .

Démonstration.

Tout d'abord, supposons que y' existe et n'est pas le voisin privé d'un sommet de DS' . L'ensemble $DS'_{y'} = N[y'] \cap DS'$ contient alors au moins deux sommets que nous notons z'_1 et z'_2 . Observons que par définition de y' , les sommets z'_1, z'_2 appartiennent à $DS' \setminus DS$. Sans perte de généralité supposons que $r(z_1) < r(z_2)$. Le sommet y' étant le sommet non dominé par DS avec la plus petite extrémité droite, alors $DS' \setminus \{z'_1\}$ est aussi un ensemble dominant, ce qui contredit le fait que DS' est un ensemble dominant minimal et donc $|DS'_{y'}| = 1$. $\square$

Le lemme suivant montre que certains bons triplets ne peuvent pas être étendus à des ensembles dominants minimaux du graphe en entier et peuvent donc être oubliés.

Lemme 3.16. *Soit $\{y, z, DS\}$ un bon triplet. Soit y' un sommet tel que $r(y') = \min_{x \in \mathcal{I}} \{r(x) \text{ s.t. } l(x) > r(z)\}$. Si un tel sommet y' existe et que l'ensemble $N[y'] \setminus N[y]$ est vide, alors il n'existe pas d'ensemble dominant minimal DS' tel que $DS \subseteq DS'$.*

Démonstration.

Clairement si un tel sommet y' existe, alors l'ensemble DS ne peut pas être un ensemble dominant du graphe en entier. L'ensemble $N[y']$ est non vide mais étant donné que $N[y'] \setminus N[y]$ est vide, chaque sommet de $N[y']$ est aussi un voisin de y . Par conséquent, tout ensemble dominant DS' tel que $DS \subseteq DS'$ doit contenir z (car il est dans DS) et un voisin commun de y' et de y (c'est-à-dire un sommet de $N[y']$). Le sommet y étant le voisin privé de z avec l'extrémité droite la plus petite, $DS' \setminus \{z\}$ est par conséquent aussi un ensemble dominant. Il n'est donc pas possible d'étendre l'ensemble DS à un ensemble dominant minimal. $\square$

L'idée essentielle de l'algorithme **EnumDomSetInterval** est de déterminer (itérativement) un sommet non-dominé y' qui puisse être utilisé comme voisin privé d'un sommet z' . L'algorithme teste alors tous les candidats z' comme sommet appartenant à un ensemble dominant minimal. Le lemme suivant montre que tous les triplets ajoutés par l'algorithme **EnumDomSetInterval** à la liste $\mathcal{L}$ sont bons et correspondent à des ensembles dominants minimaux de graphes partiels.

Lemme 3.17. *À chaque étape de l'algorithme **EnumDomSetInterval**, chaque triplet $\{y, z, DS\}$ de $\mathcal{L}$ est un bon triplet.*

Démonstration.

Soit $\{y, z, DS\}$ un triplet de la liste $\mathcal{L}$. Considérons comment un tel triplet $\{y, z, DS\}$ a été ajouté à $\mathcal{L}$. Soit il a été obtenu en étendant un triplet $\{\tilde{y}, \tilde{z}, \tilde{DS}\}$ dans la boucle « tant que », soit il a été ajouté à $\mathcal{L}$ durant la *phase d'initialisation* (c'est-à-dire avant la boucle « tant que »).

Par le lemme 3.14, le sommet y avec la plus petite extrémité droite doit être le voisin privé d'un sommet $z \in N[y]$. Par conséquent chaque triplet $\{y, z, DS\}$, avec $DS = \{z\}$, ajouté durant la phase d'initialisation est clairement un bon triplet.

Supposons maintenant que $\{y, z, DS\}$ est obtenu à partir d'un bon triplet $\{\tilde{y}, \tilde{z}, \tilde{DS}\}$ dans la boucle « tant que ». Par le lemme 3.15, le sommet y non dominé par $\tilde{DS}$ avec la plus petite extrémité

droite est le voisin privé (avec la plus petite extrémité droite) d'un sommet $z \in N[y]$. Le sommet $\tilde{y}$ étant le voisin privé de $\tilde{z}$, z est choisi dans $N[y] \setminus N[\tilde{y}]$. Par conséquent, z est le sommet de DS avec l'extrémité droite la plus grande. $\widetilde{DS}$ étant un ensemble dominant minimal de $G_{\leq \tilde{z}}$, il s'ensuit que $DS = \widetilde{DS} \cup \{z\}$ est un ensemble dominant minimal de $G_{\leq z}$. $\square$

Le lemme suivant montre que si des bons triplets sont retournés par notre algorithme, et que tous les sommets du graphe en entrée sont dominés, alors un ensemble dominant minimal du graphe en entrée a été trouvé et est retourné. Par conséquent tous les sous-ensembles retournés par notre algorithme sont des ensembles dominants minimaux du graphe en entrée. Les lemmes 3.18, 3.19 et 3.20 montrent que tous les ensembles dominants minimaux sont en effet retournés.

Lemme 3.18. *Soit $\{y, z, DS\}$ un bon triplet de $\mathcal{L}$ tel qu'il n'y ait aucun sommet donc l'extrémité gauche ne soit plus grande que $r(z)$. Alors $G_{\leq z} \cong G$ et DS est un ensemble dominant minimal de G .*

Démonstration.

Si un bon triplet $\{y, z, DS\}$ donné n'a pas de sommets y' tel que $r(y') = \min_{x \in \mathcal{I}} \{r(x) \text{ tel que } l(x) > r(z)\}$, alors tous les sommets sont dominés par DS . Par définition d'un bon triplet, DS est un ensemble dominant minimal du graphe. $\square$

Lemme 3.19. *Soit DS un ensemble dominant minimal de G . Il existe alors deux sommets y et z tels que $\{y, z, DS\}$ est un bon triplet.*

Démonstration.

Soit z le sommet de DS ayant l'extrémité droite la plus grande. L'ensemble DS étant un ensemble dominant minimal, d'après la propriété 3.5 (définie dans les préliminaires de la section 3.1), z a au moins un voisin privé. Soit y son voisin privé ayant l'extrémité droite la plus petite. Alors $\{y, z, DS\}$ est un bon triplet. $\square$

Prouvons finalement que tous les bons triplets possibles du graphe en entrée sont considérés durant l'exécution de notre algorithme :

Lemme 3.20. *Soit $\{y, z, DS\}$ un bon triplet d'un graphe en entrée G . Alors à une certaine étape de l'algorithme l'ensemble $\{y, z, DS\}$ appartient à $\mathcal{L}$.*

Démonstration.

Montrons par induction que tout bon triplet est considéré par l'algorithme à une certaine étape et ajouté à $\mathcal{L}$.

Soit $\{y, z, DS\}$ un bon triplet du graphe en entrée G . Clairement, si $DS = \{z\}$ alors le bon triplet $\{y, z, DS\}$ est produit durant la *phase d'initialisation*, c'est-à-dire la première boucle « pour » de l'algorithme. Le sommet y avec l'extrémité droite la plus petite est choisie et chacun de ses voisins $z \in N[y]$ est alors considéré pour ajouter un triplet $\{y, z, \{z\}\}$ à $\mathcal{L}$.

Supposons que $|DS| \geq 2$ et, par induction, que tout les bons triplets possibles $\{\tilde{y}, \tilde{z}, \widetilde{DS}\}$ ont été

générés par l'algorithme (et ajoutés à $\mathcal{L}$ à une certaine étape de l'algorithme) pour tout sommet $\tilde{z}$ avec $r(\tilde{z}) < l(y)$.

Soit $DS' = DS \setminus \{z\}$ et soit $z' \in DS'$ tel que $r(z')$ soit maximum. $\{y, z, DS\}$ étant un bon triplet, l'ensemble DS est un ensemble dominant minimal de $G_{\leq z}$ et, d'après la propriété 3.5, z' a un voisin privé. DS' est donc un ensemble dominant minimal de $G_{\leq z'}$. Soit y' le voisin privé de z' avec l'extrémité droite la plus petite. Alors $\{y', z', DS \setminus \{z\}\}$ est un bon triplet et $r(z') < l(y)$. D'après l'hypothèse d'induction, à une certaine étape de l'algorithme ce bon triplet $\{y', z', DS \setminus \{z\}\}$ appartient à $\mathcal{L}$. Grâce à la boucle « tant que », un tel triplet est extrait de $\mathcal{L}$ et est étendu au triplet $\{y, z, DS\}$, où y est choisi pour être le sommet non dominé par DS' (c'est-à-dire $l(y) > r(z')$), avec la plus petite extrémité droite, et z en tant que l'un de ses voisins. $\square$

Nous avons montré dans le lemme précédent que tous les bons triplets sont produits par notre algorithme et que ceux correspondants aux ensembles dominants minimaux du graphe en entrée sont retournés. On montre dans le lemme suivant que le temps d'exécution de l'algorithme **EnumDomSetInterval** est borné par $\mathcal{O}^*(3^{n/3})$, cet algorithme utilisant un temps polynomial par bon triplet et le nombre de ces bons triplets étant $\mathcal{O}^*(3^{n/3})$. De plus, les ensembles dominants minimaux du graphe en entrée correspondant aux bons triplets retournés par l'algorithme, on en déduit que tout graphe d'intervalles a au plus $3^{n/3}$ ensembles dominants minimaux.

Lemme 3.21. *Les graphes d'intervalles à n sommets ont $\mathcal{O}^*(3^{n/3})$ bons triplets.*

Démonstration.

Soit $\{y, z, DS\} \in \mathcal{L}$ un bon triplet. Soit $R = \{x \text{ s.t. } l(x) > r(y)\}$. Pour tout bon triplet $\{\tilde{y}, \tilde{z}, \widetilde{DS}\}$ obtenu par une séquence d'extensions à partir de $\{y, z, DS\}$ (c'est-à-dire tel que $DS \subseteq \widetilde{DS}$), on a $(\widetilde{DS} \setminus DS) \subseteq R$.

L'algorithme **EnumDomSetInterval** étend tout d'abord $\{y, z, DS\}$ en fixant un sommet y' (tel que $r(y') = \min_{x \in \mathcal{I}} \{r(x) \text{ s.t. } l(x) > r(z)\}$), puis en regardant tous ses voisins z' (formellement, $z' \in N[y'] \setminus N[y]$). D'après le lemme 3.15, y' est le voisin privé de z' et aucun autre voisin de y' (à l'exception de z') ne peut être ajouté à DS .

En dénotant $L(k)$, avec $k = |R|$, le nombre de telles séquences d'extensions possibles (comprenant la séquence *vide* sans extension), $L(k)$ représente le nombre de bons triplets qui peuvent être obtenus à partir de $\{y, z, DS\}$ (y compris $\{y, z, DS\}$ lui-même).

Rappelons que z' a été choisi à partir de $N[y'] \setminus N[y] = N[y'] \cap R$. Soit $d = |N[y'] \cap R|$. Le nombre maximum de bons triplets est donné par $L(k) \leq 1 + d \cdot L(k - d)$ (c'est-à-dire le triplet courant plus tous ceux qui peuvent être obtenus par des extensions).

On établit par les méthodes de calcul standard (voir e.g. [FK10]), que $L(k) = \mathcal{O}^*(3^{k/3})$, cette récurrence étant maximisée lorsque $d = 3$. Comme $k \leq n$, alors le nombre maximum de bons triplets d'un graphe d'intervalles est $\mathcal{O}^*(3^{n/3})$. $\square$

Le corollaire suivant répond à une question ouverte, posée par *Couturier et al.* dans [CHvHK13] :

Corollaire 3.22. *Tout graphe d'intervalles a au plus $3^{n/3}$ ensembles dominants minimaux.*

Démonstration.

Nous avons établi une borne supérieure sur le nombre de bons triplets qu'un graphe d'intervalles possède dans la preuve du lemme 3.21. On observe que les ensembles dominants minimaux correspondent à certains de ces bons triplets, ceux qui ne peuvent plus être étendus. En notant $T(k)$ le nombre maximum d'ensembles dominants minimaux retournés par l'algorithme **EnumDomSetInterval**, et en utilisant des arguments similaires à ceux de la preuve du lemme 3.21, $T(k)$ respecte les récurrences $T(k) \leq d \cdot T(k - d)$ et par conséquent $T(k) \leq 3^{n/3}$. $\square$

On déduit finalement de l'algorithme **EnumDomSetInterval** ainsi que des lemmes et corollaires précédents le théorème 3.23 :

Théorème 3.23.

*L'algorithme **EnumDomSetInterval** énumère tous les ensembles dominants minimaux en temps $\mathcal{O}^*(3^{n/3})$.*

Démonstration.

Les lemmes 3.18 et 3.19 assurent que l'algorithme **EnumDomSetInterval** énumère bien tous les ensembles dominants minimaux du graphe d'intervalle en entrée. D'après le lemme 3.21, tout graphe d'intervalles a au plus $\mathcal{O}^*(3^{n/3})$ bons triplets, qui peuvent être énumérés par l'algorithme **EnumDomSetInterval** en temps $\mathcal{O}^*(3^{n/3})$. $\square$

On conclut cette section en observant que l'algorithme **EnumDomSetInterval** peut facilement être transformé en algorithme utilisant un espace polynomial. En effet, la liste $\mathcal{L}$ de bons triplets peut être implémentée comme une *pile* (une structure de donnée du type *Last In First Out*). Dans ce cas, on ne garde dans $\mathcal{L}$ qu'un nombre polynomial de bons triplets.

3.5 Conclusion

Nous nous sommes intéressés tout au long de ce chapitre à un problème classique en algorithmique des graphes, l'énumération de structures particulières dans des graphes non-orientés, en l'occurrence les ensembles dominants minimaux par inclusion. Nous nous sommes plus particulièrement intéressés à l'énumération de tels ensembles dans des classes de graphes particulières : dans les graphes splits, les graphes cobipartis ainsi que dans les graphes d'intervalles.

Nous avons tout d'abord montré en section 3.2 que tout graphe split contient au plus $3^{n/3}$ ensembles dominants minimaux, et nous avons donné un algorithme utilisant un espace polynomial qui, étant donné un graphe split, énumère tous les ensembles dominants minimaux de ce graphe en temps $\mathcal{O}(1.4423^n)$. *Couturier, Heggernes, van't Hof et Kratsch* ayant montré dans [CHvHK13] qu'il existe des graphes splits ayant $3^{n/3}$ ensembles dominants minimaux (nous rappelons d'ailleurs

la famille des graphes pour lesquels cette borne est atteinte en section 3.1), on en conclut que notre borne est serrée, ce qui résout une conjecture établie par Couturier et al. dans [CHvHK13].

Nous avons ensuite montré en section 3.3 que tout graphe cobiparti contient au plus $n^2 + 2 \cdot 1.4511^n$ ensembles dominants minimaux, et nous donnons un algorithme de branchement et réduction énumérant en $\mathcal{O}(1.4511^n)$ ces ensembles. Ce résultat améliore la précédente borne sur le nombre d'ensembles dominants minimaux de $\mathcal{O}(1.5875^n)$ établi par Couturier et al. dans [CHvHK12].

Nous donnons finalement une borne supérieure de $\mathcal{O}^*(3^{n/3})$ sur le nombre d'ensembles dominants minimaux lorsque le graphe en entrée est un graphe d'intervalles. Nous établissons un algorithme énumérant les ensembles dominants d'un graphe d'intervalles en entrée en temps $\mathcal{O}^*(3^{n/3})$. Couturier et al. ont montré dans [CHvHK13] que les graphes d'intervalles propres ont au plus 1.4656^n ensembles dominants minimaux. Notre résultat améliore cette borne, et la généralise aux graphes d'intervalles non nécessairement propres. Couturier et al. ont également montré qu'il existe des graphes d'intervalles qui ont $3^{n/3}$ ensembles dominants minimaux (nous rappelons la famille des graphes d'intervalles ayant exactement $3^{n/3}$ ensembles dominants minimaux en section 3.1). Nous déduisons ainsi que notre borne est la meilleure borne possible à un facteur polynomial près.

Il est à noter que la différence entre la borne supérieure et la borne inférieure sur le nombre d'ensembles dominants minimaux dans les graphes cobipartis est relativement large. Il serait par conséquent très intéressant de réduire cet écart, soit en améliorant la borne supérieure, en utilisant un algorithme en branchement et réduction analysant de manière plus détaillée les différents cas ou encore en analysant de manière plus précise les vecteurs et facteurs de branchement de notre algorithme via la technique *Mesurer pour Conquérir* décrite dans la section 1.2 du chapitre 1, soit en trouvant un graphe cobiparti contenant plus d'ensembles dominants minimaux que la meilleure borne inférieure actuelle (établie par Couturier et al. dans [CHvHK12]). Nous pensons cependant que notre borne supérieure est surestimée.

Quelques indices montrent qu'il est difficile d'obtenir une borne précise, l'énumération d'ensembles dominants minimaux dans les graphes cobipartis généralisant certains problèmes, comme l'énumération des transversaux dans les hypergraphes (voir par exemple [LT08]).

Nous pourrions de même rechercher des bornes supérieures sur des classes de graphes autres que celles étudiées dans la section 3.1, notamment pour des superclasses des classes de graphes pour lesquelles nous avons une borne précise, et ainsi généraliser nos résultats à un plus grand nombre de graphes.

Finalement, notons que l'écart entre la borne inférieure (qui est de 1.5704^n) et la borne supérieure (qui est de 1.7159^n) dans le cas général est relativement grand. Il serait donc très intéressant de réduire cet écart, bien que cela semble être très difficile. En effet, la technique actuellement

utilisée par [FGPS08] est un algorithme très fin en branchements et réductions, dont l'analyse est faite via *Mesurer pour Conquérir*.

4 Algorithme pour la domination romaine faible sur les graphes d'intervalles

Sommaire

4.1	Introduction	92
4.1.1	Domination, domination romaine et domination romaine faible	92
4.1.2	État de l'art	94
4.1.3	Résultats de ce chapitre	95
4.1.4	Preliminaires et notations	95
4.2	Domination Romaine Faible dans le cas général	97
4.3	Domination Romaine Faible sur les graphes d'intervalles	99
4.3.1	Description de l'algorithme	99
4.3.2	Analyse du temps d'exécution	106
4.4	Conclusion	111

4.1 Introduction

Nous nous intéressons dans ce chapitre à la recherche d'une fonction de domination romaine faible de coût minimum. Étant donné un graphe $G = (V, E)$, une fonction $f : V \rightarrow \{0, 1, 2\}$ est une fonction de domination romaine faible pour G si tout sommet v est *défendu* (c'est-à-dire que v a un voisin u , éventuellement $u = v$, tel que $f(u) \geq 1$), et si pour tout sommet v tel que $f(v) = 0$ il existe un voisin u de v tel que $f(u) \geq 1$ et tel que la fonction $f_{u \rightarrow v}$ définie par :

$$f_{u \rightarrow v}(x) = \begin{cases} 1 & \text{si } x = v \\ f(u) - 1 & \text{si } x = u \\ f(x) & \text{si } x \notin \{u, v\} \end{cases}$$

ne contient aucun sommet non défendu. Le coût d'une telle fonction f est égal à $\sum_{v \in V} f(v)$.

Nous donnons un algorithme qui, étant donné un graphe d'intervalles en entrée, construit une fonction de domination romaine faible de coût minimum. Cet algorithme utilise un temps et un espace polynomial en la taille du graphe en entrée. Nous modifions ensuite légèrement cet algorithme afin qu'il résolve le problème en temps linéaire en la taille de l'entrée.

Il y a une certaine dynamique dans ce problème par rapport à d'autres problèmes classiques de domination : on cherche en effet à résister à une attaque, tout en restant dominant.

Les résultats de ce chapitre sont actuellement soumis à *EUROCOMB 2015, European Conference on Combinatorics, Graph Theory and Applications*. Une version journal est aussi en cours de préparation, en combinaison avec le résultat de *Chapelle et al.* [CCC⁺13].

4.1.1 Domination, domination romaine et domination romaine faible

Nous avons vu dans la section 1.2 du chapitre 1, le problème de décision **ENSEMBLE DOMINANT** : étant donné un graphe $G = (V, E)$ non orienté et un entier k , on recherche un ensemble dominant S de taille au plus k , c'est-à-dire un sous-ensemble $S \subseteq V$ tel que tout sommet $v \in V$ appartient soit à S , soit à (au moins) un voisin dans S . Ce problème est l'un des problèmes *NP*-complets les plus célèbres [GJ79] et a été très largement étudié durant les dernières années.

Diverses variantes du problème **ENSEMBLE DOMINANT** ont été définies et étudiées du point de vue structurel comme du point de vue algorithmique [HHS98b, HHS98a]. De nombreux articles et livres concernent la domination dans les graphes [HHS98a]. On s'intéresse généralement à des problèmes d'optimisation, dont le but est de protéger un ensemble de sommets en utilisant le plus petit nombre de « jetons » possible (qui doivent être placés sur ces sommets et qui pourront éventuellement, selon le problème, être déplacés d'un sommet à un autre).

Parmi ces variantes, citons notamment le problème **DOMINATION ROMAINE MINIMUM**, introduit par Cockayne et al. dans [CDJHH04] et motivé par [RR00, Ste99]. La problématique est un décret de l'empereur Constantin le Grand :

« L'empereur Constantin le Grand est au IV^{me} siècle à la tête d'un grand empire, constitué d'un ensemble de régions, et d'un ensemble de légions pour les défendre. Le but est alors d'assurer la protection de cet empire en plaçant intelligemment les légions de manière à en utiliser le moins possible. »

Dans la **DOMINATION ROMAINE MINIMUM** on considère les règles suivantes pour protéger un graphe : un sommet peut se protéger lui-même s'il possède une légion et peut protéger chacun de ses voisins s'il en a deux. La problématique consiste alors à minimiser le nombre de légions à utiliser pour défendre tous les sommets (c'est-à-dire toutes les régions de l'empire).

Le problème **DOMINATION ROMAINE MINIMUM** est proche de plusieurs autres variantes de domination du type « défense ». Citons tout d'abord le problème **DOMINATION SÉCURISÉE MINIMUM** [CF03, CGG⁺05, GM09] : étant donné un graphe $G = (V, E)$ non orienté, $V' \subseteq V$ est un *ensemble dominant sécurisé* si tout sommet $u \in V \setminus V'$ est adjacent à un sommet $v \in V'$ tel que $V' \setminus \{v\} \cup \{u\}$ est un ensemble dominant.

Une autre variante est le problème **DOMINATION ÉTERNELLE MINIMUM** [GHH05, GK08] : étant donné un graphe $G = (V, E)$, un sous-ensemble de sommets $D_0 \subseteq V$ est un *ensemble dominant éternel* de G si D_0 est un ensemble dominant, et si pour toute suite de k sommets $v_0, \dots, v_{k-1}$ il existe une suite de k sommets $v'_0, \dots, v'_{k-1}$ telle que pour tout $i \in \{0, k-1\}$, $\{v'_i, v_i\} \in E$ et l'ensemble D_{i+1} défini par $D_{i+1} = (D_i \setminus \{v'_i\}) \cup \{v_i\}$ est un ensemble dominant.

Nous nous intéressons dans ce chapitre à une autre variante de **DOMINATION ROMAINE MINIMUM**, le problème **DOMINATION ROMAINE FAIBLE MINIMUM** défini en 2003 par Henning et Hedetniemi [HH03]. L'idée est la suivante : un sommet v peut être protégé si l'un de ses voisins possède une légion pouvant être déplacée vers v de telle manière que tous les sommets restent protégés. Ce problème, dans lequel on introduit une certaine dynamique par rapport au problème de **DOMINATION ROMAINE MINIMUM** peut être formellement défini de la manière suivante :

Problème **DOMINATION ROMAINE FAIBLE MINIMUM**

entrée. Un graphe non-orienté $G = (V, E)$.

sortie. Une fonction de coût minimum $f : V \rightarrow \{0, 1, 2\}$ telle que tout sommet v est défendu par f (il existe $u \in N[v]$ tel que $f(u) \geq 1$), et pour tout sommet v tel que $f(v) = 0$ il existe $u \in N[v]$ tel que $f(u) \geq 1$ et tout sommet est défendu par la fonction $f_{u \rightarrow v}$ définie par $f_{u \rightarrow v}(x) = 1$ si $x = v$, $f_{u \rightarrow v}(x) = f(x) - 1$ si $x = u$ et $f_{u \rightarrow v}(x) = f(x)$ sinon. Le coût d'une telle fonction f est égal à $\sum_{v \in V} f(v)$.

4.1.2 État de l'art

Le problème **ENSEMBLE DOMINANT MINIMUM** est un problème très étudié depuis quelques années du point de vue des algorithmes exacts. Dans le cas général, *Grandoni* donne dans [Gra04, Gra06] un algorithme exact exponentiel qui calcule un ensemble dominant minimum pour tout graphe à n sommets en temps $\mathcal{O}^*(2^{0.930n}) = \mathcal{O}(1.9053^n)$ et en espace polynomial, puis montre comment réduire le temps d'exécution de cet algorithme à $\mathcal{O}^*(2^{0.850n}) = \mathcal{O}(1.8026^n)$ en utilisant un espace exponentiel. *Fomin, Grandoni et Kratsch* améliorent dans [FGK09] l'analyse de ces algorithmes via l'analyse *Mesurer pour Conquérir* présentée dans la section 1.2 du chapitre 1 et montrent que l'algorithme utilisant un espace polynomial calcule un ensemble dominant minimum en temps $\mathcal{O}^*(2^{0.610n}) = \mathcal{O}(1.5263^n)$ et que l'algorithme utilisant un espace exponentiel calcule un tel ensemble en temps $\mathcal{O}^*(2^{0.598n}) = \mathcal{O}(1.5137^n)$. Ils donnent finalement une borne inférieure de $\Omega(2^{0.396n}) = \Omega(1.3159^n)$ sur le temps d'exécution de l'algorithme utilisant un espace polynomial. Les meilleurs algorithmes connus sont, à ma connaissance, ceux d'*Iwata* décrits dans [Iwa11] qui résolvent **ENSEMBLE DOMINANT MINIMUM** en $\mathcal{O}(1.4689^n)$ en temps et en espace, et en temps $\mathcal{O}(1.4689^n)$ en utilisant un espace polynomial.

Concernant **DOMINATION ROMAINE**, *Liedloff et al.* montrent dans [KLLP08] que le problème peut être résolu en temps linéaire en la taille du graphe en entrée lorsque celui-ci est un graphe d'intervalles ou un cografe, et en temps polynomial lorsque le graphe en entrée est sans triplet astéroïde ou lorsque ce graphe admet une d-pieuvre. D'autres résultats structurels sont donnés dans [CDJHH04, CKPW09, Fer08].

DOMINATION ROMAINE FAIBLE a notamment été étudié d'un point de vue structurel. Il s'agit surtout de l'étude du coût de la fonction de domination romaine faible minimum d'un graphe G (noté $\gamma_r(G)$) pour un certain nombre de classes de graphes.

Hedetniemi et Henning ont montré dans [HH03] que pour tout graphe G on a $\gamma(G) \leq \gamma_r(G) \leq 2\gamma(G)$, $\gamma(G)$ étant la cardinalité d'un ensemble dominant minimum pour G . Ils caractérisent également des graphes G pour lesquels on a $\gamma_r(G) = \gamma(G)$ et des graphes G pour lesquels on a $\gamma_r(G) = 2\gamma(G)$.

Cockayne et al. ont montré dans [CF03] que pour tout graphe G on a $\gamma_S(G) \leq v(G)$, $v(G)$ étant la taille du plus grand couplage de G et $\gamma_S(G)$ le coût d'une domination sécurisée minimum pour G . Ils montrent également que dans les graphes sans griffes on a $\gamma_r(G) = \gamma_S(G) \leq 2\gamma(G)$.

Cockayne et al. donnent dans [CGG⁺05] des bornes pour les coûts des fonctions de domination faible minimum pour un certain nombre de classes de graphes, comme les graphes complets, bipartis complets, multipartis complets, les cycles, ou encore les produits cartésiens de cycles et les produits cartésiens de chemins.

Mai et Pushpam [MMP11] caractérisent les arbres et les graphes splits G pour lesquels on a $\gamma_r(G) = \gamma(G)$, et donnent les valeurs de $\gamma_r(G)$ lorsque G est une chenille, une grille $2 \times n$ ou un arbre binaire complet.

Le seul article à ma connaissance traitant des aspects algorithmiques du problème DOMINATION ROMAINE FAIBLE est l'article de *Chapelle et al.* [CCC⁺13]. Ils donnent dans cet article deux algorithmes plus efficaces que l'algorithme en force brute en temps $\mathcal{O}^*(3^n)$ obtenu en énumérant toutes les 3-partitions de sommets du graphe en entrée : le premier s'exécute en $\mathcal{O}^*(2^n)$ et nécessite un espace exponentiel ; le second s'exécute en $\mathcal{O}^*(2.2279^n)$ et nécessite un espace polynomial.

4.1.3 Résultats de ce chapitre

Nous rappelons dans la section 4.2 les résultats précédemment cités de *Chapelle et al.* établis dans [CCC⁺13], puis nous nous intéressons dans la section 4.3 à la conception d'un algorithme exact lorsque le graphe en entrée est un graphe d'intervalles. Nous montrerons que dans ce cas le problème devient polynomial et nous donnons un algorithme calculant une fonction de domination romaine faible minimum en temps linéaire en la taille de l'entrée.

4.1.4 Préliminaires et notations

Définition 4.1. *Fonction de répartition des légions*

On appelle fonction de répartition des légions d'un graphe G une fonction $f : V \rightarrow \{0, 1, 2\}$.

Définition 4.2. *Sommet sécurisé / sommet défendu*

Un sommet $v \in V$ est sécurisé si $f(v) \geq 1$, et non sécurisé sinon. De manière similaire, un sommet $v \in V$ est défendu s'il existe $u \in N[v]$ tel que $f(u) \geq 1$.

Définition 4.3. *Fonction de domination romaine faible*

La fonction f est une fonction de domination romaine faible (abrégée en wrd-fonction) si f est une fonction de répartition des légions, s'il n'y a aucun sommet non-défendu par f , et si pour tout sommet $v \in V$ tel que $f(v) = 0$ il existe un sommet sécurisé $u \in N(v)$ tel que la fonction $f_{u \rightarrow v}$ définie par :

$$f_{u \rightarrow v}(x) = \begin{cases} 1 & \text{si } x = v \\ f(u) - 1 & \text{si } x = u \\ f(x) & \text{si } x \notin \{u, v\} \end{cases}$$

n'a pas de sommet non défendu. La fonction $f_{u \rightarrow v}$ désigne alors la fonction de répartition des légions obtenue en déplaçant une légion de u vers v .

Étant donnée une fonction de répartition des légions f , V_f^1 , V_f^2 désignent les ensembles $\{v \in V : f(v) = 1\}$ et $\{v \in V : f(v) = 2\}$. Le coût de f est alors défini par $\text{cout}(f) = \sum_{v \in V} f(v) = |V_f^1| + 2 \cdot |V_f^2|$. On note V_f l'union de ces deux ensembles. Lorsque f est une wrd-fonction, l'ensemble V_f est un ensemble dominant (qui n'est pas forcément minimal) de G .

Définition 4.4. *Sommets défendus en toute sécurité / Sommets faiblement défendus.*

On distingue maintenant deux types de sommets défendus : les sommets qui sont défendus en toute sécurité, et les sommets faiblement défendus.

Soit $v \in V$ un sommet et f une fonction de répartition des légions. On dit que v est **défendu en toute sécurité** par f si l'une des conditions suivante est respectée :

- v est sécurisé (c'est-à-dire que $f(v) > 1$) ;
- il existe $u \in N(v)$ tel que $f(u) = 2$;
- il existe $u \in N(v)$ tel que $f(u) = 1$ et les sommets non-défendus par $f_{u \rightarrow v}$ sont les mêmes que ceux non-défendus par f ($f_{u \rightarrow v}$ ne crée aucun sommet non défendu).

Dans le cas contraire on dit que v n'est pas défendu en toute sécurité.

Notons qu'une fonction de répartition des légions est une wrd-fonction si tout sommet $v \in V$ est défendu en toute sécurité par f .

Observons que pour tout sommet v qui n'est pas défendu en toute sécurité on a $f(v) = 0$, pour tout voisin sécurisé u de v on a $f(u) = 1$, et la fonction de répartition des légions $f_{u \rightarrow v}$ précédemment définie contient parmi les voisins de u autre que v un sommet non défendu w . Par la suite nous dirons qu'un tel sommet w est **faiblement défendu par u** , **faiblement défendu à cause de v** , ou simplement **faiblement défendu** lorsqu'il n'y a pas d'ambiguïté.

Nous illustrons la notion de sommet faiblement défendu dans la figure 4.1, et nous donnons dans la figure 4.2 un exemple de wrd-fonction.

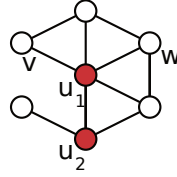


FIGURE 4.1 – Un graphe et une fonction de répartition des légions f définie par $f(x) = 1$ si $x \in \{u_1, u_2\}$ et $f(x) = 0$ sinon (chaque sommet rouge contient une légion). v étant faiblement défendu à cause de w , f n'est donc pas une wrd-fonction.

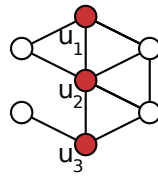


FIGURE 4.2 – Un graphe et une wrd-fonction f définie par $f(x) = 1$ si $x \in \{u_1, u_2, u_3\}$ et $f(x) = 0$ sinon (chaque sommet rouge contient une légion). Tout sommet est défendu en toute sécurité par f .

4.2 Domination Romaine Faible dans le cas général

Dans cette section nous nous intéressons à la recherche d'une fonction de domination romaine faible dans les graphes en général. Ce travail est issu de l'article « Exact Algorithms for Weak Roman Domination » de *Chapelle et al.* [CCC⁺13].

Chapelle et al. donnent plusieurs propriétés structurelles « clés » des fonctions de domination romaine faible dans les graphes en général.

Définition 4.5. Fonction caractéristique

Soit $V' \subseteq V$ un sous-ensemble de sommets d'un graphe G . Notons $\chi^{V'}$ la fonction caractéristique définie par :

$$\chi^{V'}(x) = \begin{cases} 1 & \text{si } x \in V'; \\ 0 & \text{sinon.} \end{cases}$$

Propriété 4.6.

Soit f une wrd-fonction d'un graphe G de coût minimum, et V_f l'ensemble de sommets associé. L'en-

semble V_f^2 est alors un ensemble dominant minimum des sommets qui ne sont pas défendus en toute sécurité par χ^{V_f} .

Ils montrent ensuite comment construire une wrd-fonction à partir d'un ensemble dominant.

Propriété 4.7.

Étant donné un ensemble dominant V' d'un graphe G , une wrd-fonction peut être obtenue en calculant un ensemble dominant S des sommets n'étant pas défendus en toute sécurité par $\chi^{V'}$. Cette wrd-fonction est définie par :

$$f(x) = \begin{cases} 2 & \text{si } x \in (V' \cap S) \\ 1 & \text{si } x \in (V' \cup S) \setminus (V' \cap S) \\ 0 & \text{sinon} \end{cases}$$

Ils établissent finalement le lemme suivant :

Lemme 4.8.

Soit $V'_1 \subseteq V$ un ensemble dominant d'un graphe G et S_1 un ensemble dominant minimum des sommets n'étant pas défendus en toute sécurité par $\chi^{V'_1}$. En supposant que $S_1 \not\subseteq V'_1$, il existe un super-ensemble $V'_2 \supset V'_1$ tel que pour tout ensemble dominant minimum S_2 de l'ensemble des sommets n'étant pas défendus en toute sécurité par $\chi^{V'_2}$, on a $\text{cout}(f_2) \leq \text{cout}(f_1)$, la fonction f_i , ($i \in \{1, 2\}$) étant la fonction de répartition des légions définie par :

$$f_i(x) = \begin{cases} 2 & \text{si } x \in (V'_i \cap S_i); \\ 1 & \text{si } x \in (V'_i \cup S_i) \setminus (V'_i \cap S_i); \\ 0 & \text{sinon.} \end{cases}$$

Ils montrent que l'on peut résoudre **DOMINATION ROMAINE FAIBLE MINIMUM** en temps et en espace $\mathcal{O}^*(2^n)$. Pour ce faire, ils donnent un algorithme fonctionnant en deux étapes. La première est une étape de pré-traitement utilisant une programmation dynamique inspirée de [Lie08] qui calcule un ensemble dominant minimum pour chaque sous-ensemble de sommets de G . Cette étape de pré-traitement nécessite un espace exponentiel. La seconde étape calcule une wrd-fonction à partir des ensembles dominants calculés à l'étape de pré-traitement, et est basée sur la propriété 4.7 ainsi que sur le lemme 4.8.

Ils établissent le théorème suivant :

Théorème 4.9.

Une fonction de domination romaine faible de tout graphe à n sommets peut être calculée en temps $\mathcal{O}^(2^n)$ et en espace exponentiel.*

Le second algorithme peut être vu comme une version modifiée du premier, effectue l'étape de pré-traitement en espace polynomial en utilisant l'algorithme exact exponentiel de *van Rooij* [vR11] pour résoudre **ENSEMBLE DOMINANT BLEU ET ROUGE MINIMUM** en temps $\mathcal{O}^*(1.2279^{|R|+|B|})$, pour tout graphe $G = (R \cup B, E)$.

Ils déduisent de cet algorithme le théorème suivant :

Théorème 4.10.

Une fonction de domination romaine faible de tout graphe à n sommets peut être calculée en temps $\mathcal{O}^(2.2279^n)$ et en espace polynomial.*

4.3 Domination Romaine Faible sur les graphes d'intervalles

On s'intéresse maintenant à la recherche d'une fonction de domination romaine faible de coût minimum lorsque le graphe en entrée est un graphe d'intervalles. Nous présentons à cette section un algorithme exact linéaire en la taille de l'entrée afin de résoudre ce problème. À la section 4.3.2 nous donnons l'analyse du temps d'exécution de cet algorithme.

4.3.1 Description de l'algorithme

L'algorithme que nous allons maintenant décrire est un algorithme « glouton », plus sophistiqué que l'algorithme **EnumDomSetInterval** décrit en section 3.4 du chapitre 3 énumérant les ensembles dominants minimaux dans les graphes d'intervalles. Cet algorithme construit un couple d'ensembles de sommets (V_1, V_2) , V_1 étant l'ensemble des sommets ayant une légion, et V_2 étant l'ensemble des sommets ayant deux légions.

A chaque étape l'algorithme ajoute un sommet à V_1 ou déplace un sommet de V_1 vers V_2 . Étant donnée une étape dans l'algorithme, celui-ci aura besoin d'un ensemble d'informations concernant la configuration de l'instance à l'étape précédente de l'algorithme, un système de *marquage* que nous décrirons par la suite.

Finalement nous déduirons du couple (V_1, V_2) retourné par l'algorithme une wrd-fonction f de coût minimum et définie par :

$$f(x) = \begin{cases} 2 & \text{si } x \in V_2 \\ 1 & \text{si } x \in V_1 \\ 0 & \text{sinon} \end{cases}$$

Soit $G = (V, E)$ un graphe à n sommets. Pour tout $v \in V$, on note par $l(v)$ (respectivement $r(v)$) l'extrémité gauche (respectivement l'extrémité droite) de l'intervalle correspondant à v dans le modèle. Nous supposons dans la suite que $G = (V, E)$ est donné avec son modèle normalisé $\mathcal{I}$, c'est-à-dire toutes les extrémités ont des valeurs distinctes dans $\{1, 2, \dots, 2n\}$. Pour plus de précisions concernant les graphes d'intervalles le lecteur pourra se référer à la section 1.1 du chapitre 1.

Étant donné un entier $d \in \{1, \dots, 2n\}$, on définit $\mathcal{D}_{>d} = \{\delta \in V : l(\delta) > d\}$ et soit $G_{\leq d}$ le graphe induit par l'ensemble d'intervalles $\{\delta \in V : l(\delta) \leq d\}$. Observons que $G_{\leq d}$ est le graphe $G[V \setminus \mathcal{D}_{>d}]$. Étant donnés deux ensembles V_1, V_2 , on note $r(V_1, V_2) = \max\{r(\delta) : \delta \in V_1 \cup V_2\}$. Pour $d, d' \in \{1, \dots, 2n\}$, $mark \in \{\star, -\}$ et $V_1, V_2 \subseteq V$, le quintuplet $(d, d', mark, V_1, V_2)$ est un *bon quintuplet* si chacune des propriétés suivantes sont remplies :

- (P1) (V_1, V_2) est un ensemble de domination romaine faible de coût minimum de $G_{\leq d}$ tel que $r(V_1, V_2)$ est maximum ;
- (P2) (V_1, V_2) n'est pas un ensemble de domination romaine faible de $G_{\leq d} \cup \{x\}$ pour tout $x \in \mathcal{D}_{>d}$;
- (P3) $r(V_1, V_2) = d'$ (ou $r(V_1, V_2) = 0$ dans le cas où $V_1 \cup V_2 = \emptyset$) ;
- (P4) si $mark = \star$, chaque sommet de $G_{\leq d} \setminus (V_1, V_2)$ doit avoir au moins un voisin dans $(V_1 \cup V_2) \setminus \{\kappa\}$, κ étant le sommet dans V_1 tel que $r(\kappa) = d'$.

On définit le *coût* d'un quintuplet $t = (d, d', mark, V_1, V_2)$ par $cout(t) = cout(V_1, V_2) = |V_1| + 2 \cdot |V_2|$. On note $\mathcal{T}$ l'ensemble des bons quintuplets (les quintuplets qui satisfont ces quatre propriétés).

Remarque. (à propos de la propriété (P4)). Notons que si κ est le sommet tel que $r(\kappa) = d'$ mais que $\kappa \in V_2$, alors on a nécessairement $mark = -$. Supposons qu'il existe un quintuplet $t = (d, d', mark, V_1, V_2)$ qui est un bon quintuplet tel que $mark = \star$ et tel que chaque sommet de $G_{\leq d} \setminus (V_1 \cup V_2)$ a au moins un voisin dans $(V_1 \cup V_2) \setminus \{\kappa\}$, avec κ un sommet tel que $r(\kappa) = d'$. Supposons que $\kappa \in V_2$. On observe alors que $t' = (d, d', mark, V_1 \cup \{\kappa\}, V_2 \setminus \{\kappa\})$ est un bon ensemble de domination romaine faible partiel satisfaisant les quatre propriétés [P1]-[P4], et $cout(t') < cout(t)$, ce qui contredit la propriété [P1] du quintuplet t .

Étant donnés deux quintuplets de $\mathcal{T}$, $t = (d, d', mark, V_1, V_2)$ et $\tilde{t} = (\tilde{d}, \tilde{d}', \widetilde{mark}, \widetilde{V_1}, \widetilde{V_2})$, on écrit $t < \tilde{t}$ lorsque $d < \tilde{d}$. Nous sommes maintenant prêts à présenter l'algorithme glouton calculant un ensemble de domination romaine faible d'un graphe d'intervalles. Il calcule itérativement les quintuplets $t_0, t_1, \dots, t_k$ tels que $t_0 < t_1 < \dots < t_k$ (Observons que cette suite est finie, la valeur de la première coordonnée augmentant mais étant inférieure à $2n$). L'algorithme commence par le quintuplet initial $t_0 = (0, 0, -, \emptyset, \emptyset)$ qui est trivialement un bon quintuplet. Puis il étend itérativement le quintuplet courant (correspondant à un bon quintuplet) jusqu'à ce qu'un ensemble de

domination romaine faible (de coût minimum) du graphe entier ait été calculé. Nous décrivons maintenant cet algorithme :

Algorithme 3 : DominationRomaineFaible($G = (V, E), \mathcal{I}$)

Entrées : Un graphe d'intervalles $G = (V, E)$ et son modèle d'intervalles $\mathcal{I}$.

Sorties : Un WRD-ensemble (V_1, V_2) de coût minimum de G .

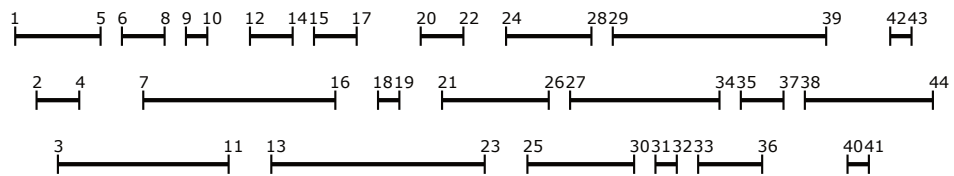
```

1 Soit  $(d, d', mark, V_1, V_2) \leftarrow (0, 0, -, \emptyset, \emptyset)$ 
2 tant que  $\mathcal{D}_{>d}$  n'est pas vide faire
3   Soit  $x \in \mathcal{D}_{>d}$  tel que  $r(x)$  est minimum
4   Soit  $y \in N[x]$  tel que  $r(y)$  est maximum
5   si  $r(y) = d'$  alors
6      $(d, d', mark, V_1, V_2) \leftarrow (r(y), r(y), -, V_1 \setminus \{y\}, V_2 \cup \{y\})$ 
7   sinon
8     si  $mark = \star$  alors
9       Soit  $x'$  tel que  $x' \neq y$ ,  $l(x') > d'$  et  $r(x')$  est minimum
10      si il n'existe pas de tel  $x'$  ou  $r(x') \geq r(y)$  alors  $\bar{d} \leftarrow r(y)$ 
11      sinon  $\bar{d} \leftarrow r(x')$ 
12    sinon si  $r(x) < d'$  alors  $\bar{d} \leftarrow d'$ 
13    sinon  $\bar{d} \leftarrow r(x)$ 
14    si il n'existe pas de  $\lambda$  tel que  $\lambda \neq y$  et  $d' < l(\lambda) \leq \bar{d}$  alors  $\overline{mark} \leftarrow \star$ 
15    sinon  $\overline{mark} \leftarrow -$ 
16     $(d, d', mark, V_1, V_2) \leftarrow (\bar{d}, r(y), \overline{mark}, V_1 \cup \{y\}, V_2)$ 
17 retourner la solution optimale  $(V_1, V_2)$ 

```

Afin de mieux comprendre cet algorithme, nous donnons maintenant son exécution sur un exemple. Soit la collection d'intervalles $\mathcal{I}$ suivante :

Les quintuplets successivement générés par notre algorithme sont les quintuplets suivants :



$(0, 0, -, \emptyset, \emptyset);$

```

    le quintuplet initial;
(4, 11, -, {(3, 11)}, ∅);
    (3, 11) est ajouté à  $V_1$ ;
(11, 16, ★, {(3, 11), (7, 16)}, ∅);
     $mark = ★$  car il n'existe pas de  $\lambda$  tel que  $11 < l(\lambda) \leq 11$ ;
(19, 23, -, {(3, 11), (7, 16), (13, 23)}, ∅);
(23, 26, ★, {(3, 11), (7, 16), (13, 23), (21, 26)}, ∅);
(32, 34, -, {(3, 11), (7, 16), (13, 23), (21, 26), (27, 34)}, ∅);
(36, 39, -, {(3, 11), (7, 16), (13, 23), (21, 26), (27, 34), (29, 39)}, ∅);
(41, 44, -, {(3, 11), (7, 16), (13, 23), (21, 26), (27, 34), (29, 39), (38, 44)}, ∅);
(44, 44, -, {(3, 11), (7, 16), (13, 23), (21, 26), (27, 34), (29, 39)}, {(38, 44)});

```

À l'issue du dernier quintuplet généré par notre algorithme, on en déduit la fonction de domination romaine faible f de coût minimum pour le graphe associé à la collection d'intervalles $\mathcal{I}$ suivante :

$$f(x) = \begin{cases} 1 & \text{si } x \in \{v_{(3,11)}, v_{(7,16)}, v_{(13,23)}, v_{(21,26)}, v_{(27,34)}, v_{(29,39)}\} \\ 2 & \text{si } x = v_{(38,44)} \\ 0 & \text{sinon} \end{cases}$$

avec $v_{(i,j)}$ l'intervalle v tel que $l(v) = i$ et $r(v) = j$;

Afin de montrer la correction de cet algorithme, on montre l'invariant de boucle suivant :

Lemme 4.11.

A chaque itération de la boucle « tant que » de l'algorithme **DominationRomaineFaible**, le quintuplet $(d, d', mark, V_1, V_2) \in \mathcal{T}$ est un bon ensemble partiel et satisfait donc les propriétés [P1]-[P4].

Démonstration.

La preuve est obtenue par récurrence sur les quintuplets produits par l'algorithme. Soit $t_0 = (0, 0, -, \emptyset, \emptyset)$ le quintuplet initialement créé dans l'étape d'initialisation de l'algorithme. Les quatre propriétés [P1]-[P4] sont clairement remplies pour ce quintuplet, étant donné que $V_1 \cup V_2 = \emptyset$. Donc t_0 appartient à $\mathcal{T}$.

Supposons maintenant par induction que $t = (d, d', mark, V_1, V_2)$ est un quintuplet de $\mathcal{T}$ ayant été calculé à une certaine itération de la boucle « tant que » et notons $\tilde{t} = (\tilde{d}, \tilde{d}', \tilde{mark}, \tilde{V}_1, \tilde{V}_2)$ le quintuplet construit par l'algorithme **DominationRomaineFaible** durant la prochaine itération de la boucle « tant que ». Soient x et y les sommets tels que décrits aux lignes 3 et 4 de l'algorithme **DominationRomaineFaible**, c'est-à-dire soit $x \in \mathcal{D}_{>d}$ tel que $r(x)$ est minimum et soit $y \in N[x]$ tel que $r(y)$ est maximum. Étant donné que $x \in \mathcal{D}_{>d}$, on a $r(x) > d$. De plus, par construction de $\tilde{d}$ (dont la valeur est donnée par la ligne 6, 10, 11, 12 ou 13) on a $d < \tilde{d}$ et donc $t < \tilde{t}$. Étant donné que t satisfait [P1] et [P2], pour tout quintuplet t' tel que $t < t'$ on a nécessairement $cout(t') > cout(t)$.

En particulier, par construction de $\tilde{t}$ on a $\text{cout}(\tilde{t}) = \text{cout}(t) + 1$, ce qui est facilement vérifiable étant donné qu'à chaque itération de la boucle « tant que », soit on ajoute un sommet à V_1 , soit on déplace un sommet de V_1 vers V_2 .

Par conséquent, $(\tilde{V}_1, \tilde{V}_2)$ est un ensemble de domination romaine faible minimum de $G_{\leq \ell}$ pour un certain ℓ . Nous montrons dans la suite que $\ell = \tilde{d}$. Par construction de $\tilde{d}'$ on a $\tilde{d}' = r(y)$ (observons que la valeur de $\tilde{d}'$ est donnée soit par la *ligne 6*, soit par la *ligne 16* dans l'algorithme). Étant donné que $\{y\} = (\tilde{V}_1 \setminus V_1) \cup (\tilde{V}_2 \setminus V_2)$ (c'est-à-dire que y est le nouveau sommet ajouté à V_1 par la *ligne 16* ou le sommet ajouté à V_2 par la *ligne 6*), on a $\tilde{d}' \geq d' = r(V_1, V_2)$, et la propriété **[P3]** est donc satisfaite par $\tilde{t}$. Dans la suite de la preuve, on distingue plusieurs cas afin de prouver que $\tilde{t}$ satisfait bien les autres propriétés.

- Premier cas : « y est ajouté à V_2 » .

Supposons tout d'abord que $r(y) = d'$. Considérons le quintuplet $(\tilde{d}, \tilde{d}', \widetilde{\text{mark}}, \tilde{V}_1, \tilde{V}_2) = (r(y), r(y), -, V_1 \setminus \{y\}, V_2 \cup \{y\})$, comme décidé en *ligne 6* de l'algorithme. Nous montrons maintenant que ce quintuplet satisfait les propriétés **[P1]** et **[P2]**. Rappelons que par hypothèse d'induction $t \in \mathcal{T}$ et respecte donc les propriétés **[P1]**-**[P4]**. Étant donné que $x \in \mathcal{D}_{>d}$ et tel que $r(x)$ est minimum (*ligne 3*) et (V_1, V_2) respecte **[P2]**, x est donc le sommet non-défendu (par (V_1, V_2)) avec la plus petite extrémité droite. Le sommet y est choisi parmi $N[x]$ comme étant celui avec la plus grande extrémité droite (*ligne 4*) et déplacé de V_1 vers V_2 , car il était déjà dans V_1 *ligne 6*. Étant donné que y est ajouté à V_2 et (V_1, V_2) est un ensemble de domination romaine faible de $G_{\leq d}$ (par **[P1]**) il s'ensuit que $(\tilde{V}_1, \tilde{V}_2) = (V_1 \setminus \{y\}, V_2 \cup \{y\})$ est un ensemble de domination romaine faible de $G_{\leq r(y)}$ avec $r(\tilde{V}_1, \tilde{V}_2) = d' = r(y)$. Par le choix glouton de y (c'est-à-dire $y \in N[x]$ et $r(y)$ est maximum) il est clair qu'il n'existe pas d'ensemble de domination romaine faible $(\tilde{V}'_1, \tilde{V}'_2)$ de $G_{\leq r(y)}$ avec $r(\tilde{V}'_1, \tilde{V}'_2) > r(\tilde{V}_1, \tilde{V}_2)$ (en d'autres termes, tout ensemble de domination romaine faible $(\tilde{V}'_1, \tilde{V}'_2)$ de $G_{\leq r(y)}$ avec $r(\tilde{V}'_1, \tilde{V}'_2) > r(\tilde{V}_1, \tilde{V}_2)$ et $\text{cout}(\tilde{V}'_1, \tilde{V}'_2) = \text{cout}(\tilde{V}_1, \tilde{V}_2)$ soit contredit que la propriété **[P1]** est bien respectée par $(\tilde{V}_1, \tilde{V}_2)$ ou laisse x non-défendu).

A cette étape nous avons presque montré que **[P1]** est satisfaite par $(\tilde{V}_1, \tilde{V}_2)$; nous devons encore montrer que $(\tilde{V}_1, \tilde{V}_2)$ est de coût minimum. Observons que $(\tilde{V}_1, \tilde{V}_2)$ est un ensemble de domination romaine faible de $G_{\leq d}$ tel que $r(V_1, V_2)$ est maximum (par **[P1]**) et x n'est pas défendu par (V_1, V_2) (par **[P2]**). Tout ensemble de domination romaine faible $(\tilde{V}'_1, \tilde{V}'_2)$ pour lesquels x est défendu a un coût $\text{cout}(\tilde{V}'_1, \tilde{V}'_2) > \text{cout}(\tilde{V}_1, \tilde{V}_2)$. Étant donné que y a été déplacé de V_1 vers V_2 dans $(\tilde{V}_1, \tilde{V}_2)$ on a $\text{cout}(\tilde{V}_1, \tilde{V}_2) = \text{cout}(V_1, V_2) + 1$, ce qui prouve la propriété **[P1]**. La propriété **[P2]** est clairement satisfaite, aucun sommet de $\mathcal{D}_{>r(y)}$ n'ayant de voisin dans $(\tilde{V}_1, \tilde{V}_2)$ (rappelons que $r(\tilde{V}_1, \tilde{V}_2) = r(y)$). Étant donné que $\text{mark} = -$, il n'y a rien à prouver pour **[P4]**.

- Second cas : « y est ajouté à V_1 » .

Supposons maintenant que $r(y) \neq d'$ et considérons le quintuplet $(\tilde{d}, \tilde{d}', \widetilde{\text{mark}}, \tilde{V}_1, \tilde{V}_2) =$

$(\bar{d}, r(y), \overline{mark}, V_1 \cup \{y\}, V_2)$, avec $\bar{d}$ et $\overline{mark}$ tels que définis aux lignes 8 et 16 de l'algorithme. Notons $\tilde{x}$ et $\tilde{y}$ les deux sommets considérés par l'itération précédente de la boucle « tant que ». Dans la suite de la preuve nous notons f la fonction de domination romaine faible correspondant à (V_1, V_2) ; rappelons que par hypothèse d'induction $t = (d, d', mark, V_1, V_2) \in \mathcal{T}$ et donc t respecte les propriétés [P1]-[P4]. Nous distinguons trois cas dans la déclaration conditionnelle des lignes 8, 12 et 13 de l'algorithme définissant la valeur de $\bar{d}$:

- « si $mark = \star$ » (ligne 8).

Observons que lorsque l'algorithme ajoute un sommet à V_2 il fixe la valeur de $mark$ à $-$; $mark$ étant égal à $\star$ il s'ensuit que $\tilde{y}$ a été ajouté à V_1 dans l'itération précédente de la boucle « tant que », et $r(\tilde{y}) = d'$. La figure 4.3 illustre cette situation.

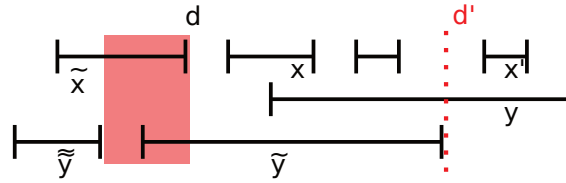


FIGURE 4.3 – Un ensemble d'intervalles correspondant à un quintuplet $(d, d', \star, V_1, V_2)$ avec $\{\tilde{y}, \tilde{y}\} \subseteq V_1$. Les sommets $\tilde{x}$ et $\tilde{y}$ sont les deux sommets considérés lors de la précédente itération de la boucle « tant-que », c'est-à-dire $\tilde{y}$ est le dernier sommet ajouté à V_1 . Le sommet $\tilde{y}$ est le sommet ajouté à V_1 juste avant $\tilde{y}$. Soit x , y et x' les sommets choisis par l'algorithme **DominationRomaineFaible** durant la prochaine itération de la boucle « tant que ». D'après la propriété [P4], il n'y a aucun sommet (éventuellement excepté $\tilde{y}$) dont l'extrémité gauche est dans la zone rouge.

D'après [P4], chaque sommet de $G_{\leq d} \setminus (V_1 \cup V_2)$ a au moins un voisin dans $(V_1 \cup V_2) \setminus \{\kappa\}$, κ étant le sommet de V_1 tel que $r(\kappa) = d'$, c'est-à-dire $\kappa = \tilde{y}$. Observons maintenant que chaque sommet $z \notin (V_1 \cup V_2 \cup \{y\})$ avec $d \leq l(z) \leq d'$ est adjacent à $\tilde{y}$ et à y . Étant donné que $mark = \star$, d'après [P4] chaque voisin w de $\tilde{y}$ avec $l(w) \leq d$ est aussi adjacent à un autre sommet de $V_1 \cup V_2$, ou appartient à $V_1 \cup V_2$. Donc z est défendu et $f_{\tilde{y} \rightarrow z}$ ne doit pas contenir un sommet non défendu dans $G_{\leq \bar{d}}$. Tout sommet v avec $d' < l(v) \leq r(x')$ est également défendu et $f_{y \rightarrow v}$ ne doit pas contenir de sommet non défendu dans $G_{\leq \bar{d}}$. Par conséquent, $(\tilde{V}_1, \tilde{V}_2) = (V_1 \cup \{y\}, V_2)$ est un ensemble de domination romaine faible de $G_{\leq \bar{d}}$, la valeur de $\bar{d}$ étant fixée à $\min(r(y), r(x'))$ par l'algorithme aux lignes 10 et 11 (en fonction de l'existence de x' à la ligne 9). Par le choix de y (qui est le voisin de x avec l'extrémité droite la plus grande), et pour les mêmes raisons que le cas précédent, il n'y a pas d'ensemble de domination romaine faible $(\tilde{V}_1', \tilde{V}_2')$ de $G_{\leq \bar{d}}$ avec $r(\tilde{V}_1', \tilde{V}_2') > r(\tilde{V}_1, \tilde{V}_2)$. Tout ensemble de domination romaine faible minimum $(\tilde{V}_1', \tilde{V}_2')$ pour lesquels les sommets de $G_{\leq \bar{d}} \setminus G_{\leq d}$ sont défendus ont donc un coût plus grand que $cout(V_1, V_2)$ (d'après

les propriétés [P1] et [P2] ce cout ne peut pas être égal à $\text{cout}(V_1, V_2)$). Étant donné que y a été ajouté à V_1 on a $\text{cout}(\widetilde{V}_1, \widetilde{V}_2) = \text{cout}(V_1, V_2) + 1$. On prouve donc que $(\widetilde{V}_1, \widetilde{V}_2)$ est un ensemble de domination romaine faible de coût minimum de $G_{\leq \bar{d}}$ avec $r(\widetilde{V}_1, \widetilde{V}_2)$ maximum, et la propriété [P1] est donc respectée. La propriété [P2] est aussi respectée, tout sommet v avec $l(v) > \bar{d}$ n'ayant soit aucun voisin dans $\widetilde{V}_1 \cup \widetilde{V}_2$ si $\bar{d} = r(y)$, soit la fonction $f_{\widetilde{y} \rightarrow v}$ laisserait x' non défendu si $\bar{d} = r(x')$.

- « si $r(x) = d'$ » (ligne 12). Admettons maintenant que $\text{mark} = -$. On observe que, par le choix de x et y (lignes 3 et 4), tout sommet w tel que $d < l(w) \leq d'$ est adjacent à $\widetilde{y}$ et à y . Soit $\bar{d} = d'$ (comme défini dans la ligne 12). La paire $(\widetilde{V}_1, \widetilde{V}_2) = (V_1 \cup \{y\}, V_2)$ est un ensemble de domination romaine faible de $G_{\leq \bar{d}}$, car tout sommet z avec $d < l(z) \leq d'$ est défendu et $f_{\widetilde{y} \rightarrow z}$ ne doit pas contenir de sommet non défendu dans $G_{\leq \bar{d}}$. Par le choix de y et d'après les mêmes arguments que précédemment sur le coût de la solution, $(\widetilde{V}_1, \widetilde{V}_2)$ respecte donc la propriété [P1]. La propriété [P2] est elle aussi respectée, étant donné que pour tout sommet v avec $l(v) > d'$ la fonction $f_{\widetilde{y} \rightarrow x}$ laisserait v non défendu (et par induction $f_{\widetilde{y} \rightarrow x}$ laisserait un sommet non défendu étant donné que $\text{mark} = -$).

- « sinon » (ligne 13).

Dans ce dernier cas, on fixe $\bar{d} = r(x)$ (à la ligne 13) et on montre que $(\widetilde{V}_1, \widetilde{V}_2) = (V_1 \cup \{y\}, V_2)$ est un ensemble de domination romaine faible minimum de $G_{\leq \bar{d}}$. En effet, on peut observer que l'ensemble des sommets z avec $d < l(z) \leq r(x)$ forme une clique (contenant y). C'est dû aux choix de x et de y aux lignes 3 et 4 : tout sommet z avec $d < l(z) \leq r(x)$ satisfait $l(z) \leq r(x) \leq r(z) \leq r(y)$. La fonction $f_{\widetilde{y} \rightarrow z}$ ne doit donc pas contenir de sommet non défendu dans $G_{\leq \bar{d}}$. Donc $(\widetilde{V}_1, \widetilde{V}_2) = (V_1 \cup \{y\}, V_2)$ est un ensemble de domination romaine faible de $G_{\leq \bar{d}}$. Puisque x est le sommet non-défendu (par (V_1, V_2)) dans $G_{\leq d}$ avec la plus petite extrémité droite, et par le choix de y (qui est choisi parmi $N[x]$ et tel que $r(y)$ est maximum), il n'y a pas d'ensemble de domination romaine faible $(\widetilde{V}'_1, \widetilde{V}'_2)$ de $G_{\leq \bar{d}}$ avec $r(\widetilde{V}'_1, \widetilde{V}'_2) > r(\widetilde{V}_1, \widetilde{V}_2)$. Comme précédemment, tout ensemble de domination romaine faible minimum $(\widetilde{V}'_1, \widetilde{V}'_2)$ pour lesquels x est défendu à un coût plus grand que $\text{cout}(V_1, V_2)$ (d'après les propriétés [P1] et [P2]). L'ajout de y à V_1 donne le coût $\text{cout}(\widetilde{V}_1, \widetilde{V}_2) = \text{cout}(V_1, V_2) + 1$. La propriété [P1] est donc remplie. Il est clair que [P2] est remplie car pour tout sommet v avec $l(v) > r(x)$ la fonction $f_{\widetilde{y} \rightarrow x}$ laisserait v non défendu (et par induction $f_{\widetilde{y} \rightarrow x}$ laisserait un sommet non défendu dans $G_{\leq d}$).

Observons finalement que dans tout ces cas, s'il n'existe pas de sommet λ tel que $\lambda \neq y$ et $d' < l(\lambda) \leq \bar{d}$ (on peut considérer que c'est vrai lorsque $d' = \bar{d}$), la propriété [P4] est remplie, et on fixe mark à $\star$, et à $-$ le cas échéant.

□

Lemme 4.12. *Après un nombre fini d'itérations, l'algorithme **DominationRomaineFaible** se termine par un quintuplet $(d, d', \text{mark}, V_1, V_2)$, où (V_1, V_2) est un ensemble de domination romaine faible minimum du graphe G en entrée.*

Démonstration. L'algorithme s'arrête dès lors qu'un quintuplet $t = (d, d', \text{mark}, V_1, V_2)$ a été calculé, tel que $\mathcal{D}_{>d}$ est vide. D'après le lemme 4.11, $t \in \mathcal{T}$ et (V_1, V_2) est un ensemble de domination romaine faible minimum du graphe G en entrée.

La valeur de la première coordonnée de chaque quintuplet augmentant mais étant plus petite que $2n$, on observe que ce quintuplet est obtenu après un nombre fini d'itérations. $\square$

Les lemmes 4.11 et 4.12 montrent la correction de l'algorithme **DominationRomaineFaible**. On établit maintenant le théorème suivant :

Théorème 4.13.

*L'algorithme **DominationRomaineFaible** calcule un ensemble de domination romaine faible (V_1, V_2) de coût minimum, lorsque le graphe donné en entrée est un graphe d'intervalles donné avec son modèle d'intersection. Cet algorithme est de complexité linéaire.*

L'analyse du temps d'exécution est donnée dans la section suivante.

4.3.2 Analyse du temps d'exécution

Le lemme 4.14 montre qu'un nombre linéaire de bons ensembles de domination romaine faible partiels sont calculés durant l'exécution de l'algorithme **DominationRomaineFaible**. Toutes les opérations peuvent clairement être exécutées en temps polynomial, et l'algorithme s'exécute donc en temps polynomial. Dans le reste de cette section, nous verrons que l'algorithme peut être implémenté de telle sorte que son temps d'exécution soit linéaire. Ce résultat est obtenu en utilisant un pré-traitement inspiré de celui donné dans [KLLP08], mais étendu pour nos besoins.

Lemme 4.14.

*Le nombre d'itérations de la boucle « tant que » dans l'algorithme **DominationRomaineFaible** est au plus n , c'est-à-dire le nombre d'intervalles du modèle d'intervalles $\mathcal{I}$ en entrée.*

Démonstration.

Observons que durant une itération de la boucle « tant que », le quintuplet $t = (d, d', \text{mark}, V_1, V_2)$ est remplacé par un nouveau quintuplet calculé $\tilde{t} = (\tilde{d}, \tilde{d}', \text{mark}, \tilde{V}_1, \tilde{V}_2)$ tels que $\tilde{d} > d$, c'est-à-dire qu'il respecte la relation $t < \tilde{t}$. De plus, ces valeurs de d et $\tilde{d}'$ correspondent aux extrémités droites de certains intervalles. Étant donné qu'il y a au plus n différentes valeurs dans $\{1, 2, \dots, 2n\}$ correspondant à une extrémité droite, ceci prouve le lemme. $\square$

Afin d'accélérer l'algorithme et d'obtenir un temps d'exécution linéaire, les opérations suivantes doivent être réalisées en temps constant durant chaque itération de la boucle « tant que » :

- étant donné $d \in \{0, 1, \dots, 2n\}$, trouver un sommet $x \in \mathcal{D}_{>d}$ tel que $r(x)$ est minimum ;
- étant donné un sommet x , trouver un sommet $y \in N[x]$ tel que $r(y)$ est maximum ;
- étant donné $d' \in \{0, 1, \dots, 2n\}$ et un sommet y donné, trouver un sommet x' tel que $x' \neq y, l(x') > d'$ et $r(x')$ est minimum ;
- étant donné $d', \bar{d} \in \{0, 1, \dots, 2n\}$ et un sommet y , déterminer s'il existe un sommet λ tel que $\lambda \neq y$ et $d' < l(\lambda) \leq \bar{d}$.

Nous décrivons un premier algorithme de pré-traitement, l'algorithme **Tri-PreTraitement** suivant :

Algorithme 4 : Tri-PreTraitement($G = (V, E), \mathcal{I}$)

Entrées : Un graphe d'intervalles et son modèle d'intervalles $\mathcal{I}$

Sorties : Les tableaux T'_L, T'_R, T_L, T_R nécessaires au pré-traitement

Soient T'_L, T'_R deux tableaux de taille $2n$ initialisés à NULL ;

Soient T_L, T_R deux tableaux de taille $2 \times n$;

pour chaque $x \in V$ **faire**

$T'_L[l(x)] \leftarrow x$;

$T'_R[r(x)] \leftarrow x$;

$index_L, index_R \leftarrow 1$;

pour $k = 1$ à $2n$ **faire**

si $T'_L[k] \neq \text{NULL}$ **alors**

$T_L[index_L] \leftarrow T'_L[k]$;

$index_L \leftarrow index_L + 1$;

si $T'_R[k] \neq \text{NULL}$ **alors**

$T_R[index_R] \leftarrow T'_R[k]$;

$index_R \leftarrow index_R + 1$;

retourner (T'_L, T'_R, T_L, T_R)

Étant donné un graphe d'intervalles (et son modèle d'intervalles $\mathcal{I}$) , l'algorithme **Tri-PreTraitement** ($G = (V, E)$) construit 4 tableaux T'_L, T'_R, T_L et T_R tels que :

- pour tout $i \in \{1, 2, \dots, 2n\}$, $T'_L[i]$ (respectivement $T'_R[i]$) contient l'indice de l'intervalle x tel que $l(x) = i$ (respectivement $r(x) = i$), si un tel intervalle existe, et contient NULL sinon ;

- les tableaux T_L et T_R de taille n contiennent les indices des n intervalles triés respectivement selon leur extrémité gauche et leur extrémité droite ;

Algorithme 5 : PreTraitement ($G = (V, E), \mathcal{I}$)

Entrées : Un graphe d'intervalles et son modèle d'intervalles $\mathcal{I}$

Sorties : Les tableaux MaxR , MinR et MinL

```

1  ( $T'_L, T'_R, T_L, T_R$ )  $\leftarrow$  Tri-PreTraitement( $G = (V, E), \mathcal{I}$ ) ;
2  Soit  $\text{MaxR}$  un tableau de taille  $n$  ;
3   $k, \text{index} \leftarrow n$  ;
4  tant que  $k \geq 1$  faire
5      si  $l(T_R[\text{index}]) \leq r(T_R[k])$  alors
6           $\text{MaxR}[T_R[k]] \leftarrow T_R[\text{index}]$  ;
7           $k \leftarrow k - 1$  ;
8      sinon
9           $\text{index} \leftarrow \text{index} - 1$  ;
10 Soit  $\text{MinR}$  un tableau de taille  $2 \times (2n + 1)$  initialisé à  $\text{NULL}$  ;
11  $\text{index} \leftarrow 0$  ;
12 pour  $k = 1$  à  $2n$  faire
13     si  $T'_R[k] \neq \text{NULL}$  alors
14         tant que  $\text{index} < l(T'_R[k])$  faire
15              $\text{MinR}[1][\text{index}] \leftarrow T'_R[k]$  ;
16              $\text{index} \leftarrow \text{index} + 1$  ;
17  $\text{index} \leftarrow 0$  ;
18 pour  $k = r(\text{MinR}[1][0]) + 1$  à  $2n$  faire
19     si  $T'_R[k] \neq \text{NULL}$  alors
20          $\text{prev} \leftarrow \text{MinR}[1][\text{index}]$  ;
21         tant que  $l(T'_R[k]) > \text{index}$  et  $\text{MinR}[1][\text{index}] = \text{prev}$  faire
22              $\text{MinR}[2][\text{index}] \leftarrow T'_R[k]$  ;
23              $\text{index} \leftarrow \text{index} + 1$  ;
24 Soit  $\text{MinL}$  un tableau de taille  $2 \times (2n + 1)$  initialisé à  $\text{NULL}$  ;
25  $\text{index} \leftarrow 1$  ;
26 pour  $k = 0$  à  $2n$  faire
27     tant que  $\text{index} \leq n$  et  $l(T_L[\text{index}]) \leq k$  faire  $\text{index} \leftarrow \text{index} + 1$  ;
28     si  $\text{index} \leq n$  alors  $\text{MinL}[1][k] \leftarrow T_L[\text{index}]$  ;
29     si  $\text{index} \leq n - 1$  alors  $\text{MinL}[2][k] \leftarrow T_L[\text{index} + 1]$  ;
30 retourner ( $\text{MaxR}, \text{MinR}, \text{MinL}$ )

```

Supposons que les intervalles de $\mathcal{I}$ soient arbitrairement indicés de 1 à n . On applique tout d'abord un pré-traitement en temps linéaire sur le modèle d'intervalles $\mathcal{I}$ de telle sorte que ces opérations puissent être réalisées en temps constant. Ce pré-traitement est décrit par l'algorithme **PreTraitement**, qui fait d'abord appel à une sous-routine **Tri-PreTraitement** triant les intervalles selon leur extrémité gauche et selon leur extrémité droite. L'algorithme **PreTraitement** retourne alors trois tableaux MaxR , MinR et MinL tels que :

- pour tout $x \in \{1, 2, \dots, n\}$, $\text{MaxR}[x]$ est égal à l'indice de l'intervalle y tel que $y \in N[x]$ et $r(y)$ est maximum ;
Ce tableau est calculé par la première boucle « tant que » de l'algorithme **PreTraitement** (lignes 2 à 9).
Les intervalles sont considérés selon un ordre particulier (par ordre décroissant selon leur extrémité droite, ce tri étant réalisé par l'algorithme **Tri-PreTraitement**).
On vérifie si l'intervalle x correspondant à l'indice $T_R[\text{index}]$ est adjacent à l'intervalle y correspondant à l'intervalle $T_R[k]$: par ordre de choix des intervalles à chaque étape on a $r(x) \leq r(y)$, par conséquent x et y sont adjacents si et seulement si $l(y) \leq r(x)$, ce que l'algorithme vérifie à la ligne 5.
La maximalité de y est garantie par l'ordre sur le choix des sommets ;
- pour tout $d \in \{0, 1, \dots, 2n\}$, $\text{MinR}[1][d]$ et $\text{MinR}[2][d]$ sont les indices des deux intervalles x et x' tels que $x, x' \in \mathcal{D}_{>d}$, $x \neq x'$ et $r(x), r(x')$ sont minimums (ces valeurs sont fixées à NULL si elles ne sont pas définies) ;
Ce tableau est calculé par la seconde boucle et la troisième boucle « tant que » de l'algorithme **PreTraitement** (lignes 10 à 23).
Les intervalles sont considérés selon un ordre particulier (par ordre croissant selon leur extrémité droite, ce tri étant réalisé par l'algorithme **Tri-PreTraitement**).
Le calcul de x est fait par les lignes 12 à 16. On considère l'intervalle avec la plus petite extrémité droite et on l'associe à toute valeur de d inférieure à son extrémité gauche. On considère ensuite le second intervalle et on répète la même opération pour tous les intervalles.
Le calcul de x' est fait par les lignes 17 à 23. Pour chaque valeur de d on va chercher l'indice du prochain intervalle qui fut affecté après x à une valeur de d durant la précédente boucle (celle des lignes 12 à 16). Cet intervalle x' est différent de x (on recherche le prochain intervalle différent de x), a son extrémité droite supérieure à d et est minimum, ce qui est garanti sur l'ordre dans lequel ces sommets sont considérés.
- pour tout $d' \in \{0, 1, \dots, 2n\}$, $\text{MinL}[1][d']$ et $\text{MinL}[2][d']$ sont les indices des deux intervalles λ et λ' tels que $\lambda, \lambda' \in \mathcal{D}_{>d'}$, $\lambda \neq \lambda'$ et $l(\lambda), l(\lambda')$ sont minimum (ces valeurs sont fixées à NULL si elles ne sont pas définies) ;
Ce tableau est calculé par la quatrième et dernière boucle « tant que » de l'algorithme **Pre-**

Traitement (lignes 24 à 29).

Les intervalles sont considérés selon l'ordre croissant selon leur extrémité gauche.

Le principe est ici très simple : pour toute valeur de d on affecte les indices des deux prochains intervalles (dans la liste des intervalles triés selon leur extrémité gauche), s'ils existent, dont l'extrémité gauche est supérieure à d . La minimalité des extrémités gauches est donc garantie par l'ordre dans lequel on choisit les intervalles.

Il est facile de voir que l'algorithme **PreTraitement** s'exécute en temps linéaire. Sa correction est simple et principalement technique (voir [KLLP08] pour un pré-traitement d'une idée similaire). Il reste à expliquer comment utiliser les tableaux MaxR , MinR et MinL dans l'algorithme **DominationRomaineFaible**. Les lignes 3 et 4 peuvent être remplacées par $x \leftarrow \text{MinR}[1][d]$ et $y \leftarrow \text{MaxR}[x]$. La ligne 9 peut être remplacée par

$$x' = \begin{cases} \text{MinR}[1][d'] & \text{si } \text{MinR}[1][d] \neq y ; \\ \text{MinR}[2][d'] & \text{sinon.} \end{cases}$$

Finalement, le test de la condition de la ligne 14 (c'est-à-dire qu'il n'existe pas de λ tel que $\lambda \neq y$ et $d' < l(\lambda) \leq \bar{d}$) peut être fait en évaluant la formule $(\text{MinL}[1][d'] = \text{NULL} \text{ ou } (\text{MinL}[1][d'] = y \text{ et } \text{MinL}[2][d'] = \text{NULL}) \text{ ou } (\text{MinL}[1][d'] = y \text{ et } l(\text{MinL}[2][d']) > \bar{d}) \text{ ou } (\text{MinL}[1][d'] \neq y \text{ et } l(\text{MinL}[1][d']) > \bar{d})$.

L'algorithme PreTraitement sur un exemple.

Nous considérons ici un exemple de graphe d'intervalles G , dont le modèle d'intervalles est défini comme suit :

- $\mathcal{I} = \{ v_1, v_2, v_3, v_4, v_5, v_6 \}$;
- $l(v_1) = 4 ; l(v_2) = 10 ; l(v_3) = 3 ; l(v_4) = 1 ; l(v_5) = 8 ; l(v_6) = 2 ;$
- $r(v_1) = 7 ; r(v_2) = 11 ; r(v_3) = 5 ; r(v_4) = 6 ; r(v_5) = 12 ; r(v_6) = 9 ;$

Nous donnons dans la figure 4.4 une représentation de $\mathcal{I}$.

Nous faisons référence dans les tableaux qui suivent aux *indices* des intervalles de $\mathcal{I}$. L'indice i fait donc référence à l'intervalle v_i pour tout $i \in \{1, \dots, 6\}$.

1. **Construction de MaxR.** Pour tout $i \in \{1, \dots, 6\}$, $\text{MaxR}[i]$ correspond à l'indice de l'intervalle v tel que $v \in N[v_i]$ et $r(v)$ est maximum. Dans notre exemple :

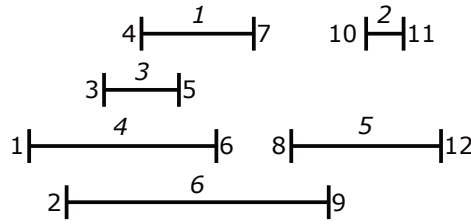


FIGURE 4.4 – Une représentation de $\mathcal{I}$. Pour chacun des intervalles, la valeur centrale correspond à l'indice de l'intervalle ; les valeurs aux extrémités représentent les valeurs de leur extrémité gauche et droite.

$$\begin{array}{rcl}
 i & = & 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \\
 \text{MaxR} & = & \begin{array}{|c|c|c|c|c|c|} \hline 6 & 5 & 6 & 6 & 5 & 5 \\ \hline \end{array}
 \end{array}$$

2. **Construction de MinR.** Pour tout $i \in \{0, \dots, 12\}$, $\text{MinR}[1][i]$ et $\text{MinR}[2][i]$ correspondent aux indices des deux intervalles v et v' tels que $l(v) > i$, $l(v') > d$, $r(v)$ et $r(v')$ sont minimums. Dans notre exemple :

$$\begin{array}{rcl}
 i & = & 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad 12 \\
 \text{MinR} & = & \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 3 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & \text{NULL} & \text{NULL} & \text{NULL} \\ \hline 3 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & \text{NULL} & \text{NULL} & \text{NULL} \\ \hline \end{array}
 \end{array}$$

Remarque. Par construction de MinR, pour tout $i \in \{0, \dots, 2n\}$ l'extrémité droite de l'intervalle indicé par $\text{MinR}[1][i]$ est inférieur à celui de l'intervalle indicé par $\text{MinR}[2][i]$.

3. **Construction de MinL.** Pour tout $i \in \{0, \dots, 12\}$, $\text{MinL}[1][i]$ et $\text{MinL}[2][i]$ correspondent aux indices des deux intervalles v et v' tels que $l(v) > i$, $l(v') > d$, $l(v)$ et $l(v')$ sont minimums. Dans notre exemple :

$$\begin{array}{rcl}
 i & = & 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \quad 10 \quad 11 \quad 12 \\
 \text{MinL} & = & \begin{array}{|c|c|c|c|c|c|c|c|c|c|c|c|c|} \hline 4 & 3 & 3 & 1 & 5 & 5 & 5 & 5 & 2 & 2 & \text{NULL} & \text{NULL} & \text{NULL} \\ \hline 6 & 6 & 1 & 5 & 2 & 2 & 2 & 2 & \text{NULL} & \text{NULL} & \text{NULL} & \text{NULL} & \text{NULL} \\ \hline \end{array}
 \end{array}$$

4.4 Conclusion

Nous nous sommes intéressés dans ce chapitre à la recherche, étant donné un graphe $G = (V, E)$, d'une fonction de pondération $f : V \rightarrow \{0, 1, 2\}$ des sommets de ce graphe, appelée *wrd*-

fonction, qui soit de coût minimum, et telle que chaque sommet de G soit défendu en toute sécurité. Un sommet v est défendu en toute sécurité si :

- soit $f(v) \geq 1$;
- soit il existe $u \in N(v)$ tel que $f(u) \geq 1$, et la wrd-fonction $f_{u \rightarrow v}$ définie par :

$$f_{u \rightarrow v}(x) = \begin{cases} 1 & \text{si } x = v \\ f(u) - 1 & \text{si } x = u \\ f(x) & \text{si } x \notin \{u, v\} \end{cases}$$

ne contient aucun sommet non-défendu (c'est-à-dire que pour tout $v \in V$ il existe $u \in N[v]$, possiblement $v = u$, tel que $f_{u \rightarrow v}(u) \geq 1$).

Le coût d'une telle fonction correspond à $\sum_{v \in V} f(v)$.

Nous avons rappelé en section 4.2 les résultats de *Chapelle et al.* [CCC⁺13], concernant certaines propriétés structurelles clés des wrd-fonctions, ainsi que sur la recherche de wrd-fonctions minimums dans les graphes en général.

Nous avons montré en section 4.3 que l'on peut résoudre **DOMINATION ROMAINE FAIBLE MINIMUM** en temps linéaire en la taille du graphe en entrée lorsque celui-ci est un graphe d'intervalles. Nous donnons tout d'abord un algorithme calculant une wrd-fonction de poids minimum demandant un temps polynomial, puis nous construisons un ensemble de structures de données dans une étape de pré-traitement inspirée de celle de [KLLP08] permettant de transformer notre algorithme en temps polynomial en algorithme en temps linéaire.

Comme nous l'avons dit en introduction, bien que ce problème ait été étudié par de nombreux aspects du point de vue structurel, son étude du point de vue de l'algorithme exact reste assez nouvelle ; la conception d'un algorithme utilisant le paradigme *Brancher et Réduire* pourrait permettre de calculer une wrd-fonction dans un temps plus raisonnable que ceux des algorithmes de *Chapelle et al.*

On peut voir le problème **DOMINATION ROMAINE FAIBLE MINIMUM** comme une version de **DOMINATION ROMAINE MINIMUM** dans laquelle nous aurions rajouté une certaine idée de dynamique. Néanmoins nous avons montré qu'une wrd-fonction est calculable en temps linéaire en la taille de l'entrée lorsque le graphe en entrée est un graphe d'intervalles. Il serait intéressant d'étudier ce problème pour d'autres classes de graphes pour lesquelles **DOMINATION ROMAINE MINIMUM** se résout en temps linéaire [KLLP08]. Si on considère par exemple la classe des graphes d'arcs circulaires, ou encore les graphes de permutation, est-il possible d'adapter l'approche décrite pour les graphes d'intervalles afin de l'appliquer à ces nouvelles classes ?

Dans les problèmes **DOMINATION SÉCURISÉE MINIMUM** et **DOMINATION ÉTERNELLE MINIMUM** qui sont deux problèmes proches de **DOMINATION ROMAINE FAIBLE MINIMUM**, l'idée de dynamisme est plus poussée que pour **DOMINATION ROMAINE FAIBLE MINIMUM**. Pour **DOMINATION ÉTERNELLE MINIMUM** par exemple, on cherche à ce que les sommets possédant une légion forment un ensemble dominant du graphe non pas juste après une attaque, comme c'est le cas pour **DOMINATION ROMAINE FAIBLE MINIMUM**, mais après chaque attaque étant donnée une succession d'attaques. Une continuité naturelle de notre étude serait donc de s'intéresser à ces nouveaux problèmes, et vérifier s'il est possible ou non d'adapter l'algorithme général de *Chapelle et al.*, ainsi que l'algorithme établi dans ce chapitre concernant les graphes d'intervalles.

5 Conclusion

Nous arrivons avec cette conclusion au terme de cette thèse. Nous rappelons ici les différents résultats présentés tout au long de ce manuscrit et nous proposons des pistes d'ouverture afin d'étendre nos résultats.

Le premier problème auquel nous nous sommes intéressés est le problème **ENSEMBLE CONNEXE TROPICAL MINIMUM** : étant donné un graphe $G = (V, E)$ dont les sommets sont colorés par une fonction de coloration $c : V \rightarrow \mathbb{N}$, un ensemble connexe tropical de G est un sous-ensemble de sommets $V' \subseteq V$ tel que $G[V']$ soit connexe et V' contienne au moins un sommet de chacune des couleurs de G . La problématique est alors de rechercher, étant donné un graphe coloré (G, c) un ensemble connexe tropical qui soit minimum. Ce problème est une variante du problème **MOTIF DE GRAPHE** défini par *Mc Morris et al.* dans [MWW94]. L'étude d'**ENSEMBLE CONNEXE TROPICAL** a été initiée par *Angles d'Auriac et al.* dans [AdCH⁺14], ils donnent en particulier des preuves d' NP -complétude pour ce problème.

En première partie du chapitre 2 nous nous sommes intéressés à ce problème dans le cas général. Nous avons alors donné un algorithme exact exponentiel calculant, étant donné un graphe en entrée, un ensemble connexe tropical minimum en temps $\mathcal{O}^*(1.5359^n)$. Cet algorithme est basé sur l'algorithme de *Nederlof* pour résoudre **ARBRE DE STEINER** [Ned12], sur l'algorithme de *Abu-Khzam et al.* pour résoudre **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE** [AKLM11] et sur un algorithme d'énumération en force brute.

Dans une seconde partie nous avons prouvé que, sous l'*Hypothèse du Temps Exponentiel* énoncée par *Implagliazzo et Paturi* [IP01], le problème **ENSEMBLE CONNEXE TROPICAL MINIMUM** n'admet pas d'algorithme sous-exponentiel même lorsque le graphe est un arbre de hauteur au plus 3. Ceci est montré à partir de la réduction d'**ENSEMBLE DOMINANT** vers **ENSEMBLE CONNEXE TROPICAL** établie par *Angles d'Auriac et al.* dans [AdCH⁺14] afin de montrer la NP -complétude de **ENSEMBLE CONNEXE TROPICAL**, ainsi qu'à partir de la preuve de non-existence d'un algorithme sous-exponentiel pour **ENSEMBLE DOMINANT** sous l'*ETH*, établie par *Fomin et al.* dans [FKW04].

L'étude de ce problème s'est achevée par la conception d'un algorithme exact exponentiel construisant un ensemble connexe tropical lorsque le graphe en entrée est un arbre, en temps $\mathcal{O}^*(1.2721^n)$ et en espace polynomial. Cet algorithme est constitué d'un ensemble de règles de branchement et de réduction.

Nous nous sommes ensuite intéressés à l'énumération des ensembles dominants minimaux d'un graphe. Un ensemble dominant d'un graphe $G = (V, E)$ est un sous-ensemble de sommets $V' \subseteq V$ tel que pour tout sommet $v \in V$, $N[v] \cap V' \neq \emptyset$. Le but est alors de rechercher, étant donné un graphe, tous les ensembles dominants de ce graphe qui soient minimaux par inclusion, c'est-à-dire ne contenant pas de sous-ensembles étant eux-mêmes des ensembles dominants. *Fomin et al.* montrent dans [FGPS08] que les graphes ont au plus 1.7159^n ensembles dominants minimaux et ont montré qu'il existe des graphes ayant au moins $15^{n/6} \approx 1.5704^n$ ensembles dominants minimaux. Bien qu'il y ait une différence importante entre la borne inférieure et la borne supérieure, diminuer cette différence semble être difficile, que ce soit en essayant de diminuer la borne supérieure ou en essayant de construire un graphe contenant plus de sommets que la borne inférieure actuelle. *Couturier et al.* se sont quant à eux intéressés à ce problème pour certaines classes de graphes, comme pour les graphes triangulés, splits ou encore pour les graphes d'intervalles propres [CHvHK13], ou pour les graphes cobipartis [CHvHK12].

Afin de clore certaines conjectures proposées par *Couturier et al.* et afin d'améliorer certains résultats précédents nous nous sommes aussi intéressés à ce problème dans certaines classes de graphes.

En première partie du chapitre 3 nous avons montré que les graphes splits ont $\mathcal{O}^*(3^{n/3}) = \mathcal{O}(1.4423^n)$ ensembles dominants minimaux. Pour ce faire, nous avons donné un algorithme d'énumération des ensembles dominants minimaux pour cette classe basé sur des règles de branchement, et nous avons montré qu'il est de complexité $\mathcal{O}^*(3^{n/3})$. Combiné au résultat établi par *Couturier et al.* dans [CHvHK13] et montrant qu'il existe des graphes splits avec $3^{n/3}$ ensembles dominants minimaux, nous en déduisons que notre borne est la meilleure possible.

Nous avons ensuite montré que les graphes cobipartis ont au plus $n^2 + 2 \cdot 1.4511^n$ ensembles dominants minimaux. Afin d'établir cette borne nous avons là encore construit un algorithme composé d'un ensemble de règles de branchements et de règles de réductions énumérant tous les ensembles dominants minimaux d'un tel graphe en temps $\mathcal{O}(1.4511^n)$. Ce résultat améliore le précédent résultat établi par *Couturier et al.* dans [CHvHK12].

Nous nous sommes finalement intéressés à la classe de graphes d'intervalles. Nous avons montré que de tels graphes admettent au plus $3^{n/3} \approx 1.4423^n$ ensembles dominants minimaux et donnons un algorithme, basé sur la programmation dynamique, énumérant les ensembles dominants dans les graphes d'intervalles en temps $\mathcal{O}^*(3^{n/3})$. Cette nouvelle borne améliore la borne précédente de $\mathcal{O}(1.4656^n)$ pour les graphes d'intervalles propres et la généralise aux graphes d'intervalles. Ce résultat, combiné avec la preuve de *Couturier et al.* montrant qu'il existe des graphes d'intervalles admettant $3^{n/3}$ ensembles dominants minimaux, assure que notre borne est la meilleure borne possible.

Dans le dernier chapitre de cette thèse nous nous sommes intéressés à un problème de domination, le problème **DOMINATION ROMAINE FAIBLE** : étant donné un graphe $G = (V, E)$ une fonction $f : V \rightarrow \{0, 1, 2\}$ est une fonction de domination romaine faible si pour tout sommet $v \in V$ tel que

$f(v) = 0$, v a un voisin u tel que $f(u) \geq 1$ et tel que l'ensemble $D = \{x \mid f_{u \rightarrow v}(x) \geq 1\}$ est un ensemble dominant de G , la fonction $f_{u \rightarrow v}$ étant définie par :

$$f_{u \rightarrow v}(x) = \begin{cases} 1 & \text{si } x = v ; \\ f(u) - 1 & \text{si } x = u ; \\ f(x) & \text{si } x \notin \{u, v\}. \end{cases}$$

Le coût de f est égal à $\sum_{v \in V} f(v)$. L'objectif de ce problème est donc, étant donné un graphe G , de construire une fonction de domination romaine faible pour G qui soit de coût minimum. Ce problème a été défini par *Henning et Hedetniemi* dans [HH03]. L'étude de ce problème du point de vue des algorithmes exacts a été initiée par *Chapelle et al.* dans [CCC⁺13], dans lequel ils donnent un algorithme exact exponentiel calculant, étant donné un graphe, une fonction de domination romaine faible minimum en temps $\mathcal{O}^*(2^n)$ et en espace exponentiel, et un second s'exécutant en temps $\mathcal{O}^*(2.2279^n)$ et en espace polynomial. Nous nous sommes intéressés dans le chapitre 4 au problème **DOMINATION ROMAINE FAIBLE** restreint aux graphes d'intervalles. Nous avons montré qu'une telle fonction de domination est calculable en temps linéaire pour cette classe de graphes.

Les ouvertures et objectifs de travail en continuité des études réalisées au cours de cette thèse sont multiples. Concernant le travail réalisé pour **ENSEMBLE CONNEXE TROPICAL** tout d'abord : nous avons obtenu un algorithme dans le cas général composé de réductions vers d'autres problèmes, le problème **ARBRE DE STEINER** ainsi que le problème **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE**. On a donc plusieurs possibilités afin d'améliorer notre algorithme. L'une consisterait à améliorer les réductions et/ou la résolution des problèmes **ARBRE DE STEINER** et/ou **ENSEMBLE DOMINANT CONNEXE BLEU ET ROUGE**. La deuxième, plus prometteuse, serait d'attaquer directement le problème initial en utilisant le paradigme *Brancher et Réduire*, combiné à une analyse du temps d'exécution du type *Mesurer pour Conquérir*, analyse que je n'ai malheureusement pas eu l'occasion d'utiliser durant cette thèse. Nous nous sommes ensuite intéressés à une classe de graphe, celle des arbres. De manière similaire aux autres problèmes étudiés durant cette thèse, nous pourrions tout à fait étudier ce problème sur d'autres classes de graphes, comme sur les graphes splits, pour lesquelles **ENSEMBLE CONNEXE TROPICAL** reste *NP*-complet.

Concernant le travail réalisé sur **ENSEMBLES DOMINANTS MINIMAUX**, notons que la question d'une estimation précise du nombre maximum d'ensembles dominants minimaux dans les graphes à n sommets est un problème ouvert majeur. Une continuité pour ce travail serait donc de s'intéresser aux classes de graphes pour lesquelles les bornes ne sont pas encore précises, ce qui est le cas entre autres des graphes triangulés, ou encore des arbres. Le résultat que nous avons donné dans cette thèse concernant les graphes cobipartis est encore améliorable, en concevant un algorithme plus sophistiqué par exemple (on peut imaginer un algorithme avec des règles de branchement et de réduction plus fines), ou en construisant un graphe cobiparti admettant plus d'ensembles dominants que la borne inférieure actuelle.

L'algorithme que nous avons réalisé pour résoudre **DOMINATION ROMAINE FAIBLE MINIMUM** est intéressant, dans le sens où ce n'est pas un algorithme trivial malgré sa nature « gloutonne », et cet algorithme nous permet d'obtenir une fonction de domination romaine faible optimale en temps linéaire. Cet algorithme exploite les bonnes propriétés structurelles des graphes d'intervalles. La question est double : peut-on concevoir des algorithmes permettant d'obtenir une fonction de domination romaine faible pour certaines superclasses de graphes de la classe des graphes d'intervalles notamment les graphes triangulés, ou encore modifier (ou s'inspirer) de notre algorithme afin de résoudre d'autres problèmes connexes, comme les problèmes **DOMINATION SÉCURISÉE MINIMUM** et **DOMINATION ÉTERNELLE MINIMUM** dont les définitions sont données en section 4.1.1 ? Une certaine notion de dynamique est en effet utilisée dans le problème **DOMINATION ROMAINE FAIBLE MINIMUM**, dans le sens où après un mouvement de légion chaque sommet du graphe doit rester sécurisé. L'idée dans **DOMINATION SÉCURISÉE MINIMUM** est similaire, sauf que le graphe doit rester défendu à chaque étape d'une séquence de mouvement de légions. Il serait donc intéressant de s'intéresser à ce problème lorsque le graphe en entrée est un graphe d'intervalles, voire dans le cas général. L'étude sur ce problème du point de vue des algorithmes exacts reste assez récente, et est constituée de l'article de *Chapelle et al.* [CCC⁺13] ainsi que de l'article duquel est issu le chapitre 4. On peut donc envisager de concevoir un algorithme plus efficace que le meilleur algorithme actuel pour les graphes en général, ainsi qu'une meilleure analyse de son temps d'exécution via une analyse du type *Mesurer pour Conquérir*. Finalement, on peut voir le problème **DOMINATION ROMAINE FAIBLE MINIMUM** comme une version plus « complexe » du problème **DOMINATION ROMAINE MINIMUM**. On pourrait donc regarder si **DOMINATION ROMAINE FAIBLE MINIMUM** reste difficile pour les classes de graphes pour lesquelles **DOMINATION ROMAINE MINIMUM** peut être résolu en temps polynomial.

Bibliographie

- [AdCH⁺14] J.-A. Angles d'Auriac, N. Cohen, A. Harutyunyan, S. Legay, H. Maftouhi, and Y. Manoussakis. Connected Tropical Subgraphs in Vertex-Colored Graphs. In *9th International Colloquium on Graph Theory and Combinatorics, ICGT '14*, Grenoble, France, 2014.
- [AKLM11] F. N. Abu-Khzam, M. Liedloff, and A. E. Mouawad. An Exact Algorithm for Connected Red-Blue Dominating Set. *Journal of Discrete Algorithms*, 9(3) :252–262, 2011.
- [Ber76] C. Berge. *Graphs and Hypergraphs*. Graphs and Hypergraphs. North-Holland Publishing Company, 1976.
- [BLS99] A. Brandstädt, V. B. Le, and J. P. Spinrad. *Graph Classes : A Survey*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.
- [BvBF⁺11] 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(5) :1296–1308, 2011.
- [BvR08] H. L. Bodlaender and J. M. M. van Rooij. Design by Measure and Conquer, A Faster Exact Algorithm for Dominating Set. In Pascal Weil Susanne Albers, editor, *STACS 2008*, pages 657–668, Bordeaux, France, 2008. IBFI Schloss Dagstuhl.
- [BvR11] H. L. Bodlaender and J. M. M. van Rooij. Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17) :2147–2164, 2011.
- [CCC⁺13] M. Chapelle, M. Cochefert, J.-F. Couturier, D. Kratsch, M. Liedloff, and A. Perez. Exact Algorithms for Weak Roman Domination. In *24th International Workshop On Combinatorial Algorithms, IWOCA '13*, pages 81–93, Rouen, France, 2013.
- [CCK⁺14] M. Chapelle, M. Cochefert, D. Kratsch, R. Letourneur, and M. Liedloff. Exact Exponential Algorithms to Find a Tropical Connected Set of Minimum Size. In M. Cygan and P. Heggernes, editors, *9th International Symposium on Parameterized and Exact Computation*, volume 8894 of *Lecture Notes in Computer Science*, pages 147–158, Wroclaw, Poland, 2014. Springer.

- [CDJHH04] E. J. Cockayne, P. A. Dreyer Jr, S. M. Hedetniemi, and S. T. Hedetniemi. Roman Domination in Graphs. *Discrete Mathematics*, 278(1-3) :11–22, 2004.
- [CF03] E. J. Cockayne and O. Favaron. Secure domination, weak roman domination and forbidden subgraphs. *Bulletin of the Institute of Combinatorics and its Applications*, 39, 2003.
- [CGG⁺05] E. J. Cockayne, P. J. P. Grobler, W. R. Gründlingh, J. Munganga, and J. H. van Vuuren. Protection of a Graph. *Utilitas Mathematica*, 67 :19–32, 2005.
- [CHvHK12] J.-F. Couturier, P. Heggernes, P. van’t Hof, and D. Kratsch. Minimal Dominating Sets in Graph Classes : Combinatorial Bounds and Enumeration. In *38th International Conference on Current Trends in Theory and Practice of Computer Science*, pages 202–213, 2012.
- [CHvHK13] 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 :82–94, 2013.
- [CKPW09] E. W. Chambers, W. B. Kinnersley, N. Prince, and D. B. West. Extremal Problems for Roman Domination. *SIAM J. Discrete Math.*, 23(3) :1575–1586, 2009.
- [CLL15] J.-F. Couturier, R. Letourneur, and M. Liedloff. On the Number of Minimal Dominating Sets on Some Graph Classes. *Journal of Theoretical Computer Sciences*, 562 :634–642, 2015.
- [Coo71] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC ’71, pages 151–158, New York, USA, 1971.
- [Cou90] B. Courcelle. The monadic second-order logic of graphs. recognizable sets of finite graphs. *Information and Computation*, 85(1) :12–75, 1990.
- [DFV11] R. Dondi, G. Fertin, and S. Vialette. Complexity issues in vertex-colored graph pattern matching. *Journal of Discrete Algorithms*, 9(1) :82–99, 2011.
- [DFVW99] R. G. Downey, M. R. Fellows, A. Vardy, and G. Whittle. The parametrized complexity of some fundamental problems in coding theory. *SIAM Journal of Computing*, 29(2) :545–570, 1999.
- [DH08] E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *The Computer Journal*, 51(3) :292–302, 2008.
- [Die12] R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.
- [Fer08] H. Fernau. Roman domination : a parameterized perspective. *International Journal of Computer Mathematics*, 85(1) :25–38, 2008.

Bibliographie

- [FFHV07] M. R. Fellows, G. Fertin, D. Hermelin, and S. Vialette. Sharp Tractability Borderlines for Finding Connected Motifs in Vertex-Colored Graphs. In L. Arge, C. Cachin, T. Jurdziński, and A. Tarlecki, editors, *Automata, Languages and Programming*, volume 4596 of *Lecture Notes in Computer Science*, pages 340–351. Springer Berlin Heidelberg, 2007.
- [FFHV11] 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(4) :799–811, 2011.
- [FGK05] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and Conquer : Domination – A Case Study. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 191–203. Springer Berlin Heidelberg, 2005.
- [FGK06] F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and Conquer : A Simple $O(2.0288^n)$ Independent Set Algorithm. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 18–25, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.
- [FGK09] F. V. Fomin, F. Grandoni, and D. Kratsch. A Measure & Conquer Approach for the Analysis of Exact Algorithms. *J. ACM*, 56(5), 2009.
- [FGK⁺13] F. V. Fomin, F. Grandoni, D. Kratsch, D. Lokshtanov, and S. Saurabh. Computing optimal steiner trees in polynomial space. *Algorithmica*, 65(3) :584–604, 2013.
- [FGPR08] F. V. Fomin, S. Gaspers, A. V. Pyatkin, and I. Razgon. On the Minimum Feedback Vertex Set Problem : Exact and Enumeration Algorithms. *Algorithmica*, 52(2) :293–307, 2008.
- [FGPS08] 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(1), 2008.
- [FHK⁺11] F. V. Fomin, P. Heggernes, D. Kratsch, C. Papadopoulos, and Y. Villanger. Enumerating Minimal Subset Feedback Vertex Sets. In *WADS*, pages 399–410, 2011.
- [FK10] F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 1st edition, 2010.
- [FKK⁺09] H. Fernau, J. Kneis, D. Kratsch, A. Langer, M. Liedloff, D. Raible, and P. Rossmanith. An exact algorithm for the maximum leaf spanning tree problem. In J. Chen and F. V. Fomin, editors, *Parameterized and Exact Computation*, volume 5917 of *Lecture Notes in Computer Science*, pages 161–172. Springer Berlin Heidelberg, 2009.

- [FKW04] F. V. Fomin, D. Kratsch, and G. J. Woeginger. Exact (Exponential) Algorithms for the Dominating Set Problem. In J. Hromkovič, M. Nagl, and B. Westfechtel, editors, *Graph-Theoretic Concepts in Computer Science*, number 3353 in Lecture Notes in Computer Science, pages 245–256. Springer Berlin Heidelberg, 2004.
- [FLS06] C. G. Fernandes, V. Lacroix, and M.-F. Sagot. Motif search in graphs : application to metabolic networks. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, 3(4) :360–368, 2006.
- [FT04] F. V. Fomin and D. M. Thilikos. A Simple and Fast Approach for Solving Problems on Planar Graphs. In V. Diekert and M. Habib, editors, *STACS 2004*, number 2996 in Lecture Notes in Computer Science, pages 56–67. Springer Berlin Heidelberg, 2004.
- [FV10] F. V. Fomin and Y. Villanger. Finding Induced Subgraphs via Minimal Triangulations. In *STACS*, pages 383–394, 2010.
- [GHH05] W. Goddard, S. M. Hedetniemi, and S. T. Hedetniemi. Eternal security in graphs. *J. Combin. Math. Combin. Comput*, 52 :169–180, 2005.
- [GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Freeman, W. H., 1979.
- [GK08] J. L. Goldwasser and W. Klostermeyer. Tight bounds for eternal dominating sets in graphs. *Discrete Mathematics*, 308(12) :2589–2593, 2008.
- [GM09] P. J. P. Grobler and C. M. Mynhardt. Secure domination critical graphs. *Discrete Mathematics*, 309(19) :5820–5827, 2009.
- [Gol04] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs : Second Edition*. Annals of Discrete Mathematics. Elsevier Science, 2004.
- [Gra04] F. Grandoni. *Exact Algorithms for Hard Graph Problems*. PhD thesis, Università di Roma Tor Vergata, Roma, Italy, 2004.
- [Gra06] F. Grandoni. A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms*, 4(2) :209 – 214, 2006.
- [GS10] S. Guillemot and F. Sikora. Finding and counting vertex-colored subtrees. In *Mathematical Foundations of Computer Science 2010*, pages 405–416. Springer, 2010.
- [HH03] S. T. Hedetniemi and M. A. Henning. Defending the Roman Empire—A new strategy. *Discrete Mathematics*, 266(1-3) :239–251, 2003.
- [HHS98a] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Domination in Graphs : Volume 2 : Advanced Topics*. CRC Press, New York, 1 edition edition, 1998.
- [HHS98b] T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of domination in graphs*. New York : Marcel Dekker, 1998.

Bibliographie

- [IKSS06] T. Ideker, R. M. Karp, J. Scott, and R. Sharan. Efficient algorithms for detecting signaling pathways in protein interaction networks. *Journal of Computational Biology*, 13(2) :133–144, 2006.
- [IP01] R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62 :367–375, 2001.
- [IPZ01] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4) :512–530, 2001.
- [Iwa11] Y. Iwata. A Faster Algorithm for Dominating Set Analyzed by the Potential Method. In D. Marx and P. Rossmanith, editors, *Parameterized and Exact Computation - 6th International Symposium, IPEC 2011, Saarbrücken, Germany, September 6-8, 2011. Revised Selected Papers*, volume 7112 of *Lecture Notes in Computer Science*, pages 41–54. Springer, 2011.
- [Kar72] R. M. Karp. Reducibility among Combinatorial Problems. In J. D. Bohlinger, R. E. Miller, and J. W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.
- [KLLP08] T. Kloks, M. Liedloff, J. Liu, and S.-L. Peng. Efficient algorithms for Roman domination on some classes of graphs. *Discrete Applied Mathematics*, 156(18) :3400–3415, 2008.
- [KLM⁺13] M. M. Kanté, V. Limouzy, A. Mary, L. Nourine, and T. Uno. On the Enumeration and Counting of Minimal Dominating sets in Interval and Permutation Graphs. In *ISAAC*, pages 339–349, 2013.
- [KLMN11] M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. Enumeration of Minimal Dominating Sets and Variants. In *FCT*, pages 298–309, 2011.
- [KLMN12] M. M. Kanté, V. Limouzy, A. Mary, and L. Nourine. On the Neighbourhood Helly of Some Graph Classes and Applications to the Enumeration of Minimal Dominating Sets. In *ISAAC*, pages 289–298, 2012.
- [Krz13] M. Krzywkowski. Trees having many minimal dominating sets. *Information Processing Letters*, 113(8) :276–279, 2013.
- [Lie08] M. Liedloff. Finding a dominating set on bipartite graphs. *Inf. Process. Lett.*, 107(5) :154–157, 2008.
- [LT80] R. Lipton and R. Tarjan. Applications of a Planar Separator Theorem. *SIAM Journal on Computing*, 9(3) :615–627, 1980.
- [LT08] Z. Lonc and M. Truszczynski. On the number of minimal transversals in 3-uniform hypergraphs. *Discrete Mathematics*, 308(16) :3668–3687, 2008.

- [MM65] J. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1) :23–28, 1965.
- [MMP11] T. N. M. Malini Mai and P. R. L. Pushpam. Weak Roman domination in graphs. *Discussiones Mathematicae Graph Theory*, 31(1) :161–170, 2011.
- [MWW94] F. McMorris, T. Warnow, and T. Wimer. Triangulating Vertex-Colored Graphs. *SIAM Journal on Discrete Mathematics*, 7(2) :296–306, 1994.
- [Ned12] J. Nederlof. Fast Polynomial-Space Algorithms Using Inclusion-Exclusion. *Algorithmica*, 65(4) :868–884, 2012.
- [NvDvR09] J. Nederlof, T. C. van Dijk, and J. M. M. van Rooij. Inclusion/exclusion meets measure and conquer. In A. Fiat and P. Sanders, editors, *Algorithms - ESA 2009*, volume 5757 of *Lecture Notes in Computer Science*, pages 554–565. Springer Berlin Heidelberg, 2009.
- [Pap07] C. H. Papadimitriou. *Computational complexity*. Academic Internet Publ., 2007.
- [RR00] C. S. ReVelle and K. E. Rosing. Defendens Imperium Romanum : A Classical Problem in Military Strategy. *American Mathematical Monthly*, 107(7), 2000.
- [Ste99] I. Stewart. Defend the Roman Empire ! *Scientific American*, 6(281) :136–139, 1999.
- [vR11] J. M. M. van Rooij. *Exact exponential-time algorithms for domination problems in graphs*. PhD thesis, Universiteit Utrecht, 2011.
- [Wes00] D. B. West. *Introduction to Graph Theory*. Pearson, Upper Saddle River, N.J, 2 edition edition, 2000.
- [Wol91] P. Wolper. *Introduction à la calculabilité*. Collection iia. InterEditions, 1991.

Algorithmes exacts et exponentiels pour des problèmes de graphes

Résumé :

De nombreux problèmes algorithmiques sont « difficiles », dans le sens où on ne sait pas les résoudre en temps polynomial par rapport à la taille de l'entrée, soit parce qu'ils sont NP-difficiles, soit, pour certains problèmes d'énumération, à cause du nombre exponentiel d'objets à énumérer.

Depuis une quinzaine d'années on trouve un intérêt grandissant dans la littérature pour la conception d'algorithmes exacts sophistiqués afin de les résoudre le plus efficacement possible. Dans le cadre de cette thèse, nous nous intéressons à la conception d'algorithmes exacts exponentiels autour de trois problèmes difficiles. Nous étudions tout d'abord le problème d'optimisation Ensemble Connexe Tropical pour lequel nous décrivons un algorithme afin de le résoudre en général, puis un algorithme de branchement plus rapide pour le résoudre sur les arbres, ce problème restant difficile même dans ce cas. Nous nous intéressons ensuite au problème d'énumération Ensembles Dominants Minimaux, pour lequel nous donnons des algorithmes résolvant ce problème dans les graphes splits, cobipartis, ainsi que dans les graphes d'intervalles. Nous déduisons des bornes supérieures sur le nombre d'ensembles dominants minimaux admis par de tels graphes. La dernière étude de cette thèse concerne le problème d'optimisation Domination Romaine Faible dans lequel, étant donné un graphe nous cherchons à construire une fonction de pondération selon certaines propriétés. Le problème est NP-difficile en général, mais nous donnons un algorithme glouton linéaire calculant une telle fonction pour les graphes d'intervalles.

Mots clés : algorithmes, algorithmes exacts exponentiels, graphes, ensembles dominants, domination romaine faible, ensemble connexe tropical.

Exact exponential algorithms for solving graph problems

Summary :

Many algorithmic problems are « hard », in the sense of we do not know how to solve them in polynomial time, either because they are NP-hard, or, for some enumeration problems, because the number of objects to be produced is exponential.

During the last fifteen years there was a growing interest in the design of exact algorithms to solve such problems as efficiently as possible. In the context of this thesis, we focus on the design of exponential exact algorithms for three hard problems. First, we study the optimisation problem Tropical Connected Set for which we describe an algorithm to solve it in the general case, then a faster branch-and-reduce algorithm to solve it on trees; the problem remains difficult even in this case. Secondly we focus on the Minimal Dominating Sets enumeration problem, for which we give algorithms to solve it on split, cobipartite and intervals graphs. As a byproduct, we establish upper bounds on the number of minimal dominating sets in such graphs. The last focus of this thesis concerns the Weak Roman Domination optimisation problem for which, given a graph, the goal is to build a weight function under some properties. The problem is NP-hard in general, but we give a linear greedy algorithm which computes such a function on interval graphs.

Keywords : algorithms, exact exponential algorithms, graphs, dominating sets, weak roman domination, tropical connected set.