Algorithmes exponentiels pour l'étiquetage, la domination et l'ordonnancement

Mathieu Liedloff

Centre Val de Loire UNIVERSITÉ
Spirit of the Loire Valley

UNIVERSITÉ D'ORLÉANS

UNIVERSITE D'ORLEANS

Laboratoire d'Informatique Fondamentale d'Orléans

Habilitation à Diriger des Recherches

présentée par :

Mathieu Liedloff

soutenue le 4 décembre 2015

Discipline/ Spécialité : informatique

Algorithmes exponentiels pour l'étiquetage, la domination et l'ordonnancement

RAPPORTEURS :

Jean CARDINAL	Professeur, Université Libre de Bruxelles
Michel HABIB	Professeur, Université Paris Diderot
Dimitrios M. THILIKOS	Directeur de recherche CNRS

JURY :

Jean-Charles BILLAUT	Professeur, Université F. Rabelais, Tours
Jean CARDINAL	Professeur, Université Libre de Bruxelles
Michel HABIB	Professeur, Université Paris Diderot
Iyad KANJ	Professeur, De Paul University, Chicago
Vangelis PASCHOS	Professeur, Université Paris Dauphine
Dimitrios M. THILIKOS	Directeur de recherche CNRS
Ioan TODINCA	Professeur, Université d'Orléans

Résumé

Ce manuscrit d'*Habilitation à Diriger des Recherches* met en lumière quelques résultats obtenus depuis ma thèse de doctorat soutenue en 2007. Ces résultats ont été, pour l'essentiel, publiés dans des conférences et des journaux internationaux. Des algorithmes exponentiels sont donnés pour résoudre des problèmes de décision, d'optimisation et d'énumération. On s'intéresse tout d'abord au problème d'étiquetage $L(2,1)$ d'un graphe, pour lequel différents algorithmes sont décrits (basés sur du *branchement*, le paradigme *diviser-pour-régner*, ou la *programmation dynamique*). Des bornes combinatoires, nécessaires à l'analyse de ces algorithmes, sont également établies. Dans un second temps, nous résolvons des problèmes autour de la domination. Nous développons des algorithmes pour résoudre une généralisation de la domination et nous donnons des algorithmes pour énumérer les ensembles dominants minimaux dans des classes de graphes. L'analyse de ces algorithmes implique des bornes combinatoires. Finalement, nous étendons notre champ d'applications de l'algorithmique modérément exponentielle à des problèmes d'ordonnancement. Par le développement d'approches de type *programmation dynamique* et la généralisation de la méthode *trier-et-chercher*, nous proposons la résolution de toute une famille de problèmes d'ordonnancement.

Mots clés

algorithmes, algorithmes modérément exponentiels, problèmes NP-complets, problèmes de graphes, problèmes d'ordonnancement

---

Abstract

This manuscript of *Habilitation à Diriger des Recherches* enlights some results obtained since my PhD, I defended in 2007. The presented results have been mainly published in international conferences and journals. Exponential-time algorithms are given to solve various decision, optimization and enumeration problems. First, we are interested in solving the $L(2,1)$-labeling problem for which several algorithms are described (based on *branching*, *divide-and-conquer* and *dynamic programming*). Some combinatorial bounds are also established to analyze those algorithms. Then we solve domination-like problems. We develop algorithms to solve a generalization of the dominating set problem and we give algorithms to enumerate minimal dominating sets in some graph classes. As a consequence, the analysis of these algorithms implies combinatorial bounds. Finally, we extend our field of applications of moderately exponential-time algorithms to scheduling problems. By using *dynamic programming* paradigm and by extending the *sort-and-search* approach, we are able to solve a family of scheduling problems.

Keywords

algorithms, moderately exponential-time algorithms, NP-complete problems, graphs problems, scheduling problems

Remerciements

J'adresse en premier lieu mes remerciements à Dimitrios Thilikos, Jean Cardinal et Michel Habib, les trois rapporteurs de cette habilitation. Ce sont trois scientifiques de premier plan et c'est un grand honneur qu'ils aient accepté de porter leur attention à mes travaux. Je remercie également les examinateurs : Vangelis Paschos, Iyad Kanj, Jean-Charles Billaut et Ioan Todinca. Je suis particulièrement honoré de votre présence dans mon jury. Je remercie mon *Doktorvater*, Dieter Kratsch qui verra en cette habilitation la concrétisation de quelques travaux initiés en thèse, il y a près de dix ans.

Les résultats présentés dans ce manuscrit sont issus de collaborations, avec des co-auteurs auprès desquels j'ai eu un grand plaisir à travailler : Jean-François Couturier, Petr A. Golovach, Frédéric Havet, Konstanty Junosza-Szaniawski, Martin Klazar, Jan Kratochvíl, Dieter Kratsch, Christophe Lenté, Romain Letourneur, Artem V. Pyatkin, Peter Rossmanith, Paweł Rzążewski, Lei Shang, Ameur Soukhal, Vincent T'Kindt, Ioan Todinca, Yngve Villanger. Merci pour tous nos échanges scientifiques et votre amitié. Bien évidement, mes remerciements s'adressent également à l'ensemble de mes co-auteurs et collègues, même si ce manuscrit ne permet pas de présenter l'intégralité de nos collaborations. Votre apport scientifique et personnel a été essentiel. J'ai beaucoup appris auprès de chacun de vous. J'espère que nos collaborations s'enrichirons encore davantage.

Mes remerciements vont également à mes collègues du Laboratoire d'Informatique Fondamentale d'Orléans qui, depuis mon recrutement comme maître de conférences, m'ont offert un cadre idéal pour mener des recherches et entretenir de riches échanges scientifiques. Je remercie également les collègues du Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier qui m'ont chaleureusement accueilli en délégation CNRS en 2015 (Alex, Benjamin, Christophe, Daniel, Dimitrios, Ignasi, Marin, Mickael, Pascal, Rodolphe, Stéphane). Cette période fût propice à la rédaction de ce manuscrit.

J'exprime des remerciements tout particuliers à mes deux « mentors », Dieter Kratsch et Ioan Todinca, avec qui j'ai exploré de nombreuses pistes et qui m'ont toujours consacré du temps. Je suis heureux de vous savoir bienveillants à mon égard et que vous soyez toujours prêt à m'apporter votre aide.

Je remercie tous les proches, amis et famille, notamment mes parents, pour leurs soutiens et encouragements. J'adresse un grand merci à Odile qui m'a accueilli et a assuré la logistique pendant ma délégation dans le sud. Je ne peux pas terminer ces éloges sans remercier ma mathématicienne préférée, à qui je passe le relais. Carine, c'est par toi que s'achèvent mes remerciements les plus chaleureux. Tu m'apportes beaucoup.

à tous, merci.

Table des matières

Liste des symboles

$\alpha\|\beta\|\gamma$	notation utilisée pour décrire un problème d'ordonnancement (voir les pages 16 et 137)
$\alpha(G)$	taille maximum d'un ensemble stable de G
$\chi(G)$	nombre minimum de couleurs nécessaires pour colorier G
$\Delta(G)$	degré maximum d'un graphe G
$\delta(G)$	degré minimum d'un graphe G
$\gamma(G)$	taille minimum d'un ensemble dominant d'un graphe G
$\Lambda_Q^P(G)$	largeur minimum d'un étiquetage L_Q^P d'un graphe G
$\lambda_{2,1}(G)$	largeur minimum d'un étiquetage $L(2,1)$ d'un graphe G
$[\![x, y]\!]$	ensemble des entiers de l'intervalle $[x, y]$
$\omega(G)$	taille maximum d'une clique de G
G^2	graphe obtenu d'un graphe G en ajoutant une arête entre chaque paire de sommets situés à distance 2
C_k	cycle composé de k sommets
$d_G(v)$	degré du sommet v dans un graphe G
$\mathrm{diam}(G)$	diamètre d'un graphe G
$\mathrm{dist}(u, v)$	distance minimale entre deux sommets u et v
$E(G)$	ensemble E des arêtes d'un graphe $G = (V, E)$
$\overline{G}$	graphe complémentaire du graphe G
$G[S]$	sous-graphe induit par les sommets de S
$G \cong H$	les graphes G et H sont isomorphes
K_k	graphe complet à k sommets
$\mathrm{length}(P)$	longueur, en nombre d'arêtes, d'un chemin P

$N_G(v)$ voisinage ouvert d'un sommet v dans un graphe G

$N_G[v]$ voisinage fermé d'un sommet v dans un graphe G

$N_X(v)$ intersection d'un ensemble X avec le voisinage d'un sommet v

$\mathcal{O}(\cdot)$ notation asymptotique définie à la page 8

$\mathcal{O}^*$ notation asymptotique qui supprime les facteurs polynomiaux

$\Omega(\cdot)$ notation asymptotique définie à la page 8

$\Theta(\cdot)$ notation asymptotique définie à la page 8

P_k chemin à k sommets

$\overline{P_k}$ graphe complémentaire du chemin P_k à k sommets

$\mathrm{pp}(n)$ nombre maximum de paires propres dans un graphe connexe à n sommets

$pw(G)$ largeur linéaire d'un graphe G

rb-graphe graphe (V, R, B) sur l'ensemble de sommets V dont les arêtes appartiennent soit à R (arêtes rouges) soit à B (arêtes noires) et tel que $\forall u, v, w \in V$, $\{v, w\} \in B \wedge \{v, u\} \in B \Rightarrow \{u, w\} \in R \cup B$.

$tw(G)$ largeur arborescente d'un graphe G

$V(G)$ ensemble V des sommets d'un graphe $G = (V, E)$

Introduction

Pour de nombreux problèmes, l'existence d'algorithmes polynomiaux pour les résoudre de façon exacte est improbable, sauf si P=NP. Face à ces problèmes particulièrement difficiles, plusieurs approches ont été développées pour les attaquer : algorithmes d'approximation, probabilistes, à paramètres fixés, exponentiels, heuristiques, restrictions à des cas particuliers. Typiquement, ces approches demandent de relaxer un ou plusieurs critères parmi l'**exactitude**, la **généralité** et l'**efficacité**. Par exemple, la restriction à des cas ayant suffisamment de structure offre la possibilité de résoudre exactement et efficacement ; le sacrifice se fait sur la généralité. L'utilisation d'un algorithme d'approximation relaxe cette fois-ci l'exactitude au profit des autres critères. Les algorithmes modérément exponentiels sacrifient l'efficacité. Il serait réducteur de penser que ces approches ne puissent être combinées : il est parfaitement possible de concevoir des algorithmes exponentiels pour résoudre de façon approchée un problème qui resterait difficile sur des entrées particulières. L'objectif de nos travaux est de faire avancer les connaissances sur l'algorithmique modérément exponentielle pour obtenir des méthodes de résolution dont le temps d'exécution, certes exponentiel, soit le plus faible possible. Pour cela, nous développons et mettons en oeuvre diverses techniques, à la fois de conception et d'analyse. Nous mettrons en lumière ces techniques par l'étude de problèmes de graphes et d'ordonnancement.

Les premiers algorithmes exponentiels pour résoudre des problèmes difficiles datent des années 60. Davis, Putnam, Logemann, Loveland donnent les premiers algorithmes de *branchement* pour SAT (Davis et Putnam, 1960; Davis *et al.*, 1962). Encore aujourd'hui, la popularité de ces algorithmes font qu'ils sont bien souvent cités par les noms de leurs auteurs. Dans la même décennie, Held et Karp (1962) donnent un algorithme de *programmation dynamique* en temps[1] $\mathcal{O}^*(2^n)$ pour le problème VOYAGEUR DE COMMERCE. Cet algorithme demeure le meilleur connu et la recherche d'un algorithme déterministe en temps $\mathcal{O}(c^n)$, avec $c < 2$, reste largement ouverte. Plus tard, Horowitz et Sahni (1974) prouvent que le problème SAC-À-DOS peut être résolu en temps $\mathcal{O}^*(2^{n/2})$ grâce à une technique appelée *trier-et-chercher*. Nous avons déjà évoqué trois techniques : *branchement*, *programmation dynamique* et *trier-et-chercher*. Selon le problème à traiter, l'une ou l'autre de ces techniques peut se révéler la plus pertinente. Une autre différence les caractérise : les deux dernières techniques nécessitent un espace exponentiel alors que le *branchement* peut « recycler l'espace » pour n'utiliser qu'un espace polynomial. Nous pouvons naturellement nous demander si, par exemple, VOYAGEUR DE COMMERCE pourrait être résolu en utilisant un espace polynomial, avec un temps d'exécution plus rapide que l'algorithme naïf en $\mathcal{O}(n!)$. Gurevich et Shelah (1987) proposent une approche de type *diviser-pour-régner* pour résoudre ce problème en temps $\mathcal{O}(4^n\, n^{\log(n)})$ et espace polynomial. Toutes ces techniques ont été approfondies, développées et complétées durant

[1] La notation $\mathcal{O}^*$ supprime les facteurs polynomiaux.

les dix dernières années. *Mesurer-pour-conquérir* est venue améliorer les analyses de complexité des algorithmes de *branchement* (Fomin *et al.*, 2009c), *inclusion-exclusion* a autorisé la résolution de COLORATION en temps $\mathcal{O}^*(2^n)$ (Björklund *et al.*, 2009), *convolution* a amélioré l'efficacité de certaines approches par *programmation dynamique* (Björklund *et al.*, 2007). Là encore, on peut certainement trouver la racine de ces techniques dans des articles un peu plus anciens (Beigel et Eppstein, 2005; Eppstein, 2003b, 2006; Karp, 1982; Bax, 1993, 1994; Bax et Franklin, 1996; Yates, 1937; Rota, 1964). Pour preuve que le domaine de recherche de l'algorithmique modérément exponentielle est devenu mature, des résultats négatifs sont aussi prouvés. Essentiellement basées sur l'hypothèse *ETH* (*Exponential Time Hypothesis*) de Impagliazzo *et al.* (2001), des *réductions* ont montré qu'un certain nombre de problèmes ne pouvaient pas être résolus dans un temps sous-exponentiel, c'est-à-dire de la forme $\mathcal{O}(2^{o(n)})$. La conception d'algorithmes modérément exponentiels est donc certainement la meilleure alternative si on souhaite résoudre de façon exacte ces problèmes.

Il y a un autre type de problèmes pour lesquels un temps exponentiel est nécessaire. Ce sont les problèmes qui demandent d'énumérer un nombre exponentiel d'objets combinatoires. Typiquement, on se donne une propriété et le problème consiste à produire tous les objets respectant celle-ci. Idéalement, on espère construire un algorithme dont le temps d'exécution est du même ordre de grandeur que le nombre des objets à énumérer. Les algorithmes modérément exponentiels, basés sur du *branchement*, se révèlent être d'excellents outils pour aborder ce genre de problèmes. En plus de produire les objets demandés, par l'analyse de leur temps d'exécution on peut déduire des bornes combinatoires sur le nombre d'objets énumérés. C'est-à-dire que des bornes (supérieures) sont obtenues par des techniques algorithmiques ; le travail consiste alors à trouver le meilleur algorithme (et à démontrer son temps d'exécution) pour obtenir la borne la plus serrée possible. Un certain nombre de bornes ont ainsi été obtenues récemment pour les coupes cycles minimaux, les ensembles dominants minimaux, les séparateurs minimaux, les cliques maximales potentielles, les bicliques, ... (Fomin *et al.*, 2008a; Couturier *et al.*, 2013b; Gaspers et Mackenzie, 2015; Fomin et Villanger, 2010; Gaspers *et al.*, 2012).

Un certain nombre d'algorithmes modérément exponentiels ont été développés pour des problèmes d'optimisation et en particulier pour des problèmes de graphes. Le problème phare du domaine est probablement ENSEMBLE STABLE. Depuis les années 70 et l'algorithme de Tarjan et Trojanowski (1977) en $\mathcal{O}^*(2^{n/3})$, une série de publications est parue Jian (1986); Robson (1986); Shindo et Tomita (1990); Beigel (1999); Fomin *et al.* (2009c); Kneis *et al.* (2009); Bourgeois *et al.* (2012). Le meilleur algorithme actuel, dû à Xiao et Nagamochi (2013), utilise du *branchement* et a un temps d'exécution de $\mathcal{O}(1.2002^n)$. D'autres problèmes ont été étudiés tels que DOMINATION, 3-SAT, COUPE CYCLE, CHEMIN HAMILTONIEN (Fomin *et al.*, 2009c; Iwata, 2011; Xiao et Nagamochi, 2015; Björklund, 2014). De plus, des états de l'art présentant les techniques ont été publiés (Woeginger, 2001, 2004; Iwama, 2004; Fomin *et al.*, 2005; Schöning, 2005b,a; Fomin et Kaski, 2013), des thèses ont été écrites sur le sujet (Liedloff, 2007; Stepanov, 2008; Gaspers, 2010; Bourgeois, 2009; van Rooij, 2011; Couturier, 2012; Cochefert, 2012; Letourneur, 2015) et une monographie est parue (Fomin et Kratsch, 2010).

Dans les chapitres qui suivent, nous reprenons des techniques connues et cherchons à les étendre, à les pousser le plus possible, afin de résoudre des problèmes de décision, d'optimisation et également d'énumération. Nous établirons des bornes combinatoires sur un certain nombre d'objets, par des algorithmes et pour des algorithmes. Nous venons de mentionner que les algorithmes d'énumération ont pour conséquence heureuse l'obtention de bornes. Ces bornes sont utiles, non seulement pour leur intérêt mathématique, mais aussi algorithmique. En effet, les objets

énumérés sont bien souvent utilisés par d'autres algorithmes pour la résolution de problèmes. Par exemple, l'énumération des ensembles stables (maximaux) est utile à la résolution du problème COLORATION et une borne supérieure sur ceux-ci peut être nécessaire dans des analyses de temps d'exécution. Nous énumérerons et établirons de telles bornes combinatoires sur certains objets. Pour certains de ces objets, nous les utiliserons immédiatement pour résoudre d'autre problèmes.

Les techniques que nous utilisons sont variées : *programmation dynamique*, *branchement*, *mesurer-pour-conquérir*, *diviser-pour-régner*, *convolution*, *trier-et-chercher*. Nous chercherons à généraliser certaines de ces techniques (par exemple pour *trier-et-chercher*) et à les mettre en oeuvre de façon astucieuse (par exemple avec l'algorithme le plus efficace pour ÉTIQUETAGE $L(2,1)$ ou pour DOMINATION AVEC CAPACITÉS). Notre objectif sera aussi d'établir des algorithmes pour résoudre des problèmes avec un temps d'exécution au pire des cas borné par $\mathcal{O}(\alpha^n)$ avec α une constante à déterminer, idéalement la plus petite possible.

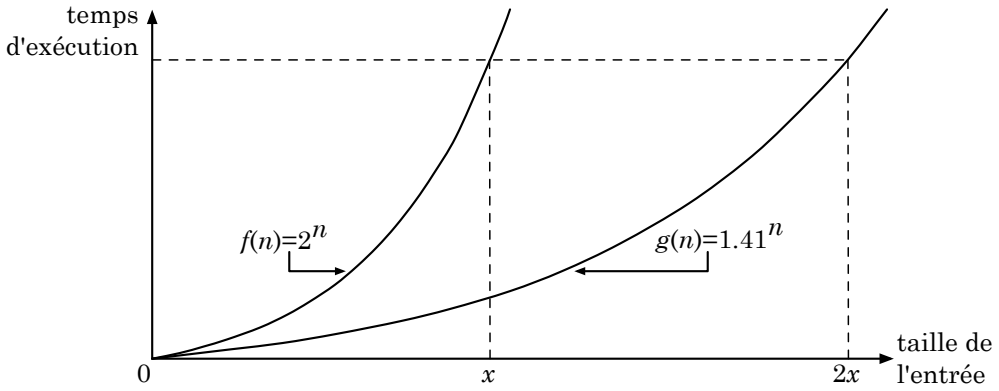


FIGURE 1 – Réduire la base α de l'exponentielle permet de traiter des entrées de taille plus grande dans une même unité de temps. La figure compare la croissance des fonctions $f(n) = 2^n$ et $g(n) = 1.41^n$ ($\sqrt{2} \approx 1.41\ldots$).

Dans ce manuscrit, nous proposons un focus sur des résultats de recherche récents. Le document est composé de quatre chapitres. Le premier chapitre se veut introductif, en apportant des définitions et notions essentielles de complexité et d'algorithmique. Les trois chapitres suivants constituent le cœur de ce manuscrit et présentent des travaux autour de trois thématiques abordées sous l'angle des algorithmes modérément exponentiels : des étiquetages de graphes, des problèmes de domination et des problèmes d'ordonnancement.

Chapitre 1 – préliminaires.

Nous présentons dans ce chapitre l'essentiel des notions qui nous seront utiles tout au long de ce manuscrit. Nous commençons par rappeler quelques notations asymptotiques puis définirons la notion de graphe et donnons des propriétés s'y rapportant. Quelques problèmes classiques de graphes sont définis ainsi que des problèmes d'ordonnancement. Finalement, nous expliquons des techniques de conception et d'analyse déployées dans le manuscrit. Certaines de ces techniques sont abondamment étudiées dans la littérature et d'autres moins connues. Nous terminons le chapitre avec l'hypothèse en temps exponentiel.

Chapitre 2 – étiquetages $L(2,1)$.

Au chapitre 2, nous étudions le problème de l'étiquetage $L(2,1)$ d'un graphe pour lequel nous donnons une collection de résultats, basés sur diverses techniques : *branchement*, *mesurer pour conquérir*, *programmation dynamique*, *diviser-pour-régner*.

> ÉTIQUETAGE $L(2,1)$
>
> ***Entrée.*** Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.
>
> ***Question.*** Existe-t-il une application f de V dans $\{0, 1, \ldots, k\}$ telle que :
>
> 1. $|f(u) - f(v)| \geq 2$ pour tout $\{u, v\} \in E$;
> 2. $|f(u) - f(v)| \geq 1$ (*i.e.* $f(u) \neq f(v)$) pour tout $u, v \in V$ tels que $N(u) \cap N(v) \neq \emptyset$.

Dans un premier temps, nous présentons des algorithmes pour une largeur d'étiquetage fixée, c'est-à-dire lorsque le nombre d'étiquettes disponibles pour étiqueter le graphe est borné par une (*petite*) constante. Nous commençons par l'étude des étiquetages de largeur au plus 4. Nous présentons un algorithme de *branchement* dont nous analysons le temps d'exécution de façon classique. Avec une analyse plus raffinée de type *mesurer-pour-conquérir*, nous établissons un temps d'exécution de $\mathcal{O}(1.3006^n)$ pour la résolution de ce problème. La construction d'une borne inférieure établira que son temps d'exécution au pire des cas est d'au moins $\Omega(1.2290^n)$.

On s'intéresse ensuite à l'énumération des étiquetages de largeur 5 dans les graphes cubiques. L'analyse d'un algorithme de *branchement* établira une borne supérieure de $\mathcal{O}(1.7990^n)$ sur le nombre maximum de ces étiquetages. Nous construirons ensuite une famille de graphes connexes cubiques ayant au moins $2^{2n/7} > 1.2190^n$ étiquetages $L(2,1)$ de largeur 5.

Dans un second temps, nous développons des approches pour déterminer la plus petite largeur d'un étiquetage $L(2,1)$. Nous commençons par concevoir des algorithmes ne nécessitant qu'un espace polynomial. Notre premier algorithme s'exécutera en temps $\mathcal{O}((k-2.5)^n)$, où k est la largeur de l'étiquetage. Nous nous efforçons ensuite de concevoir des algorithmes dont le temps d'exécution est indépendant de k (ou, tout du moins, le facteur exponentiel du temps d'exécution ne dépend pas de k). Nous développons une approche de type *diviser-pour-régner* qui aboutit à un algorithme en temps $\mathcal{O}(7.4920^n)$ (et espace polynomial). Le point d'orgue de ce chapitre est un algorithme en temps $\mathcal{O}(2.6488^n)$ obtenu par de la *programmation dynamique*. Ce dernier algorithme, dont le temps d'exécution est le meilleur connu à ce jour, nécessite aussi un espace exponentiel. Pour établir le temps d'exécution de ces deux derniers algorithmes, des résultats combinatoires sont nécessaires. Nous les établirons par des approches de *branchement* dans la dernière section du chapitre.

Ces travaux sont issus de plusieurs collaborations avec Jean-François Couturier, Petr A. Golovach, Frédéric Havet, Konstanty Junosza-Szaniawski, Martin Klazar, Jan Kratochvíl, Dieter Kratsch, Artem V. Pyatkin, Peter Rossmanith et Pawel Rzazewski. Cela fait longtemps que je travaille sur les étiquetages $L(2,1)$, avec différents groupes d'auteurs, et ce chapitre est l'occasion de présenter une grande partie de ces résultats de façon unifiée.

Chapitre 3 – autour de la domination.

Au chapitre 3, nous regardons des problèmes autour de la domination d'un graphe. Nous commençons par proposer un algorithme exponentiel pour une généralisation du problème DOMINATION

appelée DOMINATION AVEC CAPACITÉS, les sommets se voient attribuer des capacités. Ce problème est formellement défini ainsi :

> **DOMINATION AVEC CAPACITÉS**
>
> ***Entrée.*** Un graphe $G = (V, E)$, une fonction de capacités $c : V \to \mathbb{N}$ et un entier $k \in \mathbb{N}$.
>
> ***Question.*** Existe-t-il un sous-ensemble de sommets $D \subseteq V$ et une fonction $f : V \setminus D \to D$ tels que $|D| \leq k$ et
>
> - $f(u) \in N(u) \cap D$ pour tout $u \in V \setminus D$;
> - $|f^{-1}(v)| \leq c(v)$ pour tout $v \in D$?

Cygan *et al.* (2011) montrent que ce problème peut être résolu en temps $\mathcal{O}^*(1.89^n)$; nous proposons un nouvel algorithme qui se révèle être le meilleur connu à ce jour. Pour cela, nous introduisons une approche que nous pouvons qualifier de « gagnante-gagnante » qui dans une étape préliminaire utilise l'approche de Cygan *et al.* (2011), afin de garantir une certaine structure sur l'instance dans une seconde étape. La seconde étape est résolue en « poussant » une technique de Koivisto (2009) pour un problème de partitionnement. Nous montrons que l'analyse de l'approche de Koivisto (2009) peut à la fois être améliorée (par des calculs mathématiques plus précis) et qu'elle peut être étendue pour considérer des ensembles de taille non nécessairement bornée par une constante. Nous obtenons un algorithme dont le temps d'exécution $O(n^d \cdot c^n)$ est un compromis entre son terme exponentiel c^n et son terme polynomial n^d, avec $1.8463 \leq c \leq 1.8844$ et $10 \leq d \leq 40\,000$. Cela signifie que la base c de l'exponentielle peut être réduite, au coût de l'augmentation du degré d du polynôme.

La seconde partie de ce chapitre s'intéresse à l'énumération des ensembles dominants minimaux dans deux classes de graphes : les graphes splits et les graphes d'intervalles. Un ensemble $D \subseteq V$ d'un graphe $G = (V, E)$ est un ensemble dominant si tout sommet de $V \setminus D$ a au moins un voisin dans D. Il est minimal si tout ensemble $D' \subset D$ n'est pas dominant. Il est établi que pour tout entier n, le nombre maximum d'ensembles dominants minimaux $dom(n)$, dans les graphes à n sommets, respecte $1.5704^n < dom(n) < 1.7159^n$ (Fomin *et al.*, 2008b). Plus récemment, Couturier *et al.* (2013b) ont étudié $dom(n)$ pour quelques classes de graphes particulières et ils ont conjecturé que, pour les graphes d'intervalles propres et les graphes splits, on a $dom(n) = 3^{n/3}$. De plus, il existe des graphes d'intervalles et splits ayant $3^{n/3}$ ensembles dominants minimaux. Nous montrons, par la conception et l'analyse de deux algorithmes d'énumération différents, que ces graphes n'ont effectivement jamais plus que $3^{n/3}$ ensembles dominants minimaux. Cela répond à la conjecture de Couturier *et al.* (2013b), y compris pour les graphes d'intervalles.

Les résultats sur le problème DOMINATION AVEC CAPACITÉS ont été obtenus en collaboration avec Ioan Todinca et Yngve Villanger. Ils témoignent d'une collaboration de longue date avec l'université de Bergen (Norvège). L'énumération des ensembles dominants minimaux est issue de travaux avec Romain Letourneur, dont j'ai co-encadré la thèse qu'il a soutenue en juillet 2015, et Jean-François Couturier.

Chapitre 4 – problèmes d'ordonnancement.

Au chapitre 4, nous développons des algorithmes avec une garantie de performance au pire des cas pour des problèmes d'ordonnancement. La *programmation dynamique* se révèle être la technique

de prédilection pour bon nombre d'entre eux. Nous couplons cette approche avec le *produit de convolution* de fonctions pour améliorer les bornes sur le temps d'exécution. Par l'approche de *programmation dynamique*, nous résolvons des problèmes d'ordonnancement à plusieurs machines et un problème de flow-shop à trois machines. Pour ce dernier problème, nous utilisons une approche par *relâchement*.

La seconde partie de ce chapitre s'intéresse à une technique ancienne mais peu connue : *trier et chercher*. Cette technique a été initialement introduite par Horowitz et Sahni (1974) pour la résolution du problème SAC-À-DOS en temps $\mathcal{O}^*(2^{n/2})$. Dans un premier temps, nous abstrayons et étendons cette technique. Elle se révèlera très utile pour la résolution d'un grand nombre de problèmes d'affectation. Nous qualifions ces problèmes de *problèmes à une seule contrainte* (PSC) et de *problèmes à plusieurs contraintes* (PPC). Pour se donner un avant-goût, nous définissons un problème PPC de la façon suivante. Soient $A = \left(\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_{n_A}\right)$ une table de n_A vecteurs de dimension d_A et $B = \left(\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_{n_B}\right)$ une table de n_B vecteurs $\vec{b}_k = (b_k^0, b_k^1, \ldots, b_k^{d_B})$ de dimension $d_B + 1$. Soient f et g_ℓ $(1 \leq \ell \leq d_B)$ $d_B + 1$ fonctions de $\mathbb{R}^{d_A+1}$ dans $\mathbb{R}$, croissantes par rapport à leur dernière variable. Le problème PPC est défini par :

$$
\begin{array}{l}
\text{Minimiser } f(\vec{a}_j, b_k^0) \\
\text{sous contraintes} \\
\quad g_\ell(\vec{a}_j, b_k^\ell) \geq 0 \quad 1 \leq \ell \leq d_B \\
\quad \vec{a}_j \in A \\
\quad \vec{b}_k \in B.
\end{array}
$$

Nous montrons qu'un tel problème peut se résoudre plus rapidement qu'avec une approche naïve grâce à l'extension de la technique *trier-et-chercher*. Pour parvenir à ce résultat, nous utiliserons une structure de données particulière : les *arbres d'intervalles* (en anglais, *range trees*).

Nous terminons ce chapitre par une application des PPC à deux problèmes d'ordonnancement : le problème $P||C_{\max}$ et le problème $P|d_i|\sum w_i U_i$. Pour ces deux problèmes, n tâches doivent être ordonnancées sur m machines parallèles identiques. On note par C_j la date de fin de la dernière tâche traitée par la machine j. L'objectif du problème $P||C_{\max}$ est de déterminer un ordonnancement des tâches de sorte à minimiser le *makespan*, c'est-à-dire la valeur $C_{\max} = \max_{1\leq j\leq m} C_j$. Pour le problème $P|d_i|\sum w_i U_i$, à chaque tâche est associée une date de fin souhaitée d_i ainsi qu'une pénalité de retard w_i. Chaque tâche doit être affectée à l'une des machines de sorte à minimiser le nombre pondéré de travaux en retard. C'est-à-dire qu'il faut minimiser $\sum_{1\leq i\leq n} w_i U_i$ où $U_i = 1$ si la date de fin de traitement de la tâche i est postérieure à la date d_i, et $U_i = 0$ sinon.

Suite à mon arrivée à Orléans en 2008 comme maître de conférences, une collaboration est née avec Christophe Lenté, Ameur Soukhal et Vincent T'Kindt du *Laboratoire d'Informatique* de l'université François Rabelais de Tours. Ces spécialistes des problèmes d'ordonnancement m'ont introduit leurs problématiques et nous avons cherché à résoudre certains de ces problèmes par des méthodes exactes avec garantie du temps d'exécution au pire des cas. Les résultats de ce chapitre sont issus de ces découvertes.

Conclusion.

Nous concluons chacun des chapitres par quelques questions ouvertes et perspectives. La conclusion générale de ce manuscrit sera plutôt propice à un bilan.

Chapitre 1

Préliminaires

Cette section introduit les notions essentielles à la compréhension de ce manuscrit. Y seront définies les notations asymptotiques usuelles ainsi que la notation $\mathcal{O}^*(\cdot)$ que l'on retrouvera souvent dans les analyses de complexités des algorithmes modérément exponentiels. Nous donnerons diverses définitions autour des graphes et classes de graphes. Les notions plus pointues et propres à un résultat seront définies dans le chapitre correspondant. Nous rappellerons les définitions des problèmes de graphes les plus communes, ainsi que la définition de problèmes d'ordonnancement. Pour terminer ce chapitre, nous présenterons quelques techniques « classiques » pour la conception et l'analyse d'algorithmes modérément exponentiels.

Sommaire

La plupart des notions données dans ce chapitre peuvent être retrouvées dans des ouvrages généralistes tels que ceux de Brandstädt *et al.* (1999), West (2001), Diestel (2010) et de Cormen *et al.* (2009). Une partie des définitions présentées dans la suite sont extraites de ma thèse consacrée aux algorithmes exponentiels (Liedloff, 2007). Depuis la rédaction de cette thèse en 2007, la parution de l'excellente monographie de Fomin et Kratsch (2010) traite de façon exhaustive le domaine des algorithmes modérément exponentiels. Nous référons le lecteur intéressé à cet ouvrage pour des lectures complémentaires.

1.1 Algorithmes et complexité

Nous ne rappelons pas les classes de complexité usuelles (P, NP, PSPACE, ...), dont le lecteur pourra trouver les définitions et propriétés dans les ouvrages de Garey et Johnson (1979), Papadimitriou (1994) ou Arora et Barak (2009). Ces ouvrages présentent également les notions de réductions (en particulier, les réductions polynomiales) qui sont utiles pour établir la NP-complétude ou PSPACE-complétude de certains problèmes. Dans ce manuscrit, la plupart des problèmes étudiés sont NP-complets.

Un problème de décision est un problème pour lequel la réponse est soit « oui », soit « non ». La classe de complexité NP (pour *non deterministic polynomial*) contient l'ensemble des problèmes de décision qui peuvent être résolus en temps polynomial par un algorithme *non déterministe* ; autrement dit, c'est l'ensemble des problèmes pour lesquels un algorithme polynomial peut vérifier une solution (un *certificat*). Évidement, on a la relation $P \subseteq NP$. Jusqu'à présent, de nombreux problèmes ont été montrés NP-complets et pour aucun d'entre eux un algorithme polynomial n'a été trouvé. La grande question ouverte en informatique est de savoir si $P = NP$.

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

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

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

Dans la suite, lorsque nous étudierons le temps d'exécution d'un algorithme, il s'agira généralement du temps d'exécution au pire des cas. Cela signifie que, si $T(n)$ dénote le temps d'exécution au pire des cas d'un algorithme, alors pour toute entrée son temps d'exécution n'excédera pas $T(n)$, à un facteur multiplicatif constant près.

Pour compléter ces trois notations, Woeginger (2001) a introduit dans la notation $\mathcal{O}^*$ qui autorise la suppression de tous les termes bornés par un facteur polynomial dépendant de la taille de l'entrée. Ainsi une complexité de la forme $\mathcal{O}(f(n) \cdot poly(n))$, où n est la taille de l'entrée et $poly(n)$ un polynôme dépendant de n, se notera $\mathcal{O}^*(f(n))$. Cela nous permettra en particulier d'insister sur la croissance exponentielle de la fonction qui borne le temps d'exécution. Par exemple un temps d'exécution $2 + n^3 \cdot 1.8788^n$ se réécrira $\mathcal{O}^*(1.8788^n)$. Cette nouvelle notation asymptotique se justifie par le fait que $2 + n^3 \cdot 1.8788^n$ est compris entre 1.8788^n et 1.8789^n pour toute valeur de n

supérieure à 26. Par conséquent, la fonction $2+n^3 \cdot 1.8788^n$ pourra également se réécrire simplement sous la forme $\mathcal{O}(1.8789^n)$. C'est cette dernière forme de notation que nous privilégierons par la suite afin de se concentrer sur les termes exponentiels dans l'expression du temps d'exécution.

Notons que toutes ces notations peuvent également être employées pour quantifier l'espace nécessaire à un algorithme, outre son temps d'exécution.

1.2 Graphes

1.2.1 Quelques notions

Un *graphe* $G = (V, E)$ est un couple où V désigne un ensemble de *sommets* et E un ensemble d'*arêtes*. Nous ne considérerons par la suite que des graphes finis, c'est-à-dire des graphes pour lesquels les ensembles V et E sont finis. On note habituellement par $V(G)$ (respectivement $E(G)$) l'ensemble des sommets (respectivement des arêtes) d'un graphe G, et l'on définit $n(G) = |V(G)|$ et $m(G) = |E(G)|$. S'il n'y a pas d'ambiguïté on utilisera simplement V, E, n et m en omettant de préciser le graphe considéré.

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

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

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

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

Un *chemin* dans un graphe $G = (V, E)$ est une séquence de sommets $(v_1, \ldots, v_k)$ telle que $\{v_i, v_{i+1}\} \in E$ pour $1 \leq i \leq k - 1$. La longueur d'un tel chemin est $k - 1$. Étant donnés deux sommets u et v, la *distance* $d(u, v)$ entre ces deux sommets est la longueur d'un plus court chemin entre ceux-ci.

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

On dit que $H = (V', E')$ est un *sous-graphe* de $G = (V, E)$ si $V' \subseteq V$ et $E' \subseteq E$. On appelle *composante connexe* de G un sous-graphe connexe maximal (par l'inclusion) de G. Etant donnés un graphe $G = (V, E)$ et un sous-ensemble de sommets $S \subseteq V$, le graphe $G[S] = (S, \{\{u, v\} \in E \ : \ u, v \in S\})$ est appelé *sous-graphe* de G *induit* par S.

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

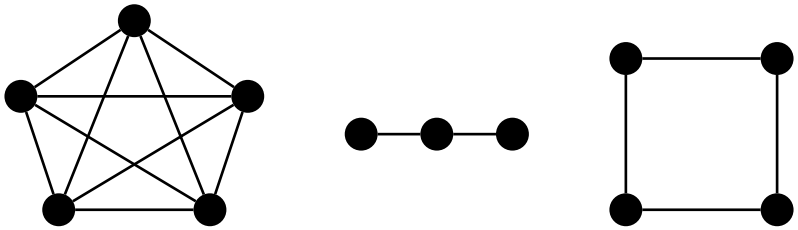


FIGURE 1.1 – Un graphe possèdant trois composantes connexes : un graphe complet à 5 sommets (K_5), un chemin à 3 sommets (P_3) et un cycle à 4 sommets (C_4).

Un ensemble $C \subseteq E$ d'un graphe $G = (V, E)$ est un *couplage* si les arêtes de C ne partagent pas d'extrémité commune. Si on note $V(C)$ les sommets apparaissant comme extrémité d'arêtes de C, et si $G[V(C)]$ ne contient pas d'autre arête que celles de C, nous dirons que C est un *couplage induit*.

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

Étant donné un graphe $G = (V, E)$, nous définissons son carré, comme le graphe $G^2 = (V, E')$ où $E' = \{(u, v) : (u, v) \in E$ ou $\exists z \in V$ t.q. $(u, z), (z, v) \in E\}$. Autrement dit, G^2 contient comme arêtes toutes celles de G, complétées par une nouvelle arête entre chaque paire de sommets situés à distance 2.

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

Un ensemble 2-*stable* (ou 2-*packing*) d'un graphe $G = (V, E)$ est un sous-ensemble de sommets $S \subseteq V$ tel que S soit un stable de G^2

Étant donné $r \in \mathbb{N}$, le graphe $K_{1,r}$ est appelé une *étoile*. Il est obtenu de $r + 1$ sommets, dont un sommet particulier (appelé le centre de l'étoile) est adjacent aux autres r sommets.

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

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

Étant donné un graphe connexe $G = (V, E)$, un arbre couvrant de G est un graphe $T = (V', E')$ tel que $V = V'$, $E' \subseteq E$, et tel que T soit un graphe connexe sans cycle. Un arbre DFS (en anglais, *depth-first search*) est un arbre obtenu à partir d'un parcours en profondeur depuis un certain sommet racine r. La définition d'un parcours DFS (ou BFS, pour *breadth-first search*) peut être trouvée dans (Cormen *et al.*, 2009).

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

Nous donnons ci-après la définition de deux concepts importants : la *largeur arborescente* et la *largeur linéaire*. Bien que peu utilisées dans ce manuscrit, nous y ferons parfois référence.

Définition 1.1 (Robertson et Seymour (1986))**.** Une *décomposition arborescente* d'un graphe $G = (V, E)$ est un couple $\big(\{X_i, i \in I\}, \mathcal{T} = (I, E_T)\big)$ où chaque $X_i \subseteq V$ est un sous-ensemble des sommets de G (habituellement appelé *sac*) et $\mathcal{T}$ est un arbre tel que :

1. $\bigcup_{i \in I} X_i = V$;
2. pour chaque arête $\{u, v\} \in E$, il existe $i \in I$ tel que $\{u, v\} \subseteq X_i$;
3. pour tout sommet $v \in V$, l'ensemble des sommets $i \in I$ de $\mathcal{T}$ pour lesquels $v \in X_i$ induit un sous-arbre (connexe) de $\mathcal{T}$.

La largeur de la décomposition est donnée par $\max_{i \in I} |X_i| - 1$. La *largeur arborescente* du graphe G, notée $tw(G)$ (pour *treewidth*), est la largeur minimum sur toutes les décompositions arborescentes de G.

Définition 1.2 (Robertson et Seymour (1983))**.** Une *décomposition linéaire* d'un graphe G est une décomposition arborescente $\big(\{X_i, i \in I\}, \mathcal{T} = (I, E_T)\big)$ pour laquelle $\mathcal{T}$ est un chemin.

La *largeur linéaire* du graphe G, notée $pw(G)$ (pour *pathwidth*), est la largeur minimum sur toutes les décompositions linéaires de G.

Notons que les calculs de ces deux paramètres sont des problèmes NP-complets.

Nous donnerons des définitions ou des propriétés supplémentaires au moment où elles seront nécessaires à la discussion d'un chapitre. Néanmoins, nous renvoyons le lecteur aux ouvrages de Diestel (2010), de Golumbic (2004) ou celui de West (2001) pour davantage d'informations.

1.2.2 Classes de graphes

Les graphes peuvent être regroupés en familles de graphes ayant des propriétés communes. On parle alors de *classes de graphes*. Une classe de graphes est dite *héréditaire* si pour chaque graphe de la classe, tous ses sous-graphes induits appartiennent aussi à la classe.

Un grand nombre de classes a été défini. De façon exhaustive, le site internet de Ridder *et al.* (2015) en liste une très grande collection. Les motivations pour l'étude de ces différentes classes sont diverses mais en particulier l'étude de (la complexité de) problèmes restreints à ces classes de graphes motive de nombreux articles de la littérature. Par exemple, il est aujourd'hui bien connu qu'un simple algorithme glouton permet de résoudre le problème DOMINATION en temps linéaire sur les arbres (Garey et Johnson, 1979), alors que ce problème reste NP-complet sur les graphes pour lesquels tous les sommets ont pour degré au plus 3 Garey et Johnson (1979). Dans les 16e colonnes sur la NP-complétude, Johnson (1987) se consacre à l'étude des restrictions de graphes et leurs effets. On peut en particulier s'y référer pour connaître la complexité, sur diverses classes de graphes, de quelques problèmes NP-complets classiques dont ENSEMBLE STABLE, CLIQUE, COLORATION, CYCLE HAMILTONIEN, DOMINATION ou encore COUPE MAXIMUM. Nous définirons certains de ces problèmes dans la suite de ce chapitre.

Passons maintenant à quelques classes de graphes connues qui nous seront utiles par la suite. Pour une vue plus générale sur un large ensemble de classes de graphes, l'ouvrage de Brandstädt *et al.* (1999) servira de référence.

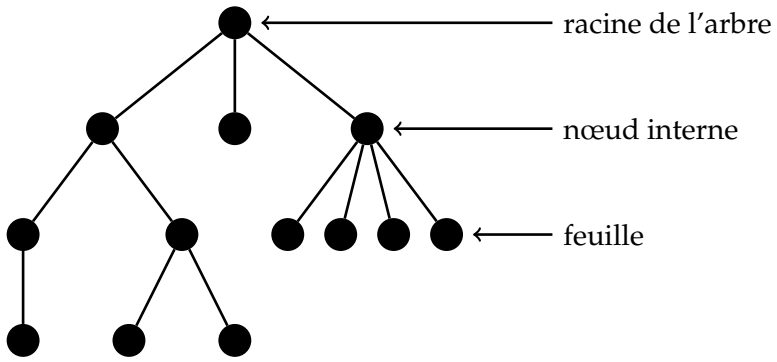


FIGURE 1.2 – Un exemple d'arbre enraciné.

Graphes sans cycle

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

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

Graphes de degré borné

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

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

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

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

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

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

Graphes bipartis

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

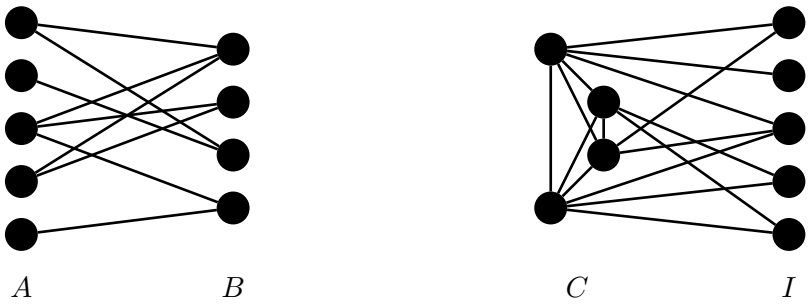


FIGURE 1.3 – Exemple d'un graphe biparti (à gauche) et d'un graphe split (à droite).

Graphes splits

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

- C est une clique de G ;
- I est un ensemble stable de G.

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

Graphes d'intervalles et d'intervalles circulaires

Soit $\mathcal{I}$ une collection de n intervalles de la droite réelle et soit $G_{\mathcal{I}} = (V, E)$ son graphe d'intersection tel que :

(i) il existe une bijection qui associe à chaque intervalle $I_v \in \mathcal{I}$ un sommet v de V ;

(ii) deux sommets $u \in V$ et $v \in V$ sont adjacents dans G si et seulement si les deux intervalles correspondants dans $\mathcal{I}$ ont une intersection non vide.

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

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

Le graphe est dit *propre* si de plus il n'y a aucun intervalle de $\mathcal{I}$ qui est inclus dans un autre.

Graphes de largeur arborescente bornée

La notion de largeur arborescente a été introduite précédemment. L'ensemble des graphes pour lesquels la largeur arborescente d'un graphe peut être bornée par une constante sont les graphes de *largeur arborescente bornée*.

Relations d'inclusion

Il existe des relations d'inclusion entre les classes de graphes. Les preuves de ces relations sont pour certaines faciles à voir (par exemple un graphe d'intervalle est aussi un graphe d'intervalle circulaire) ; d'autres, plus difficiles, peuvent se trouver dans la littérature (Brandstädt *et al.*, 1999). Les définitions des classes de graphes et relations d'inclusion non données dans ce manuscrit sont présentées dans (Brandstädt *et al.*, 1999).

1.2.3 Quelques problèmes classiques

Les graphes sont un outil de modélisation et un grand nombre de problèmes peuvent s'exprimer comme des problèmes de graphes. Certains de ceux-ci sont « faciles » à résoudre (dans le sens où l'on connait un algorithme polynomial pour les résoudre) et d'autres plus « difficiles », typiquement NP-complets. Nous définissons ci-après quelques problèmes NP-complets fondamentaux dont nous ferons mention dans ce manuscrit ou que nous étudierons au grès des chapitres.

Clique et Ensemble stable

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

CLIQUE

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

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

ENSEMBLE STABLE

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

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

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

Coloration

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

COLORATION

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

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

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

Domination et variantes

Un problème très étudié et qui admet beaucoup de variantes et généralisations est la DOMINATION :

> **DOMINATION**
>
> ***Entrée.*** Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.
>
> ***Question.*** Est-ce que G admet un ensemble dominant de taille au plus k ? C'est-à-dire, existe-t-il un sous-ensemble $D \subseteq V$ de cardinalité au plus k tel que pour tout $u \in V \setminus D$, le sommet u ait au moins un voisin dans D ?

On note $\gamma(G)$ la taille d'un plus petit ensemble dominant de G. Le problème d'optimisation correspondant demande de déterminer un tel ensemble de cardinalité $\gamma(G)$.

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

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

Il existe aussi des variations du problème classique DOMINATION qui autorisent un sommet à dominer plus que son voisinage (à distance 1) et/ou attribuent des poids aux sommets.

Un problème autour de DOMINATION est NOMBRE DOMATIQUE :

> **NOMBRE DOMATIQUE**
>
> ***Entrée.*** Un graphe $G = (V, E)$ et un entier $k \in \mathbb{N}$.
>
> ***Question.*** Est-ce que les sommets de G peuvent être partitionné en au moins k ensembles dominants de G ?

Les ouvrages de Haynes *et al.* (1998b,a) offrent un panorama assez large avec plus de 75 variantes recensées.

Le problème du voyageur de commerce

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

VOYAGEUR DE COMMERCE

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

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

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

1.3 Notations utilisées en ordonnancement

Nous désignons par $\mathcal{J}$ un ensemble de n *travaux* (également appelés *tâches*) et notés $J_1, J_2, \ldots, J_n$. Par $\mathcal{M}$, nous désignons un ensemble de m *machines* que nous notons $M_1, M_2, \ldots, M_m$. Dans la suite, nous supposerons que chaque machine ne peut traiter qu'au plus une tâche à la fois et qu'une tâche n'est traitée que par au plus une machine à la fois. Nous supposons que chaque machine ne peut traiter qu'au plus une tâche à un moment donné et qu'une tâche ne peut être affecté qu'à au plus une machine à un moment donné.

La notation en trois champs $\alpha|\beta|\gamma$ sert habituellement à décrire les problèmes d'ordonnancement. Cette notation a été introduite par Graham *et al.* (1979) et reprise par Blazewicz *et al.* (1983). Le premier champ α décrit les machines, le deuxième champ β décrit les contraintes additionnelles et le troisième champ γ définit le critère d'optimalité. En suivant la description donnée parBlazewicz *et al.* (1983), nous pouvons détailler ces trois champs (que nous redonnerons à la section 4.1).

- α : la nature de l'atelier. Le champ $\alpha = \alpha_1\alpha_2$ définit l'environnement des machines.

  Pour $\alpha_1 \in \{P, Q, R\}$, chaque travail J_j consiste en une unique opération qui doit être traitée sur l'une quelconque des machines. La durée opératoire (aussi appelée de traitement) de J_j sur M_i est donnée par p_{ij} ($1 \leq i \leq m$ et $1 \leq j \leq n$).

  - si $\alpha_1 = P$ alors l'atelier est composé de machines parallèles identiques ($p_{ij} = p_j$ où p_j est une durée donnée pour traiter J_j) ;
  - si $\alpha_1 = Q$ alors l'atelier est composé de machines parallèles uniformes ($p_{ij} = p_j/q_i$ pour une vitesse q_i de M_i donnée) ;
  - si $\alpha_1 = R$ alors l'atelier est composé de machines parallèles non reliées (p_{ij} est arbitraire).

  Pour $\alpha_1 \in \{O, F, J\}$, chaque travail J_j consiste en un ensemble O_{ij} de m_j opérations qui doivent chacune être opérée sur une machine μ_{ij} pendant une durée p_{ij} ($1 \leq i \leq m_j$ et $1 \leq j \leq n$).

  - si $\alpha_1 = O$ alors le cheminement est libre ($m_j = m$ et $\mu_{ij} = M_i$) ;
  - si $\alpha_1 = F$ alors le cheminement est unique ($m_j = m$, $\mu_{ij} = M_i$ et $O_{i-1,j}$ doit être terminé avant que O_{ij} ne puisse commencer) ;
  - si $\alpha_1 = J$ alors le cheminement est multiple (m_j et μ_{ij} sont arbitraires, $\mu_{i-1,j} \neq \mu_{ij}$ et $O_{i-1,j}$ doit être terminé avant que O_{ij} ne puisse commencer).

  Si α_2 est précisé, alors c'est un entier qui indique le nombre de machines (qui est constant) ; s'il n'est pas précisé, alors le nombre de machines fait partie de l'entrée.

- β : les contraintes additionnelles. Le champ β donne des caractéristiques additionnelles sur les travaux.
  - si β contient $p_i = p$ ou $p_{ij} = p$ cela signifie que la durée de traitement est identique pour tous les travaux. Par $p_i = 1$ ou $p_{ij} = 1$, on indique que cette durée est unitaire ;
  - si β contient r_i, alors pour chaque travail, une date de disponibilité est donnée et le travail ne peut être traité avant cette date ;
  - si β contient d_i, alors pour chaque travail, une date de fin souhaitée est donnée et le travail ne peut être traité après cette date ;
  - si β contient $size_i = k$, alors chaque travail doit au même moment être traité par k machines ;
  - si β contient « pmtn », alors les travaux peuvent à tout moment être préemptés et leur traitement terminer plus tard sur éventuellement une autre machine ;
  - si β contient « prec », alors un graphe orienté acyclique est donné et indique des relations de précédence entre les travaux ;
  - ...
- γ : le critère d'optimalité. Ce dernier champ γ indique le critère à minimiser. Chaque ordonnancement définit pour chaque travail J_j une date de fin effective C_j et, étant donnée une date de fin souhaitée d_j, il définit un retard algébrique $L_j = C_j - d_j$ $(1 \leq j \leq n)$, un retard absolu $T_i = \max\{0, L_i\}$ ou un indicateur de retard $U_i = 1_{\text{si } C_i > d_i}$. On peut aussi effecter une pondération w_j à chaque travail et obtenir ainsi diverses fonctions objectifs :
  - $C_{\max} = \max\{C_1, C_2, \ldots, C_n\}$, c'est-à-dire la date de fin du dernier travail traité dans l'ordonnancement (aussi appelé *makespan*) ;
  - $\sum C_j = C_1 + C_2 + \cdots + C_n$ qui est le temps de traitement total ;
  - $L_{\max} = \max\{L_1, L_2, \ldots, L_n\}$ qui est le maximum des retards algébriques ;
  - $T_{\max} = \max\{T_1, T_2, \ldots, T_n\}$ qui est le maximum des retards absolus ;
  - $\sum T_i = T_1 + T_2 + \cdots + T_n$ qui est la somme des retards absolus ;
  - $\sum w_i T_i = w_1 T_1 + w_2 T_2 + \cdots + w_n T_n$ qui représente la somme pondérée des retards absolus ;
  - $\sum U_i = U_1 + U_2 + \cdots + U_n$ qui représente le nombre de travaux en retard ;
  - $\sum w_i U_i = w_1 U_1 + w_2 U_2 + \cdots + w_n U_n$ qui représente le nombre pondéré de travaux en retard.

Si S_1 et S_2 sont deux séquences de travaux, nous noterons $S_1 \cdot S_2$ la concaténation de ces travaux.

1.4 Algorithmique exponentielle

Face à un problème NP-difficile, il est improbable d'obtenir un algorithme polynomial pour le résoudre, sauf si $P = NP$. Diverses approches ont été développées pour affronter ces problèmes. Bien souvent, elles relaxent l'un des critères suivants :

- résoudre le problème de façon exacte ;

- résoudre le problème pour n'importe quelle entrée ;
- résoudre le problème efficacement.

La relaxation d'un ou plusieurs de ces critères a donné naissance à plusieurs grands domaines de l'algorithmique (algorithmes modérément exponentiels, d'approximation, probabilistes, à paramètre fixé, heuristiques, ...). Dans la suite de ce manuscrit, nous cherchons à résoudre des problèmes NP-complets grâce à des algorithmes exacts dont le temps d'exécution est exponentiel. Au-delà de l'aspect algorithmique important en soi, ces algorithmes trouveront une utilisation pratique au moins pour des entrées de taille modérée et dépendant essentiellement de la base de l'exponentielle que nous cherchons à minimiser. Pour cela, nous développons un certain nombre de techniques pour concevoir et analyser les algorithmes.

Nous donnons ci-après une description de quelques unes des principales techniques de l'algorithmique exponentielle. Certaines d'entre elles ne lui sont pas propres et sont également utilisées pour la conception d'algorithmes à paramètre fixé (*compression itérative*, *branchement*) ou polynomiaux (*programmation dynamique*, *diviser-pour-régner*).

Nous commençons par présenter la technique *brancher-et-réduire*. Elle nous sera utile au chapitre 2. C'est une technique puissante car elle permet bien souvent d'établir des temps d'exécution de la forme $\mathcal{O}^*(c^n)$ avec $c < 2$, en espace polynomial. Cela en fait une technique également applicable en pratique pour des implémentations, éventuellement couplées avec du parallèlisme. La prochaine section est extraite de la référence ci-dessous, que nous avons eu l'occasion d'écrire récemment pour l'école jeunes chercheurs du GDR-IM.

> **(Lenté *et al.*, 2015)** LENTÉ, C., LIEDLOFF, M., T'KINDT, V. et TODINCA, I. (2015). Algorithmes modérément exponentiels pour problèmes NP-difficiles. *In* OLLINGER, N., editor: *Informatique Mathématique une photographie en 2015*. CNRS Editions

1.4.1 Brancher-et-réduire, mesurer-pour-conquérir et mémorisation

Les algorithmes de *branchement* sont l'un des outils fondamentaux pour la résolution de problèmes en temps exponentiels. Le *branchement* est certainement la technique la plus ancienne, la plus puissante, mais également la plus naturelle à utiliser. Les premiers algorithmes de *branchement* datent des années 1960 avec les travaux de Davis et Putnam (1960) et de Davis *et al.* (1962), pour la résolution du problème SAT.

> **SAT**
>
> ***Entrée.*** Une formule logique propositionnelle $\mathcal{F}$ en forme normale conjonctive.
>
> ***Question.*** Décider si la formule $\mathcal{F}$ est satisfiable.

L'emploi de cette technique aboutit souvent à des algorithmes dont le temps d'exécution est, certes exponentiel, mais significativement plus faible que celui de la simple énumération naïve de l'espace des solutions. Typiquement, les algorithmes de *branchement* ne nécessitent qu'un espace polynomial, ce qui les rend utilisables en pratique.

Ces algorithmes sont utiles à la résolution de problèmes de décision, d'optimisation, de dénombrement et d'énumération. Évidement, le cadre le plus général est l'énumération de l'ensemble des solutions, puisque cette énumération résout le dénombrement de ces solutions, mais également le problème d'optimisation ou de décision correspondant.

Afin d'étudier quelques exemples d'algorithmes exponentiels, considérons la notion d'ensemble stable d'un graphe. Étant donné un graphe $G = (V, E)$, nous rappelons qu'un sous-ensemble de sommets $S \subseteq V$ est stable s'il n'existe aucune arête entre deux sommets de S : $\forall u, v \in S, \{u, v\} \notin E$. Un exemple est donné à la figure 1.4.

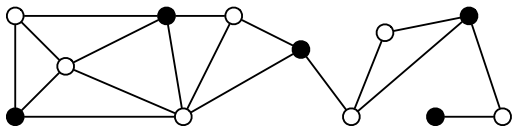


FIGURE 1.4 – Un graphe où les sommets colorés en noir forment un ensemble stable de taille maximum.

Regardons les trois problèmes suivants :

MAX-ENSEMBLE STABLE

Entrée. Un graphe $G = (V, E)$.

Question. Déterminer un stable de taille maximum de G.

ENUM-ENSEMBLES STABLES

Entrée. Un graphe $G = (V, E)$.

Question. Énumérer les stables maximaux par l'inclusion de G.

#-ENSEMBLE STABLE

Entrée. Un graphe $G = (V, E)$.

Question. Ddénombrer les stables maximaux par l'inclusion G.

Il a été montré que le problème MAX-ENSEMBLE STABLE est NP-complet (formellement, c'est le cas pour le problème de décision qui lui est associé), W[1]-complet (Downey et Fellows, 1999) et, si P ≠ NP, non approchable avec un facteur constant (Berman et Fujito, 1999). Par ailleurs, il n'admet pas d'algorithme sous-exponentiel sous l'*hypothèse du temps exponentiel* (Impagliazzo *et al.*, 2001). Le problème #-ENSEMBLE STABLE est quant à lui #P-complet (Okamoto *et al.*, 2008). On sait le résoudre par un algorithme de *branchement* en temps $\mathcal{O}(1.3642^n)$ (Gaspers *et al.*, 2012). Concernant l'énumération des stables maximaux, un résultat de Moon et Moser (1965) montre que leur nombre est au plus $3^{n/3}$; en le combinant avec un algorithme à délai polynomial (comme celui de Johnson *et al.* (1988)), il est ainsi possible de résoudre ENUM-ENSEMBLES STABLES en temps $\mathcal{O}^*(3^{n/3})$.

Plusieurs algorithmes exponentiels ont été développés pour MAX-ENSEMBLE STABLE. C'est un problème emblématique de l'algorithmique modérément exponentielle et pour lequel de nombreux

algorithmes de *branchement* ont été développés. Parfois ces algorithmes combinent le *branchement* avec une technique de *mémorisation* (que nous présenterons par la suite) pour diminuer le temps d'exécution, au prix d'un espace exponentiel. Le premier algorithme est dû à Tarjan et Trojanowski (1977) et s'exécute en temps $\mathcal{O}(1.26^n)$ (et espace polynomial) ; les meilleurs algorithmes connus à ce jour (qui utilisent également un espace polynomial) sont celui de Bourgeois *et al.* (2012) qui s'exécute en temps $\mathcal{O}(1.2114^n)$, ainsi que plus récemment celui de Xiao et Nagamochi (2013) qui s'exécute en temps $\mathcal{O}(1.2002^n)$ (la borne de $\mathcal{O}(1.1996^n)$ est même annoncée par ces auteurs).

Algorithmes d'énumération pour les stables maximaux

▸ L'algorithme

Commençons par donner une caractérisation des algorithmes de *branchement* ainsi que des notions sur l'étude de leur complexité en temps qui, nous le verrons, n'est pas toujours triviale. Typiquement, un algorithme de *branchement* est un algorithme récursif composé de règles :

- d'*arrêt*, pour stopper la récursion ;
- de *réduction*, pour simplifier ou réduire la taille de l'instance ;
- de *branchement*, pour résoudre le problème en résolvant récursivement des instances plus petites.

Regardons un exemple d'algorithme de *branchement* pour ENUM-ENSEMBLES STABLES (voir l'algorithme 1). Soit $G = (V, E)$ un graphe et soit $S \subseteq V$ un ensemble stable de G. Notons $\tilde{G}$ une copie de G. À chaque appel récursif à l'algorithme **EnumEnsStables**, le graphe G est modifié (essentiellement, nous lui supprimons des sommets et des arêtes), par contre nous ne modifions pas la copie $\tilde{G}$. Ainsi, à chaque instant, le graphe G est un sous-graphe de $\tilde{G}$. Un appel à **EnumEnsStables**$(G, \varnothing)$ (où initialement $\tilde{G} = G$) provoque l'énumération de tous les ensembles stables maximaux du graphe G.

Algorithme 1 : EnumEnsStables(G, S)

Entrée : Un graphe G et un ensemble stable S de $\tilde{G}$, où G ne contient aucun sommet de $N[S]$.
Sortie : L'union de S avec les ensembles stables maximaux de G.

si *G est vide* **alors**
  retourner « S est un ensemble stable maximal de $\tilde{G}$ »

choisir un sommet v de degré minimum dans G
pour tous les *sommets w de $N_G[v]$* **faire**
  EnumEnsStables$(G - N_G[w], S \cup \{w\})$

Remarque 1. *Notons que l'algorithme* ***EnumEnsStables*** *peut énumérer plusieurs fois un même ensemble stable maximal. Pour s'en prémunir, on pourrait légèrement modifier notre algorithme et considérer l'algorithme* ***EnumEnsStablesSansDoublon*** *présenté ci-après (voir l'algorithme 2). Néanmoins, pour ne pas compliquer inutilement notre première analyse de complexité, nous allons nous restreindre dans la suite à l'étude de l'algorithme* ***EnumEnsStables****. On peut montrer que la complexité de* ***EnumEnsStablesSansDoublon*** *au pire des cas serait similaire à celle de* ***EnumEnsStables****.*

Algorithme 2 : EnumEnsStablesSansDoublon(G, S)

Entrée : Un graphe G et un ensemble stable S de $\tilde{G}$, où G ne contient aucun sommet de $N[S]$.
Sortie : L'union de S avec les ensembles stables maximaux de G.

si *G est vide et S est un stable maximal de $\tilde{G}$* **alors**
  retourner « S est un ensemble stable maximal de $\tilde{G}$ »

choisir un sommet v de degré minimum dans G
soit $w_0, w_1, \ldots, w_{d(v)}$ les sommets de $N_G[v]$
pour *i de 0 à $d(v)$* **faire**
  EnumEnsStablesSansDoublon($G - (N[w_i] \cup \{w_0, w_1, \ldots w_i\})$, $S \cup \{w_i\}$)

▸ Son temps d'exécution

L'analyse du temps d'exécution d'un tel algorithme récursif se fait à travers l'étude de récurrences. On peut établir une récurrence qui dénombre les sous-problèmes qui sont récursivement résolus. Comme l'algorithme est exponentiel, il résout typiquement de l'ordre de $\mathcal{O}(\alpha^n)$ sous-problèmes, pour une certaine constante α à déterminer. En observant que chaque exécution de **EnumEnsStables** demande un temps polynomial, si l'on exclut les appels récursifs, on en déduit que le temps d'exécution de **EnumEnsStables** est $\mathcal{O}(\text{poly}(n) \cdot \alpha^n) = \mathcal{O}^*(\alpha^n)$.

Regardons l'algorithme **EnumEnsStables**. Notons $T(n)$ le temps d'exécution de l'algorithme sur un graphe à n sommets. L'algorithme choisit un sommet v et provoque un appel récursif sur $G - N[w]$, pour chaque $w \in N[v]$. Notons $d(u)$ le degré d'un sommet u. La récurrence peut s'écrire :

$$T(n) = 1 + \sum_{w \in N_G[v]} T(n - (d(w) + 1)).$$

Comme v est un sommet de degré minimum, $d(w) \geq d(v)$ pour tout w ; donc :

$$T(n) \leq 1 + \sum_{w \in N_G[v]} T(n - (d(v) + 1)) = 1 + (d(v) + 1) \cdot T(n - (d(v) + 1)).$$

Notons $x = d(v) + 1$. On obtient

$$T(n) \leq 1 + x \cdot T(n - x) = 1 + x + x^2 + \cdots + x^{n/x} \leq \frac{n}{x} \cdot x^{n/x} = \mathcal{O}^*(x^{n/x}).$$

Observons que dénombrer les feuilles de l'arbre de recherche, correspondant à l'exécution de cet algorithme, aurait abouti à la récurrence $T(n) \leq x \cdot T(n - x)$ dont la solution est également $\mathcal{O}^*(x^{n/x})$. Comme le montre la figure 1.5, la fonction $x^{n/x}$ est maximale pour l'entier $x = 3$. On obtient alors $T(n) = \mathcal{O}^*(3^{n/3})$.

▸ Borne inférieure

On peut montrer que le temps d'exécution de **EnumEnsStables** est asymptotiquement optimal, puisqu'il existe des graphes qui possèdent $3^{n/3}$ ensembles stables maximaux (voir la figure 1.6). Comme nous le verrons par la suite, on ne sait pas toujours établir une borne inférieure égale à la borne supérieure. Il est même assez fréquent que les bornes supérieures sur le temps d'exécution soient sur-estimées.

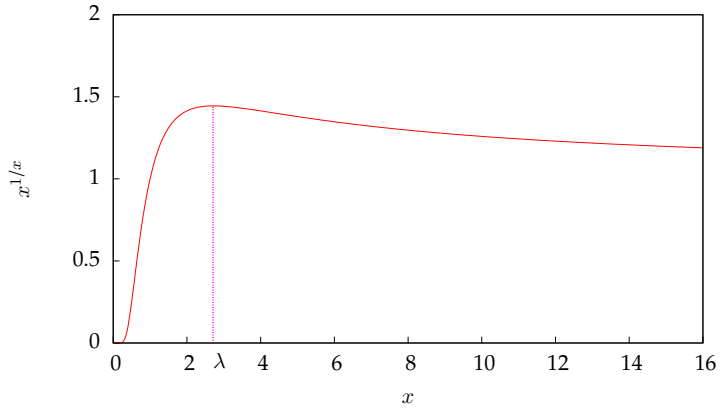


FIGURE 1.5 – La fonction $x^{1/x}$ atteint son maximum pour $\lambda \approx 2.71$.

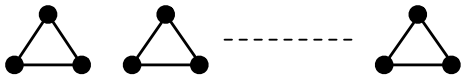


FIGURE 1.6 – Une collection de triangles possède $3^{n/3}$ ensembles stables maximaux.

▸ Conséquence combinatoire

Les algorithmes d'énumération sont des outils intéressants pour prouver des bornes combinatoires, puisqu'une borne supérieure sur leur temps d'exécution donne une borne combinatoire sur le nombre d'objets énumérés. L'algorithme **EnumEnsStables** et son analyse impliquent que tout graphe contient $\mathcal{O}^*(3^{n/3})$ ensembles stables maximaux.

À la recherche du plus grand stable

▸ L'algorithme

Intéressons-nous à présent au problème MAX-ENSEMBLE STABLE et à sa résolution par l'algorithme **StableMax** (voir l'algorithme 3). Soit $G = (V, E)$ un graphe et soit $S \subseteq V$ un ensemble stable de G. Notons $\tilde{G}$ une copie de G. À chaque appel récursif à **StableMax**, le graphe G passé en paramètre est un sous-graphe de $\tilde{G}$. Un appel à **StableMax**$(G, \varnothing)$ (où initialement $\tilde{G} = G$) retourne un ensemble stable de taille maximum du graphe G.

La preuve de correction de cet algorithme n'est pas bien difficile. Si un sommet est de degré 0 ou 1, il est toujours correct de l'ajouter au stable S (en effet, si un sommet v est de degré 1, dans tout ensemble stable S contenant l'unique voisin w de v, on peut remplacer w par v, en obtenant ainsi un nouvel ensemble stable de même taille que S). Si le graphe est de degré maximum 2, alors le graphe est une collection disjointe de cycles et de chemins, pour laquelle il est facile de calculer un stable maximum en temps polynomial. L'unique règle de branchement est triviale : si v est un sommet du graphe, alors soit il appartient au stable maximum (auquel cas on supprime $N[v]$ du graphe), soit v n'y appartient pas (on supprime v du graphe).

Algorithme 3 : `StableMax(`G`,` S`)`

Entrée : *Un graphe G et un ensemble stable S de $\tilde{G}$, où G ne contient aucun sommet de $N_{\tilde{G}}[S]$.*
Sortie : *Un ensemble stable S' de taille maximum de $\tilde{G}$ tel que $S \subseteq S'$.*

si *G est vide* **alors**
  retourner S
si *G contient un sommet v t.q. $d(v) \leq 1$* **alors**
  retourner StableMax($G - N[v]$**,** $S \cup \{v\}$**)**
choisir un sommet v de degré maximum
si $d(v) = 2$ **alors**
  `/* Observez que dans ce cas le graphe` G `est une collection disjointe de cycles et de chemins, pour laquelle un stable maximum peut être construit en temps polynomial */`
  retourner en temps polynomial le plus grand stable de G
sinon
  retourner $max\big($**StableMax**$(G - N[v], S \cup \{v\})$; **StableMax**$(G - v, S)\big)$

▸ Son temps d'exécution

Notons par $T(n)$ le temps d'exécution de cet algorithme sur un graphe contenant n sommets. On obtient la récurrence suivante, où $d(v) \geq 3$:

$$T(n) \leq T(n - (1 + d(v))) + T(n - 1).$$

Cette récurrence compte le nombre de feuilles dans l'arbre de recherche (ce qui est grossièrement équivalent à s'intéresser au nombre total de noeuds dans l'arbre, et donc de sous problèmes récursivement résolus). Pour une récurrence de la forme $T(n) \leq T(n-t_1) + T(n-t_2) + \cdots + T(n-t_r)$, on dit que $(t_1, t_2, \ldots, t_r)$ est le *vecteur de branchement* de cette récurrence. Pour la récurrence $T(n) \leq T(n-(1+d(v))) + T(n-1)$, son vecteur de branchement est donc par définition $(1+d(v), 1)$. Comme l'indique le théorème suivant, la solution d'une récurrence peut être obtenue par la recherche des racines du polynôme $x^n - x^{n-t_1} - x^{n-t_2} - \cdots - x^{n-t_r} = 0$, appelé *polynôme caractéristique* associé à la récurrence $T(n)$.

Théorème 1.1 (voir Fomin et Kratsch (2010))**.** *Soit b une règle de branchement dont le vecteur de branchement est $(t_1, t_2, \ldots, t_r)$. Alors le temps d'exécution de l'algorithme de* branchement *n'utilisant que la règle b est $\mathcal{O}^*(\alpha^n)$, où α est l'unique racine positive de $x^n - x^{n-t_1} - x^{n-t_2} - \cdots - x^{n-t_r} = 0$.*

Ainsi, nous devons résoudre la récurrence $T(n) \leq T(n - (1 + d(v))) + T(n - 1)$ pour toutes les valeurs possibles de $d(v)$, où $d(v) \geq 3$. On obtient alors les polynômes $x^n - x^{n-(1+d(v))} - x^{n-1} = 0$, pour chaque valeur de $d(v)$, dont il faut déterminer la racine positive. La plus grande de ces racines est obtenue pour le polynôme correspondant à la récurrence $T(n) \leq T(n-4) + T(n-1)$, c'est-à-dire lorsque $d(v) = 3$. La recherche de la racine positive (comprise pour ce polynôme entre 1 et 2) peut s'effectuer, par exemple, avec le logiciel `Sage` (`http://www.sagemath.org/`) et nous permet d'établir un temps d'exécution de $\mathcal{O}(1.3803^n)$ pour l'algorithme **StableMax**. Voici la commande utilisée dans `Sage` :

```
sage: find_root(x^(-4)+x^(-1)==1, 1,2)
1.3802775690976141
```

▸ Mesurer-pour-conquérir

On pourrait se demander s'il existe des graphes pour lesquels l'algorithme **StableMax** demanderait effectivement un temps de $\mathcal{O}(1.3803^n)$. Malgré la simplicité de cet algorithme, on peut montrer que notre analyse n'est pas optimale.

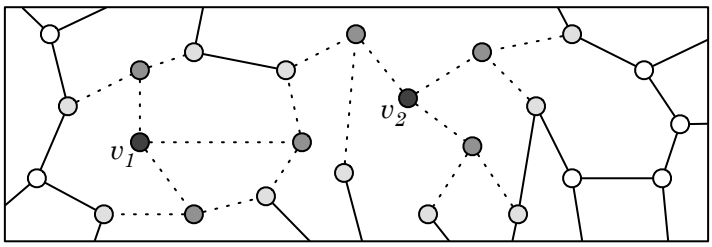


FIGURE 1.7 – Un morceau de graphe où l'on aurait commencé à brancher sur les sommets v_1 et v_2 pour les ajouter au stable.

On peut observer sur la figure 1.7 que le branchement sur des sommets v de degré au moins 3 laisse apparaître (suite à la suppression de v ou de $N[v]$) de nombreux sommets de degré inférieur. Pour ces sommets de degré 0, 1 ou 2, il n'y a pas de branchement, mais application de règles de réduction. D'une certaine façon, la suppression de sommets (comme par exemple v_1 et v_2 sur la figure 1.7) est favorable au temps d'exécution, mais la diminution du degré des autres sommets non supprimés l'est également. En effet, il ne sera plus nécessaire d'appliquer une règle de branchement sur ces sommets de degré strictement inférieur à 3 ; à un moment de l'exécution de l'algorithme, ces sommets seront soit facilement traités par une règle de réduction, soit supprimés par l'ajout à l'ensemble stable de l'un de leurs voisins. Ainsi, plutôt que de ne considérer dans la récurrence que le nombre de sommets présents dans le graphe, une mesure plus fine, prenant en compte le degré des sommets, aboutirait à une borne plus précise. Nous détaillons cette approche dans la suite et refaisons l'analyse de la complexité de l'algorithme.

Soit N_2 (respectivement, N_3) le nombre de sommets de degré 2 dans G (respectivement, de degré 3) et soit $N_{\geq 4}$ le nombre de sommets de degré au moins 4 dans G. Soient w_2, w_3 et $w_{\geq 4}$ des réels de l'intervalle $[0, 1]$. Soit $\mu(G)$ la mesure définie par :

$$\mu(G) = w_2 N_2 + w_3 N_3 + w_{\geq 4} N_{\geq 4}.$$

Refaisons l'analyse en notant $T(\mu)$ le temps d'exécution de l'algo sur un graphe de mesure $\mu(G)$.

Soit v le sommet choisi par l'algorithme. Notons respectivement d_2, d_3, d_4, $d_{\geq 5}$ le nombre de ses voisins de degré 2, 3, 4 et ≥ 5,

- si $d(v) = 3$:

$$\begin{aligned} T(\mu) \quad \leq \quad & T\big(\mu - w_3 - d_2 w_2 - d_3 w_3\big) \quad + \\ & T\big(\mu - w_3 - d_2 w_2 - d_3(w_3 - w_2)\big) \end{aligned}$$

- si $d(v) = 4$:

$$\begin{aligned} T(\mu) \quad \leq \quad & T\big(\mu - w_{\geq 4} - d_2 w_2 - d_3 w_3 - d_4 w_{\geq 4}\big) \quad + \\ & T\big(\mu - w_{\geq 4} - d_2 w_2 - d_3(w_3 - w_2) - d_4(w_{\geq 4} - w_3)\big) \end{aligned}$$

- si $d(v) \geq 5$:

$$\begin{aligned} T(\mu) \quad \leq \quad & T\big(\mu - w_{\geq 4} - d_2 w_2 - d_3 w_3 - (d_4 + d_{\geq 5}) w_{\geq 4}\big) \quad + \\ & T\big(\mu - w_{\geq 4} - d_2 w_2 - d_3 (w_3 - w_2) - d_4 (w_{\geq 4} - w_3)\big). \end{aligned}$$

Il reste à calculer ces récurrences pour toutes les valeurs possibles $0 \leq d_2, d_3, d_4, d_{\geq 5} \leq 5$, puis à déterminer les valeurs des $w_i \in [0, 1]$ qui minimisent la plus grande solution. On observe que le choix des w_i implique $\mu(G) \leq n$.

Théorème 1.2. *L'algorithme **StableMax** calcule un ensemble stable maximum en temps $\mathcal{O}(1.2905^n)$, en prenant $w_2 = 0.5967$, $w_3 = 0.9287$ et $w_{\geq 4} = 1$ dans la mesure μ.*

Sans changer une ligne de l'algorithme, nous obtenons $\mathcal{O}(1.2905^n)$ comme borne de complexité, contre $\mathcal{O}(1.3803^n)$ pour l'analyse (plus classique) précédente. Pour établir cette nouvelle borne supérieure, nous avons utilisé une mesure non standard. Ce type d'approche a été introduit par Fomin *et al.* (2009c), sous le nom *mesurer-pour-conquérir*. L'emploi de mesures « non standards » est une contribution importante à (l'amélioration de) l'analyse au pire des cas des algorithmes de branchement. Ces mesures ne changent en rien le comportement de l'algorithme ; c'est simplement un raffinement de l'étude de sa complexité au pire des cas. Évidement, une autre mesure pourrait donner une meilleure borne supérieure. Il est donc naturel de se demander si la borne $\mathcal{O}(1.2905^n)$ est très éloignée de la borne supérieure optimale.

▸ Borne inférieure

Pour répondre à la question précédente, une borne inférieure sur le temps d'exécution de **StableMax** donnerait une estimation de la précision de notre mesure et de l'analyse de la borne supérieure (voir la figure 1.8).

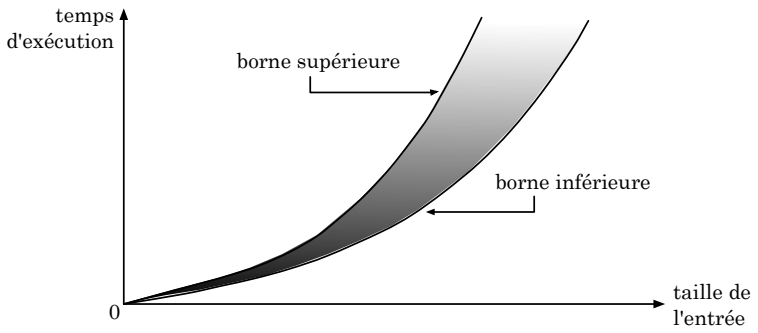


FIGURE 1.8 – Borne supérieure versus borne inférieure.

En étudiant le comportement de l'algorithme **StableMax** sur le graphe de la figure 1.9, on peut établir que son temps d'exécution est au moins $\Omega(1.2599^n)$.

Ce graphe, difficile pour l'algorithme, est constitué d'une collection disjointe de $\frac{n}{6}$ graphes complets à 6 sommets (appelé K_6). Considérons l'un de ces K_6 et notons x, y, z trois de ses sommets. L'algorithme va choisir de brancher sur un sommet de plus grand degré. Sans perte de généralité, supposons qu'il s'agisse de x. Lorsque x est ajouté à l'ensemble stable S, l'intégralité du K_6 est supprimé. Lorsque x n'est pas ajouté à S et que ce sommet est supprimé du graphe, il ne reste que 5 sommets du K_6. À un moment de l'exécution, l'algorithme choisit (sans perte de généralité) de brancher sur le sommet y. Là encore, soit $N[y]$ est supprimé (et il ne reste aucun sommet du

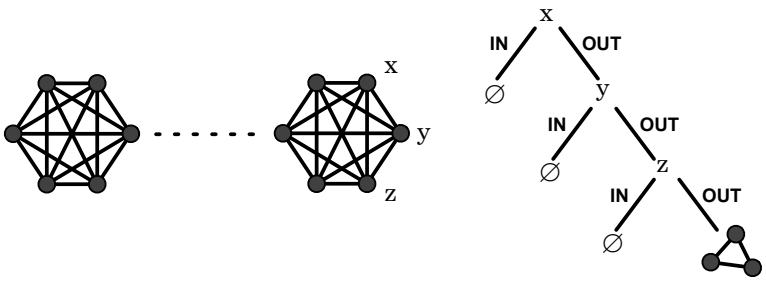


FIGURE 1.9 – Une collection de K_6 et un morceau de l'arbre de recherche correspondant à l'exécution de **StableMax** sur ce graphe.

K_6), soit y est supprimé du graphe (et il ne reste que 4 sommets du K_6). Dans ce dernier cas, il y aura un moment de l'exécution de l'algorithme où, sans perte de généralité, l'algorithme choisit de brancher sur le sommet z. Ce sommet z est soit ajouté à S (et tous les sommets restants du K_6 sont supprimés), soit z est supprimé du graphe (et il ne reste du K_6 qu'un cycle de trois sommets). Comme tous les sommets de ce cycle ont un degré égal à 2, l'algorithme détermine, en temps polynomial à la fin de son exécution, un stable maximum. Si on regarde l'arbre correspondant aux appels récursifs, chaque K_6 engendre donc 4 branches de cet arbre. Il y a $\frac{n}{6}$ graphes K_6, donc l'arbre contient au total au moins $4^{n/6}$ feuilles. On obtient ainsi le théorème suivant :

Théorème 1.3. *L'algorithme **StableMax** résout le problème* MAX-ENSEMBLE STABLE *en temps* $\Omega(4^{n/6})$, *ou encore* $\Omega(1.2599^n)$ *puisque* $\sqrt[6]{4} > 1.2599$.

▸ Au prix de l'espace

Si l'on s'autorise un espace exponentiel, il est possible de réduire davantage le temps d'exécution de certains algorithmes de *branchement*. Reprenons dans un premier temps l'analyse de **StableMax** avec la mesure standard qui avait donné $\mathcal{O}(1.3803^n)$ comme temps d'exécution.

Notons par $T_h(n)$ le nombre de sous-problèmes correspondant à des graphes à h sommets, qui sont récursivement résolus lorsque l'algorithme est appliqué à un graphe à n sommets ($n \geq h$). Puisque $T(n) = \mathcal{O}(1.3803^n)$, on peut établir que $T_h(n) = \mathcal{O}(1.3803^{n-h})$. Dans le même temps, il y a au plus $\binom{n}{h}$ sous-graphes induits à h sommets. Donc

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

On a égalité entre ces deux termes pour $h \approx 0.08652 \cdot n$, ce qui donne $T_h(n) \leq \mathcal{O}(1.3424^n)$.

On peut à présent définir l'algorithme **StableMaxExpSpace**. À chaque fois qu'il y a un appel récursif sur un graphe G', cet algorithme vérifie si le sous-problème a déjà été résolu. Si ce n'est pas le cas, il le résout comme le ferait **StableMax** pour trouver un ensemble stable maximum $S_{G'}$; puis il stocke dans une base de données le couple $(G', S_{G'})$. Si le sous-problème a déjà été résolu, l'algorithme **StableMaxExpSpace** recherche simplement G' dans la base de données pour retrouver son stable maximum $S_{G'}$. Comme un graphe G a au plus 2^n sous-graphes induits, la recherche dans la base de données peut être implémentée en $\mathcal{O}(\log(2^n)) = \mathcal{O}(n)$. Cette technique s'appelle la mémorisation.

Théorème 1.4. *L'algorithme **StableMaxExpSpace** calcule un ensemble stable maximum en temps et espace* $\mathcal{O}(1.3424^n)$.

Il est possible de combiner la *mémorisation* et *mesurer-pour-conquérir*. Reprenons la mesure du théorème 1.2 :

$$\mu(G) = w_2 N_2 + w_3 N_3 + w_{\geq 4} N_{\geq 4} = 0.5967 N_2 + 0.9287 N_3 + N_{\geq 4} \leq n.$$

On peut encore une fois noter que $T_h(n) = \mathcal{O}(1.2905^{\mu(G)-h})$. Seulement, on doit maintenant déterminer le nombre de sommets n' d'un graphe G', ayant comme mesure $\mu(G')$, correspondant à un sous-problème potentiellement stocké dans la base de données.

Les sous-problèmes considérés ne contiennent pas de sommets de degré 0 ou 1 (qui sont immédiatement ajoutés à la solution), chaque sommet a donc pour poids au moins $w_2 = 0.5967$. Par conséquent, $n' \leq h/0.5967$. Posons $h' = h/0.5967$, ainsi :

$$T_h(n) \leq \mathcal{O}\left(\min\left(1.2905^{n-h}, \sum_{i \leq h'} \binom{n}{i}\right)\right).$$

Après quelques calculs (voir par exemple le chapitre 10 de Fomin et Kratsch (2010)), on obtient le théorème suivant :

Théorème 1.5. *L'algorithme* ***StableMaxExpSpace****, analysé avec la mesure μ calcule un ensemble stable maximum en temps $\mathcal{O}(1.2775^n)$ et en espace exponentiel.*

Dénombrer plus vite qu'énumérer

Le problème #-ENSEMBLE STABLE demande de dénombrer les ensembles stables maximaux d'un graphe. Dès lors, on peut se demander s'il est possible de compter les stables, sans avoir besoin de les énumérer. Dans (Gaspers *et al.*, 2012), nous donnons un algorithme de *branchement* qui résout #-ENSEMBLE STABLE. Cet algorithme est composé de deux règles d'arrêt, de sept règles de réduction et d'une règle de branchement qui choisit les sommets sur lesquels brancher de façon judicieuse. L'étude d'une borne inférieure montre que le temps d'exécution de notre algorithme est au moins $\mathcal{O}(1.3247^n)$. Pour la borne supérieure, nous effectuons une analyse de type *mesurer-pour-conquérir*. L'étude de sous-cas et de 27 récurrences établit le théorème suivant :

Théorème 1.6 (Gaspers *et al.* (2012))**.** *Le problème* #-ENSEMBLE STABLE *peut être résolu en temps $\mathcal{O}(1.3642^n)$ et espace polynomial.*

1.4.2 Programmation dynamique

La *programmation dynamique* est une technique classique de conception d'algorithmes pour résoudre des problèmes en temps polynomial. Elle est traditionnellement enseignée dans les cours d'algorithmique et fait l'objet de chapitres dans des ouvrages tels que (Cormen *et al.*, 2009; Kleinberg et Tardos, 2006). L'idée générale est de résoudre un problème en utilisant des solutions à des sous-problèmes précédemment résolus. Pour ce faire, la *programmation dynamique* applique une approche dite « du bas vers le haut », c'est-à-dire qu'on commence par résoudre les sous-problèmes les plus petits, et donc les plus faciles, pour ensuite résoudre des problèmes de plus en plus grands, jusqu'à finalement déterminer une solution du problème initial. Pour obtenir un *bon* temps d'exécution, chaque sous-problème n'est résolu qu'au plus une fois, et lorsque la solution d'un sous-problème est calculée, elle est conservée dans une structure de données. Les

algorithmes polynomiaux obtenus en employant cette technique sont nombreux et nous en avons déjà cité quelques-uns en début de section ; nous verrons ci-après que la technique peut aussi être utilisée pour résoudre des problèmes NP-difficiles et ainsi obtenir des algorithmes exponentiels. Bien souvent, comme la *programmation dynamique* nécessite de stocker les solutions de tous les sous-problèmes résolus, il est également nécessaire de disposer d'un espace exponentiel.

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

VOYAGEUR DE COMMERCE

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

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

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

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

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

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

$$\mathrm{Opt}[\{v_i\}, i] = d(v_1, v_i);$$
$$\mathrm{Opt}[X, i] = \min_{j \in X \setminus \{v_i\}} \mathrm{Opt}[X \setminus \{i\}, j] + d(v_j, v_i).$$

Finalement la solution optimale est donnée par :

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

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

Également en utilisant la *programmation dynamique*, Lawler (1976) a proposé un algorithme exponentiel pour déterminer en temps $\mathcal{O}(2.4423^n)$ le *nombre chromatique* d'un graphe G à n sommets.

Le nombre chromatique d'un graphe G, noté $\chi(G)$, est le plus petit nombre de sous-ensembles stables en lesquels les sommets de G peuvent être partitionnés. Le meilleur algorithme connu à ce jour pour calculer $\chi(G)$ s'exécute en temps $\mathcal{O}^*(2^n)$ et est du à Björklund *et al.* (2009). Cet algorithme utilise à la fois de la *programmation dynamique* comme une étape de pré-traitement, puis l'application d'une formule de type *Inclusion-Exclusion*. Comme exemple supplémentaire, nous expliquons dans la suite l'approche de Lawler (1976).

COLORATION

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

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

L'algorithme utilise plusieurs résultats bien connus. Le premier est que pour tout graphe G, il existe une coloration optimale pour laquelle une classe de couleur, c'est-à-dire un ensemble de sommets coloriés par une même couleur, induit un ensemble stable maximal par inclusion. Le deuxième résultat utilisé provient de l'article Moon et Moser (1965) qui indique qu'un graphe à n sommets contient au plus $3^{n/3}$ ensembles stables maximaux par inclusion. De plus, il existe des algorithmes à délai polynomial qui permettent d'énumérer ces ensembles comme par exemple celui de Johnson *et al.* (1988).

Étant donné un graphe $G = (V, E)$, l'algorithme de Lawler (1976) calcule pour chaque sous-ensemble $X \subseteq V$ la valeur de $\mathrm{Opt}[X]$ égale à $\chi(G[X])$. En considérant les ensembles X par ordre de cardinalité croissante, cette valeur est calculée de la façon suivante :

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

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

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

1.4.3 Convolution

Calculer le produit de convolution de deux fonctions peut s'avérer utile pour améliorer la complexité de certaines approches, notamment des approches de *programmation dynamique*. Björklund *et al.* (2007) ont proposé un algorithme *rapide* pour calculer ce produit. Dans la suite, nous présentons les idées principales de leur approche (voir aussi le chapitre 7 de l'ouvrage de Fomin et Kratsch (2010)).

Soit V un ensemble à n éléments. Notons par 2^V l'ensemble de tous les sous-ensembles de V. Étant données deux fonctions $f, g : 2^V \to \mathbb{Z}$, leur produit de convolution, noté $f * g$, est défini pour tout ensemble $S \subseteq V$ par

$$(f * g)(S) = \sum_{T \subseteq S} f(T) g(S \setminus T).$$

Par une transformée et une inversion de Möbius, le produit de convolution peut être calculé en temps $\mathcal{O}^*(2^n)$, ce qui améliore grandement l'approche naïve en temps $\mathcal{O}^*(3^n)$. On trouvera différentes applications de cette technique dans (Björklund *et al.*, 2007). Par exemple, les auteurs montrent que le temps d'exécution de l'algorithme de Dreyfus et Wagner (1971) pour le problème ARBRE DE STEINER peut être réduit à $\mathcal{O}^*(2^k n^2 M + nm \log M)$ où k est le nombre de terminaux et M le plus grand poids des arêtes.

Plus précisément, en suivant l'approche de Björklund *et al.* (2007), un calcul *rapide* du produit de convolution $f * g$ peut être réalisé en temps $\mathcal{O}^*(2^n)$ par l'algorithme suivant. Supposons que $\mathcal{U}$ soit un univers de taille n et que les fonctions f, g et h soient définies des sous-ensembles de $\mathcal{U}$ vers l'ensemble des entiers $\{-M, \ldots, M\}$, avec $M \leq 2^{n^{\mathcal{O}(1)}}$.

On commence par définir les *transformées de zeta par niveau* de f et g par

$$f\zeta(k, S) = \sum_{\substack{W \subseteq S \\ |W|=k}} f(W) \quad \text{et} \quad g\zeta(k, S) = \sum_{\substack{W \subseteq S \\ |W|=k}} g(W). \tag{1}$$

Par une approche de *programmation dynamique*, ces transformées de zeta par niveau des fonctions f et g peuvent être calculées en temps $\mathcal{O}^*(2^n)$, pour tous les sous-ensembles $S \subseteq \mathcal{U}$. Les détails que nous ne donnons pas ici sont accessibles dans (Björklund *et al.*, 2007; Fomin et Kratsch, 2010).

Ensuite, la *convolution par niveau* des transformées de zeta, définie par

$$(f\zeta \circledast g\zeta)(k, S) = \sum_{j=0}^{k} f\zeta(j, S) \cdot g\zeta(k - j, S), \tag{2}$$

est calculée en utilisant les valeurs des transformées de zeta par niveau. On note que la convolution par niveau est effectuée sur la variable k.

Finalement, le résultat obtenu est inversé par la *transformée de Möbius* où, étant donné une fonction h, sa transformée de Möbius, notée $h\mu$, est définie par

$$h\mu(S) = \sum_{X \subseteq S} (-1)^{|S \setminus X|} h(X). \tag{3}$$

Encore une fois, la transformée de Möbius peut être calculée en temps $\mathcal{O}^*(2^n)$ par *programmation dynamique* (voir Björklund *et al.* (2007); Fomin et Kratsch (2010)). La validité de cette approche est assurée par la formule suivante qui est démontrée par Björklund *et al.* (2007) :

$$(f * g)(S) = \sum_{X \subseteq S} (-1)^{|S \setminus X|} (f\zeta \circledast g\zeta)(|S|, X). \tag{4}$$

Björklund *et al.* (2007) montrent que cette approche peut parfaitement s'appliquer au calcul de la convolution sur le demi-anneau « min-somme » pour obtenir, dans le même temps d'exécution, le calcul de :

$$(f * g)(S) = \min_{T \subseteq S} \{f(T) + g(S \setminus T)\}.$$

1.4.4 Diviser-pour-régner

Nous présentons dans cette section le paradigme *Diviser-pour-Régner* pour la conception d'algorithmes exponentiels. L'idée générale consiste à partitioner l'instance en deux instances ayant

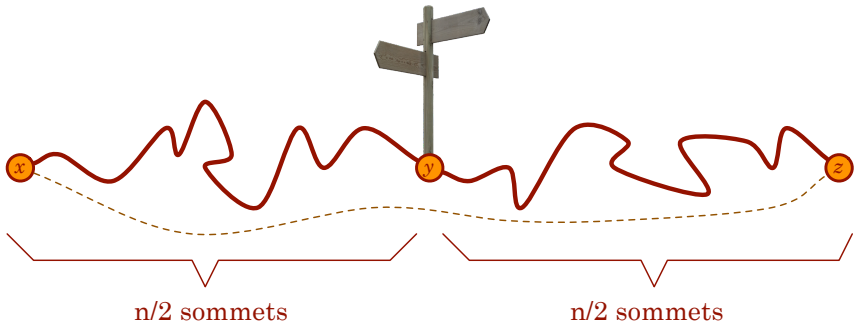


FIGURE 1.10 – Illustration de *diviser-pour-régner* appliquée au problème du CYCLE HAMILTONIEN.

comme taille (à peu près) la moitié de la taille de l'instance initiale. Ces deux instances sont ensuite résolues récursivement. Le *tri fusion* en est un exemple typique et habituellement nous obtenons des récurrences de la forme :

$$T(n) = n^{\mathcal{O}(1)} + 2 \cdot T(n/2).$$

Pour les algorithmes exponentiels, il est souvent indispensable de considérer toutes les partitions de l'instance en deux instances (avant de les résoudre récursivement) ; nous obtenons ainsi une récurrence de la forme :

$$T(n) = 2^n \cdot (n^{\mathcal{O}(1)} + 2 \cdot T(n/2)).$$

Regardons le problème du VOYAGEUR DE COMMERCE, ou plutôt la variante du cycle hamiltonien qui semble un peu plus facile, bien que ce problème soit également NP-complet :

CYCLE HAMILTONIEN

Entrée. Un graphe $G = (V, E)$.

Question. calculer un cycle passant exactement une fois par chaque sommet.

Étudions la stratégie suivante (voir aussi la figure 1.10) :

- Soit x et z tels que $\{x, z\} \in E$.
- S'il existe un cycle $\mathcal{C}$ dans le graphe, alors il existe un sommet $y \in \mathcal{C}$ qui sépare le chemin de x à z en deux sous-chemins disjoints ayant à peu près la même longueur.
- Ces sous-chemins peuvent être construits récursivement.

Cette stratégie aboutit à la récurrence suivante, où V désigne l'ensemble des sommets du graphe[1]

$$\texttt{Opt}[V, x, z] = \min_{\substack{V = S \uplus T \\ y \in V}} \big(\texttt{Opt}[S, x, y] + \texttt{Opt}[T, y, z] \big).$$

Ainsi, nous établissons le temps d'exécution :

$$\binom{n}{n/2} \cdot n \cdot 2 \cdot T(\frac{n}{2}) \quad = \quad \mathcal{O}^*(4^n \cdot n^{\log n}).$$

[1] La notation $V = S \uplus T$ indique que les sous-ensembles S et T forment une partition de l'ensemble V.

Pour calculer ce temps d'exécution, on observe que :

$$\begin{aligned} T(n) &\leq n \cdot 2^n \cdot T(n/2) \\ &\leq n^2 \cdot 2^n \cdot 2^{n/2} \cdot T(n/4) \\ &\leq n^{\log n} \cdot 2^n \cdot 2^{n/2} \cdot 2^{n/4} \ldots 2^0 \\ &\leq n^{\log n} \cdot 4^n. \end{aligned}$$

Comme la profondeur de la récursion est de l'ordre de $\log n$, le facteur n de la récurrence est au total majoré par $n^{\log n} = 2^{\log(n^{\log n})} = 2^{\log(n)^2} = 4^{\log(n)}$. D'où un temps d'exécution en $\mathcal{O}^*(4^{n+\epsilon})$, avec $\epsilon > 0$. On note que cet algorithme ne demande qu'un espace polynomial.

De la même façon, cette technique pourrait être utilisée pour résoudre le problème COLORATION en temps $\mathcal{O}^*(9^n)$. Au chapitre 2, nous l'utiliserons pour calculer une coloration un peu particulière : la $L(2,1)$-coloration.

1.4.5 Trier-et-chercher

Trier-et-chercher est une technique de conception assez peu employée dans le domaine des algorithmes exponentiels. De ce fait, la technique est largement méconnue. Elle date des années 70, lorsque Horowitz et Sahni (1974) ont proposé un algorithme pour améliorer la complexité de l'algorithme trivial de $\mathcal{O}^*(2^n)$ à $\mathcal{O}^*(2^{n/2})$ pour résoudre le problème du sac à dos (aussi parfois appelé sac à dos discret ou binaire) défini ci-après. En 1981, Schroeppel et Shamir (1981), toujours avec cette même technique, ont montré qu'un espace en $\mathcal{O}^*(2^{n/4})$ était suffisant pour résoudre ce problème en temps $\mathcal{O}^*(2^{n/2})$.

SAC-À-DOS

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

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

La technique peut se décrire de façon très générale. Supposons que l'on ait un certain problème à résoudre sur une instance I de taille n, et que I soit un ensemble (d'entiers, de sommets, ...). Une des idées fondamentales de *Trier-et-chercher* est de diviser le problème en deux sous-problèmes, comme le ferait une approche *diviser-pour-régner* classique. Contrairement à *diviser-pour-régner* qui chercherait à résoudre récursivement les sous-problèmes en appliquant à nouveau le même paradigme jusqu'à obtenir des problèmes de taille très petite, l'approche *Trier-et-chercher* ne divise qu'une seule fois pour obtenir deux problèmes I_1 et I_2 de taille $n/2$. Ensuite, on énumère tous les sous-ensembles $S \subseteq I_1$. À chaque sous-ensemble S est associé un vecteur de taille n dont le calcul se fait en temps polynomial et est fonction du problème considéré. Les vecteurs ainsi obtenus en temps $\mathcal{O}^*(2^{n/2})$ sont stockés dans une table T_1. On répète ce même processus pour tous les sous-ensembles de I_2 et l'on stocke les vecteurs correspondants dans une table T_2. De plus, les vecteurs de T_2 sont triés par ordre lexicographique en temps $\mathcal{O}(n2^{n/2}) = \mathcal{O}^*(2^{n/2})$ en utilisant un tri par base (en anglais, *Radix Sort*) ou même en temps $\mathcal{O}(2^{n/2} \log 2^{n/2}) = \mathcal{O}^*(2^{n/2})$ en

utilisant l'algorithme classique Tri Fusion. Cette étape est importante pour la suite de l'exécution de l'algorithme. Finalement, pour chaque vecteur $\vec{v_1}$ de la table T_1, on cherche s'il existe un vecteur $\vec{v_2}$ dans T_2 tel que la somme de $\vec{v_1}$ et $\vec{v_2}$ permette d'obtenir un vecteur objectif $\vec{o}$. La recherche d'un tel vecteur $\vec{v_2} \in T_2$, s'il existe, peut se faire en temps $\mathcal{O}(n)$ fois la longueur des vecteurs grâce à une recherche dichotomique (Knuth, 1997, p. 409).

Au chapitre 4, nous abstrairons un peu la méthode et la généraliserons. Comme application, nous montrerons la résolution de certains problèmes d'ordonnancement. Nous expliquons à présent l'emploi de cette technique pour résoudre le problème SOMME DE SOUS-ENSEMBLES.

> **SOMME DE SOUS-ENSEMBLES**
>
> ***Entrée.*** Un ensemble fini $A \subseteq \mathbb{N}$ d'entiers positifs et un entier positif W.
>
> ***Question.*** Existe-t-il un sous-ensemble $A' \subseteq A$ tel que $\sum_{a \in A'} a = W$?

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

1.4.6 ETH, l'hypothèse du temps exponentiel

Pour conclure ce chapitre, soulignons qu'un champ de recherche se veut complet s'il permet aussi d'établir des résultats négatifs. Le domaine des algorithmes modérément exponentiels s'est enrichi d'outils pour montrer l'impossibilité de résoudre certains problèmes au dessous d'une certaine complexité, sous certaines hypothèses. Par exemple, il est possible de montrer que certains problèmes (comme celui de la recherche du plus grand stable) n'admettent pas d'algorithmes dont le temps d'exécution serait en $\mathcal{O}(2^{o(n)})$, c'est-à-dire un temps sous-exponentiel, sauf si SNP $\subseteq$ SUBEXP. Cette dernière relation est considérée comme improbable. L'hypothèse du temps exponentiel (en anglais, *Exponential Time Hypothesis* ou ETH) est une conjecture selon laquelle le problème 3-SAT ne peut pas être résolu en temps sous-exponentiel. Notons que dans Impagliazzo *et al.* (2001), cette conjecture est donnée sous une forme un peu moins forte. Une autre hypothèse, encore plus forte, appelée *Strong Exponential Time Hypothesis* (SETH), va plus loin. Elle suppose que, pour tout $\delta < 1$, il existe un k tel que k-SAT ne puisse pas être résolu en temps $\mathcal{O}(2^{\delta n})$. Autrement dit, cette hypothèse implique que SAT ne peut pas être résolu en $\mathcal{O}(2^{\delta n})$ pour $\delta < 1$. On notera que la *Strong Exponential Time Hypothesis* implique la *Exponential Time Hypothesis*. Des résultats de (non) existence d'algorithmes (sous)-exponentiels reposent sur ces conjectures.

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

Dans les prochains chapitre, nous nous intéresserons à d'autres problèmes NP-complets. Nous commencerons par l'étiquetage de graphe, qui peut être vu comme un problème de coloration, puis à des problèmes de domination et finalement à des problèmes d'ordonnancement. Nous développerons et mettrons en œuvre, parfois de façon astucieuse, un certain nombre de techniques présentées dans ce chapitre.

Chapitre 2

Étiquetages $L(2,1)$

Dans ce chapitre, nous présentons des méthodes algorithmiques pour résoudre le problème de l'étiquetage $L(2,1)$ d'un graphe ainsi que des bornes combinatoires. Nous construisons des algorithmes de *branchement* pour décider si un graphe admet un étiquetage de largeur au plus 4 et pour énumérer les étiquetages de largeur 5 dans un graphe cubique. Pour la version d'optimisation où l'on calcule un étiquetage de largeur minimum dans un graphe, nous présentons des algorithmes dont la complexité ne dépend pas de la largeur de l'étiquetage, notamment un algorithme en temps $\mathcal{O}(2.6488^n)$ qui est le meilleur actuellement connu pour ce problème. Des études de bornes supérieures sur des objets combinatoires (étiquetage ou objets nécessaires à nos algorithmes) sont proposées, ainsi que quelques bornes inférieures sur la complexité de nos algorithmes. Ces algorithmes utilisent plusieurs techniques de conception (*brancher-et-réduire*, *diviser-pour-régner*, *programmation dynamique*) ainsi que pour l'analyse de leurs temps d'exécution (*récurrences*, *mesurer-pour-conquérir*).

Sommaire

2.1 Définition du problème et résultats connus

On s'intéresse dans ce chapitre à un problème d'affectation de fréquences dans un réseau, où l'on souhaite éviter les interférences entre les transmetteurs proches. Formellement, nous considérons un graphe $G = (V, E)$ simple et non orienté. On étiquette les sommets de G par des entiers de l'intervalle $\{0, 1, \ldots, k\}$, où k est appelé la largeur de l'étiquetage. Le problème de l'étiquetage $L(2,1)$ (ou coloration $L(2,1)$) requiert que la différence des étiquettes des sommets adjacents soit au moins 2 et que les sommets situés à distance 2 aient des étiquettes différentes. Étant donné un graphe, le problème d'optimisation demande de déterminer un étiquetage $L(2,1)$ de largeur minimum notée $\lambda_{2,1}(G)$.

ÉTIQUETAGE $L(2,1)$

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

Question. Existe-t-il une application f de V dans $\{0, 1, \ldots, k\}$ telle que :

1. $|f(u) - f(v)| \geq 2$ pour tout $\{u, v\} \in E$;
2. $|f(u) - f(v)| \geq 1$ (*i.e.* $f(u) \neq f(v)$) pour tout $u, v \in V$ tels que $N(u) \cap N(v) \neq \emptyset$.

Ce problème a été défini par Roberts (1988). Griggs et Yeh (1992) ont montré que le problème ÉTIQUETAGE $L(2,1)$ est NP-complet et Fiala *et al.* (2001) ont montré que ce problème est NP-complet pour tout k fixé où $k \geq 4$. Par conséquent, le problème n'admet pas d'algorithme à paramètre fixé (paramétré par la largeur k), sauf si $P = NP$. Lorsqu'on se restreint aux arbres, le problème peut être résolu en temps polynomial (Chang et Kuo, 1996) et Hasunuma *et al.* (2013) donnent même un algorithme linéaire. D'un autre côté, le problème est NP-complet sur les graphes séries-parallèles (Fiala *et al.*, 2008b). Par conséquent, le problème ÉTIQUETAGE $L(2,1)$ appartient à la poignée de problèmes qui séparent les graphes de largeur arborescente 1 et 2, selon une dichotomie « *polynomial* vs *NP-complet* ». Par ailleurs le problème ÉTIQUETAGE $L(2,1)$ est également *plus difficile* que le problème classique COLORATION (déterminer une coloration propre d'un graphe utilisant le plus petit nombre de couleurs) puisque celui-ci est polynomial sur les graphes séries-parallèles. Le problème ÉTIQUETAGE $L(2,1)$ est également NP-complet pour plusieurs classes de graphes et en particulier lorsqu'on se restreint aux graphes planaires, bipartis, splits et chordaux (Bodlaender *et al.*, 2004; Eggemann *et al.*, 2010).

Du point de vue structurel, Griggs et Yeh (1992) ont conjecturé que pour tout graphe $\lambda_{2,1}(G) \leq \Delta^2$, où Δ est le degré maximum de G. Cette conjecture est encore aujourd'hui largement ouverte et suscite beaucoup d'intérêts. Gonçalves (2008) a montré que $\lambda_{2,1}(G) \leq \Delta^2 + \Delta - 2$ en analysant un algorithme de Chang et Kuo (1996). Havet *et al.* (2008) ont réussi à démontrer la conjecture, pour Δ suffisamment grand (*i.e.* , pour des valeurs de Δ plus grandes que 10^{69}). Havet *et al.* (2012) montrent également que $\lambda_{2_1}(G) \leq \Delta^2 + C$, pour un certain entier C. D'un autre côté, on obtient aisément la borne inférieure $\lambda_{2,1}(G) \geq \Delta + 1$.

Une généralisation du problème ÉTIQUETAGE $L(2,1)$ est donnée par le problème AFFECTATION DE FRÉQUENCES (en anglais, CHANNEL ASSIGNMENT) :

AFFECTATION DE FRÉQUENCES

Entrée. Un graphe $G = (V, E)$, une fonction $w : E \to \mathbb{N}$ et un entier $k \in \mathbb{N}$.

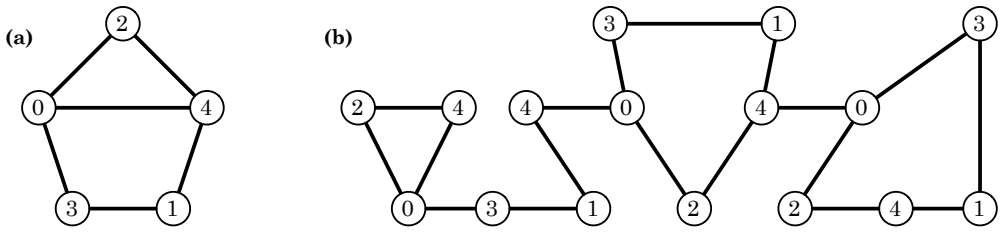


FIGURE 2.1 – (a) Le graphe $\overline{P_5}$. (b) Un graphe G avec un étiquetage $L(2,1)$ de largeur 4 qui correspond à un homomorphisme localement injectif vers le $\overline{P_5}$.

> ***Question.*** Existe-t-il une application f de V dans $\{0,1,\ldots,k\}$ telle que $|f(u) - f(v)| \geq w(u,v)$ pour tout $\{u,v\} \in E$?

Ce problème généralise à la fois le problème COLORATION (pour lequel la fonction w affecte un poids 1 à chaque arête du graphe donné en entrée) et le problème ÉTIQUETAGE $L(2,1)$ (pour lequel la fonction utilise des poids 1 ou 2 selon que les arêtes apparaissent dans le graphe G donné en entrée ou dans G^2). Král (2005) propose un algorithme en temps $\mathcal{O}(n(\ell+2)^n)$ pour résoudre le problème AFFECTATION DE FRÉQUENCES lorsque les valeurs de w sont bornées par ℓ. En utilisant un principe d'*inclusion-exclusion*, Cygan et Kowalik (2011) résolvent ce problème en temps $\mathcal{O}^*((\ell+1)^n)$, avec comme conséquence la résolution de ÉTIQUETAGE $L(2,1)$ en temps $\mathcal{O}(3^n)$. Ces algorithmes utilisent un espace exponentiel. Lorsqu'on s'autorise seulement un espace polynomial, un algorithme récent de Kowalik et Socala (2014) proposent de résoudre le problème en temps $\mathcal{O}^*((2\sqrt{\ell+1})^n)$ (et donc en temps $\mathcal{O}(3.4642^n)$ pour ÉTIQUETAGE $L(2,1)$, puisque $\ell = 2$ pour ce problème). En point d'orgue de ce chapitre, nous montrerons que le problème ÉTIQUETAGE $L(2,1)$ peut être résolu en temps $\mathcal{O}(2.6488^n)$ et espace exponentiel.

Les résultats autour du problème ÉTIQUETAGE $L(2,1)$ et de ses généralisations sont extrêmement nombreux ; nous n'en avons mentionnés ici qu'un nombre restreint. Calamoneri (2011) présente un état de l'art très complet, dont une version est régulièrement mise à jour sur son site web (celui-ci fait actuellement, une cinquantaine de pages). Le lecteur intéressé pourra donc s'y référer pour une description exhaustive des résultats de complexité, des bornes combinatoires, des résultats pour des classes de graphes, ainsi qu'une jolie collection de problèmes ouverts.

Le prisme des *homomorphismes localement injectifs* donne une autre vision du problème ÉTIQUETAGE $L(2,1)$. Donnons quelques explications. Soient deux graphes G et H. Une application $f : V(G) \to V(H)$ est un *homomorphisme* (on dit aussi plus simplement « morphisme ») de G dans H si pour toute arête $\{u,v\} \in E(G)$, on a $\{f(u), f(v)\} \in E(H)$. L'homomorphisme préserve donc les relations d'adjacence : pour deux sommets voisins de G, leurs images par f le sont encore dans H. L'homomorphisme $f : V(G) \to V(H)$ est *localement injectif* si, pour tout sommet $u \in V(G)$, son voisinage est associé de façon injective au voisinage de $f(u)$. Autrement dit, chaque paire de sommets ayant un voisin commun dans G est associée à des sommets différents de H. La propriété suivante nous sera utile par la suite ; elle est illustrée à la figure 2.1.

Propriété 2.1 (Fiala et Kratochvíl (2002))**.** *Un étiquetage $L(2,1)$ de largeur k correspond précisément à un homomorphisme localement injectif vers le graphe complémentaire d'un chemin de longueur k (noté $\overline{P_{k+1}}$).*

Dans (Havet *et al.*, 2011) nous montrons le résultat suivant :

Théorème 2.2 (Havet *et al.* (2011), théorème 1). *Étant donnés un graphe H et un graphe G à n sommets, un homomorphisme localement injectif de G vers H peut être calculé en temps $\mathcal{O}^*((\Delta(H)-1)^n)$.*

Ce théorème et la propriété 2.1 ont une conséquence notable, que nous chercherons à améliorer dans la suite de ce chapitre :

Corollaire 2.3 (Havet *et al.* (2011), corollaire 2). *Le problème* ÉTIQUETAGE $L(2,1)$ *de largeur k peut-être décidé en temps $\mathcal{O}^*((k-2)^n)$. En particulier, décider si $\lambda_{2,1}(G) \leq 4$ peut-être résolu en temps $\mathcal{O}^*(2^n)$.*

Dans la suite de ce chapitre, nous allons successivement :

- construire des algorithmes exponentiels pour déterminer des étiquetages $L(2,1)$ de largeur fixée, avec notamment un algorithme en $\mathcal{O}(1.3006^n)$ pour déterminer si $\lambda_{2,1}(G) = 4$ et un algorithme en temps $\mathcal{O}(1.7990^n)$ pour énumérer les étiquetages de largeur 5 d'un graphe cubique ;
- construire des algorithmes exponentiels, à la fois en espace polynomial et exponentiel, pour déterminer un étiquetage $L(2,1)$ de largeur minimum d'un graphe quelconque, avec notamment un algorithme en $\mathcal{O}(2.6488^n)$;
- établir des bornes supérieures sur des objets combinatoires nécessaires à nos algorithmes.

Les résultats présentés dans ce chapitre sont extraits des publications suivantes :

> **(Havet *et al.*, 2011)** HAVET, F., KLAZAR, M., KRATOCHVÍL, J., KRATSCH, D. et LIEDLOFF, M. (2011). Exact Algorithms for $L(2,1)$-Labeling of Graphs. *Algorithmica*, 59(2):169–194
>
> **(Couturier *et al.*, 2013a)** COUTURIER, J., GOLOVACH, P. A., KRATSCH, D., LIEDLOFF, M. et PYATKIN, A. V. (2013a). Colorings with few Colors: Counting, Enumeration and Combinatorial Bounds. *Theory Comput. Syst.*, 52(4):645–667
>
> **(Junosza-Szaniawski *et al.*, 2013a)** JUNOSZA-SZANIAWSKI, K., KRATOCHVÍL, J., LIEDLOFF, M., ROSSMANITH, P. et RZAZEWSKI, P. (2013a). Fast exact algorithm for $L(2,1)$-labeling of graphs. *Theor. Comput. Sci.*, 505:42–54
>
> **(Junosza-Szaniawski *et al.*, 2013b)** JUNOSZA-SZANIAWSKI, K., KRATOCHVÍL, J., LIEDLOFF, M. et RZAZEWSKI, P. (2013b). Determining the $L(2,1)$-span in polynomial space. *Discrete Applied Mathematics*, 161(13-14):2052–2061

2.2 Étiquetage $L(2,1)$ de largeur fixée

Nous l'avons indiqué, pour tout $k \geq 4$ fixé, déterminer s'il existe un étiquetage $L(2,1)$ de largeur k est un problème NP-complet (Fiala *et al.*, 2001). Pour $k = 3$, le problème est trivialement polynomial

puisque dans ce cas, le graphe doit être une collection disjointe de chemins de longueur au plus 3 (*i.e.* de P_i où $i \leq 4$). Il est donc naturel, au moins dans un premier temps, de s'intéresser à la résolution algorithmique (en temps exponentiel) du problème ÉTIQUETAGE $L(2,1)$ pour quelques valeurs de k petites ($k = 4$ et $k = 5$). Dans la suite, nous présentons deux algorithmes :

- un algorithme qui détermine si $\lambda_{2,1}(G) = 4$ en temps $\mathcal{O}(1.3006^n)$;
- un algorithme qui énumère tous les étiquetages $L(2,1)$ de largeur 5 d'un graphe cubique en temps $\mathcal{O}(1.7990^n)$.

2.2.1 Déterminer un étiquetage de largeur 4

Nous présentons un algorithme de *branchement* composé de 5 règles. Certaines de ces règles demandent de « brancher » le problème courant en plusieurs sous-problèmes, et d'autres réduisent simplement le problème en remarquant que l'étiquetage de quelques sommets est suffisamment contraint pour les étiqueter immédiatement.

L'idée de l'algorithme est d'étiqueter un sous-ensemble connexe de sommets, de plus en plus grand, jusqu'à l'étiquetage complet du graphe. Ainsi, lorsqu'un sommet v est étiqueté, nous pouvons garantir qu'il a au moins un voisin déjà étiqueté et que celui-ci a lui-même déjà un voisin étiqueté. Cela limite les possibilités d'étiquetage du sommet v et sera bénéfique pour le temps d'exécution de l'algorithme. Pour amorcer ce processus d'étiquetage, nous choisissons arbitrairement deux sommets voisins. Nous considérons les (au plus 12) façons de les étiqueter, ce qui nous donne les sous-problèmes initiaux pour lesquels nous vérifions s'il est possible d'étendre l'étiquetage de ces deux sommets à un étiquetage complet du graphe.

Supposons que $G = (V, E)$ soit le graphe donné en entrée. Nous pouvons supposer que $\Delta(G)$ est au plus 3 (car dans le cas contraire $\lambda_{2,1}(G) > 4$). Dans la suite de la section, nous supposerons qu'une partie du graphe G est étiqueté et que nous cherchons à étendre l'étiquetage. Nous notons $f : X \to \{0,1,2,3,4\}$ cet étiquetage $L(2,1)$ partiel de G. On rappelle que $G[X]$ forme un sous-graphe connexe de G. Si à un moment de l'exécution, nous détectons qu'il est impossible d'étendre l'étiquetage courant du graphe, alors l'algorithme arrête la résolution de ce sous-problème.

2.2.1.1 Description de l'algorithme

Nous décrivons chacune des cinq règles. Les trois premières règles réduisent le problème en étiquetant immédiatement des sommets dont l'étiquetage est suffisamment contraint pour ne plus avoir de choix d'étiquettes. Les deux dernières règles sont des règles de branchement, qui essayent récursivement les extensions d'étiquetages possibles. Nous montrons dans la suite que ce sont des dernières règles qui sont responsables du temps d'exécution exponentiel de l'algorithme.

Règle 1 – extensions forcées

(a) Si u est un sommet non étiqueté dont le voisin étiqueté v a deux voisins étiquetés, alors les étiquettes valides de u sont déterminées de façon unique (selon v et ses voisins étiquetés) ;

(b) si u est un sommet non étiqueté dont le voisin v est étiqueté 1, 2 ou 3, alors, puisque v a un autre voisin étiqueté, l'étiquette de u est déterminée de façon unique (selon v et ses voisins étiquetés) ;

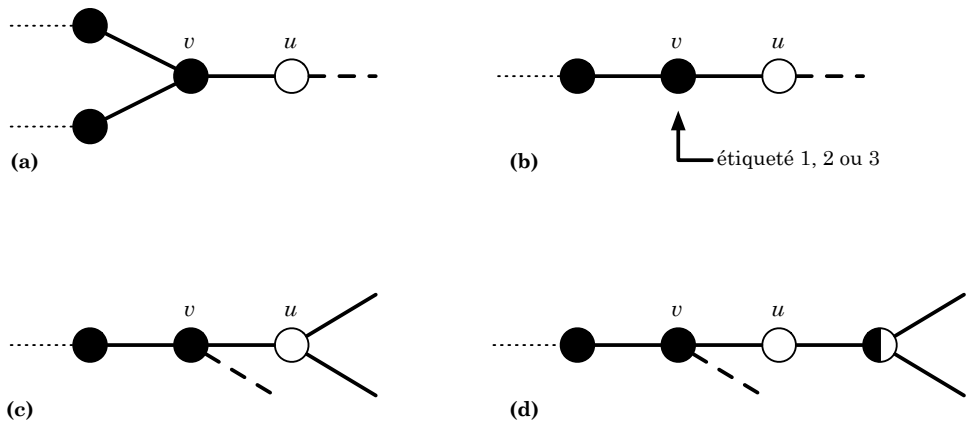


FIGURE 2.2 – Extensions forcées décrites par la règle 1. Les sommets noirs sont étiquetés. Dans chacun des cas, l'étiquette de u est déterminée de façon unique par son voisinage étiqueté.

(c) si u est un sommet non étiqueté de degré 3 avec un voisin v étiqueté, alors l'étiquette de u est soit 0, soit 4, déterminée de façon unique (selon v et son ou ses voisins étiquetés) ;

(d) si u est un sommet non étiqueté de degré 2 tel que l'un de ses voisins est étiqueté et l'autre voisin est (éventuellement non étiqueté) de degré 3, alors l'étiquette de u est déterminée de façon unique (selon les étiquettes de ses voisins).

La figure 2.2 représente les configurations possibles dans le cas des extensions forcées.

La preuve de correction de la règle d'extensions forcées est assez simple. Notons l'ensemble des étiquettes possibles d'un voisin d'un sommet étiqueté i par $N_i = \{j : 0 \leq j \leq 4 \text{ et } |j-i| \geq 2\}$, $0 \leq i \leq 4$.

Règle 1 (a). Supposons que v soit un sommet étiqueté ayant deux voisins étiquetés, notés v_1 et v_2, et un voisin u non étiqueté. Alors l'étiquette de v est 0 ou 4, et l'étiquette de u est nécessairement l'unique élément de $N_{f(v)} \setminus \{f(v_1), f(v_2)\}$.

Règle 1 (b). S'il existe un sommet v avec $f(v) \in \{1,2,3\}$ et ayant un voisin v' étiqueté, alors son voisin non étiqueté doit être étiqueté par l'unique élément de $N_{f(v)} \setminus \{f(v')\}$.

Règle 1 (c). Supposons que v soit étiqueté par 2. L'étiquette de ses voisins doit appartenir à $N_2 = \{0,4\}$. Donc, si l'on connaît l'étiquette de l'un de ses voisins, l'étiquette de l'autre voisin est contrainte. Sinon, si v a une étiquette différente de 2, alors $N_{f(v)}$ contient soit 0, soit 4, ce qui force l'étiquette du voisin de degré 3.

Règle 1 (d). Supposons qu'un sommet u non étiqueté soit adjacent à un sommet v étiqueté et à u', un sommet de degré 3. Si $f(v) = 0$ alors $f(u)$ doit être 2, sinon il n'existe aucune possibilité d'étiqueter u'. De façon symétrique, si $f(v) = 4$ alors $f(u)$ est défini à 2. Si $f(v)$ appartient à $\{1,2,3\}$ alors, comme dans la seconde extension forcée, l'étiquette de u est l'unique élément de $N_{f(v)} \setminus \{f(v')\}$, où v' est le voisin étiqueté de v.

On remarque que si la règle 1 ne peut pas s'appliquer, alors chaque sommet non étiqueté qui est adjacent à un sommet étiqueté, a pour degré au plus 2. De plus ses voisins étiquetés ont pour étiquette 0 ou 4.

Définition 2.1. Un chemin P de G est un *chemin d'extension* si tous les sommets internes sont non étiquetés de degré 2, au moins une extrémité est étiquetée et l'extrémité non étiquetée (si elle existe)

a pour degré 1 ou 3. (Par abus de notation, on s'autorise à ce que les extrémités soient les mêmes et que le chemin d'extension soit un cycle dont « l'extrémité » est étiquetée. Les figures 2.3, 2.4 et 2.7 représentent ces chemins.) La *longueur*, notée length(P), de ce chemin d'extension est son nombre d'arêtes.

Lemme 2.4. *Soit $P = v_0v_1 \dots v_k$ un chemin d'extension tel que v_0 est étiqueté et v_k a pour degré 1 (et non étiqueté). Soit $G' = G[V(G) \setminus \{v_1, \dots, v_k\}]$ le sous-graphe obtenu de la suppression de P et soit $f' : V(G') \to \{0,1,2,3,4\}$ une extension de l'étiquetage de f en un étiquetage $L(2,1)$ de G'. Alors f' peut être étendu à un étiquetage $L(2,1)$ de G.*

Démonstration. Puisque $d(v_0) \leq 3$ et $f'(v_0) \in \{0,4\}$ si $d(v_0) = 3$, on peut toujours étiqueter v_1 avec une étiquette ℓ disponible. L'arête, dont une extrémité est étiquetée par $f'(v_0)$ et l'autre par ℓ, appartient à un cycle du $\overline{P_5}$ et on peut donc étiqueter le chemin P en bouclant autour de ce cycle (voir la proposition 2.1 et la figure 2.1). □

Règle 2 – extensions faciles

- Si P est un chemin d'extension avec une extrémité de degré 1, alors le lemme 2.4 garantit que les sommets non étiquetés de P sont non pertinents. Nous les supprimons du graphe et continuons avec le graphe réduit (voir la figure 2.3).

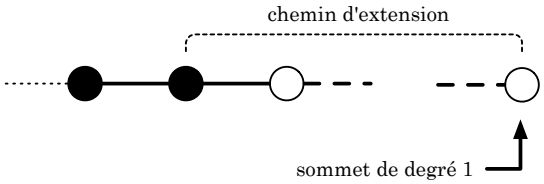


FIGURE 2.3 – Extensions faciles décrites à la règle 2. Un chemin d'extension avec une extrémité non étiquetée de degré 1.

Si aucune des règles 1 et 2 ne s'appliquent, alors chaque sommet non étiqueté qui est adjacent à un sommet étiqueté a pour degré 2.

Règle 3 - extensions économiques

(a) Si P est un chemin d'extension dont les deux extrémités sont étiquetées et de degré 2, nous pouvons décider (par *programmation dynamique*) si P admet un étiquetage $L(2,1)$ compatible avec l'étiquetage des extrémités et de leurs voisins. Si un étiquetage est possible, nous étiquetons le chemin d'extension et continuons l'étiquetage du graphe, sinon nous rejetons l'étiquetage courant f puisque celui-ci ne peut-être étendu à tout le graphe.

(b) Si P est un chemin d'extension avec des extrémités identiques, nous pouvons encore décider (par *programmation dynamique*) si P admet un étiquetage $L(2,1)$ compatible avec l'étiquette de l'extrémité et de son voisin. Selon le résultat, soit nous étiquetons P, soit nous rejetons l'étiquetage courant.

Remarque 2. *La* programmation dynamique *ne demande pas plus qu'un temps constant. En effet, en faisant une étude de cas, on observe que si le chemin est suffisamment long alors n'importe quel étiquetage de ses extrémités peut s'étendre à tout le chemin. La* programmation dynamique *n'a donc besoin de s'intéresser qu'aux chemins de longueur bornée par une constante.*

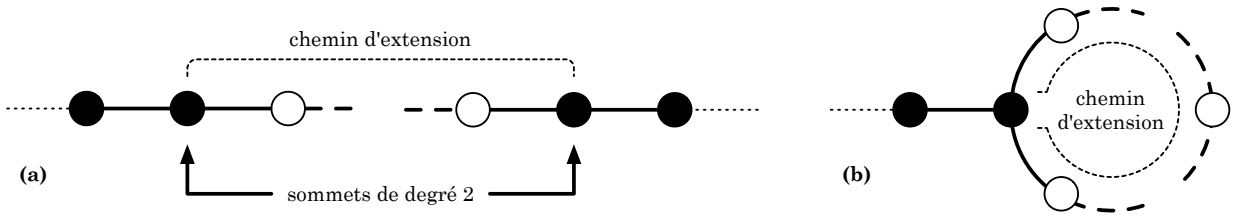


FIGURE 2.4 – Extensions économiques de la règle 3. (a) Un chemin d'extension où les deux extrémités sont étiquetées de degré 2. (b) Un chemin d'extension avec des extrémités identiques.

L'analyse du temps d'exécution de notre algorithme s'appuie essentiellement sur les règles 4 et 5 données ci-après. Les règles 1, 2 et 3 ne contribuent que pour un facteur polynomial au temps d'exécution. En effet, si l'on observe une exécution de l'algorithme, le temps d'exécution passé sur un sous-problème, avant de brancher en de nouveaux sous-problèmes, est polynomial. Pour cela, il suffit de constater que l'application des règles 1, 2 ou 3 peut être décidée en temps polynomial et que toute séquence d'application de ces règles est de longueur au plus n, puisque chaque règle réduit le nombre de sommets non étiquetés (soit en supprimant, soit en étiquetant au moins un sommet). Ainsi, le temps d'exécution de l'algorithme est borné par le nombre de sous-problèmes générés multiplié par un polynôme[1].

Règle 4 - extensions avec des contraintes fortes

- Soit P un chemin d'extension dont les deux extrémités sont étiquetées par 0 ou 4, chacune avec un seul voisin étiqueté, et au moins une extrémité a un voisin non étiqueté n'appartenant pas à P (voir la figure 2.5). Dans ce cas, nous branchons selon tous les étiquetages possibles des (au plus 4) voisins non étiquetés des extrémités de P, tout en étendant ces étiquetages à tout le chemin P (par *programmation dynamique*, comme dans la règle 3).

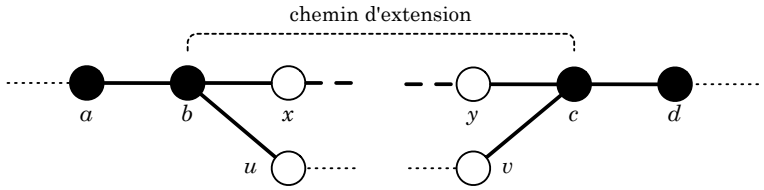


FIGURE 2.5 – Extensions avec des contraintes fortes décrites à la règle 4.

Regardons en détail ce branchement et ses conséquences sur le temps d'exécution. Comme sur la figure 2.5, nous notons b et c les extrémités étiquetées de P. Sans perte de généralité, nous supposons que b est de degré 3, nous notons a son unique voisin étiqueté, et $u \notin P$ et $x \in P$ ses voisins non étiquetés. De même pour l'extrémité c qui est de degré 2 ou 3, nous notons d son unique voisin étiqueté, et $v \notin P$ (si c est de degré 3) et $y \in P$ ses voisins non étiquetés.

Longueur 2. Si la longueur de P est égale à 2, alors $x = y$ et l'étiquette est nécessairement 2 (il n'y a donc pas besoin de branchement dans ce cas particulier).

[1] Bien que nous ne l'évaluons pas précisément, ce polynôme ne devrait pas être de grand degré ; il correspond au temps d'exécution si on exclut le coût des appels récursifs.

Longueur 3. L'étude de cas de la figure 2.6 donne tous les étiquetages possibles d'un chemin de longueur 6. De cette étude, nous obtenons les étiquetages possibles de $ab..cd$ et leurs extensions à $abxycd$ (à l'étiquetage symétrique près $f' = 4 - f$) :

$$\begin{array}{lll} 40xy40 \to 403140 & 40xy42 \to 403142 & 40xy02 \to 402402 \\ 40xy03 \to 402403 & 20xy03 \to 204203 & 20xy04 \to 204204 \\ 20xy40 \to 203140 & 20xy42 \to 203142 & 30xy02 \to 302402 \\ 30xy04 \to 304204 & 30xy03 \to 302403, 304203. & \end{array}$$

La plupart des cas ne donnent qu'une unique extension possible pour l'étiquetage de P, excepté le dernier cas où un branchement en deux étiquetages possibles apparait. Dans ce dernier cas, nous étiquetons au moins trois sommets (x, y, u et éventuellement v).

Pour analyser le temps d'exécution de notre algorithme de *branchement*, nous devons établir une borne supérieure sur $T(n)$, nombre maximum de feuilles dans l'arbre de recherche, correspondant à une exécution de l'algorithme sur un graphe ayant n sommets non étiquetés. Pour chaque sous-problème, on observe que le temps d'exécution est polynomial, si on exclut le coût des appels récursifs. De plus, à chaque appel récursif, au moins un sommet non étiqueté devient étiqueté. Ainsi le temps d'exécution total de l'algorithme sera $O^*(T(n))$.

De l'analyse précédente de la règle 4 (on rappelle que les règles 1–3 ne demandent aucun branchement et ne contribuent qu'à un facteur polynomial au temps d'exécution), nous obtenons les récurrences suivantes :

* $T(n) = 2T(n-3)$ (si c est de degré 2 ou si $v = u$) ;
* $T(n) = 2T(n-4)$ (si c a un voisin v distinct de u).

La solution d'une récurrence du type $T(n) = \alpha T(n-\beta)$ est $T(n) = \Theta(c^n)$ où $c = \sqrt[\beta]{\alpha}$.[2]. Nous obtenons donc $c = \sqrt[3]{2}$ pour la première récurrence et $c = \sqrt[4]{2}$ pour la deuxième[3].

Longueur 4. Là encore, le nombre maximum d'extensions possibles pour l'étiquetage de P de longueur 4 peut s'obtenir de l'arbre de recherche de la figure 2.6.

La table 2.1 résume le nombre d'extensions possibles et les bornes sur $T(n)$ pour des chemins d'extension de longueur 2 à 4.

longueur l du chemin P	nombre maximum de branchements t_1 si $d(c) = 2$	solution de la récurrence $T(n) = t_1 T(n-l)$	nombre maximum de branchements t_2 si $d(c) = 3$	solution de la récurrence $T(n) = t_2 T(n-l-1)$
2	1	pas de branchement	1	pas de branchement
3	2	$\mathcal{O}(2^{\frac{n}{3}}) = \mathcal{O}(1.2600^n)$	2	$\mathcal{O}(2^{\frac{n}{4}}) = \mathcal{O}(1.1893^n)$
4	2	$\mathcal{O}(2^{\frac{n}{4}}) = \mathcal{O}(1.1893^n)$	2	$\mathcal{O}(2^{\frac{n}{5}}) = \mathcal{O}(1.1487^n)$

TABLE 2.1 – Branchement sur un chemin d'extension de longueur ≤ 4 avec des contraintes fortes.

Longueur au moins 5. Si le chemin est de longueur assez grande (*i.e.* au moins 5), nous avons deux extensions possibles pour étiqueter les sommets x et u, et deux extensions possibles pour étiqueter les sommets y et v. Pour chacun de ces 4 cas, nous vérifions (en temps constant, par *programmation dynamique*, comme à la règle 3) si cet étiquetage de x, u, y, v s'étend au chemin P.

[2]Nous renvoyons le lecteur à la section 1.4.1 du chapitre 1, à (Liedloff, 2007) ou à (Fomin et Kratsch, 2010) pour des discussions plus amples sur l'analyse d'algorithmes de *branchement* via des récurrences.

[3]Nous renvoyons le lecteur à la section 1.4.1 du chapitre 1, à (Liedloff, 2007) ou à (Fomin et Kratsch, 2010) pour des discussions plus amples sur l'analyse d'algorithmes de *branchement* via des récurrences.

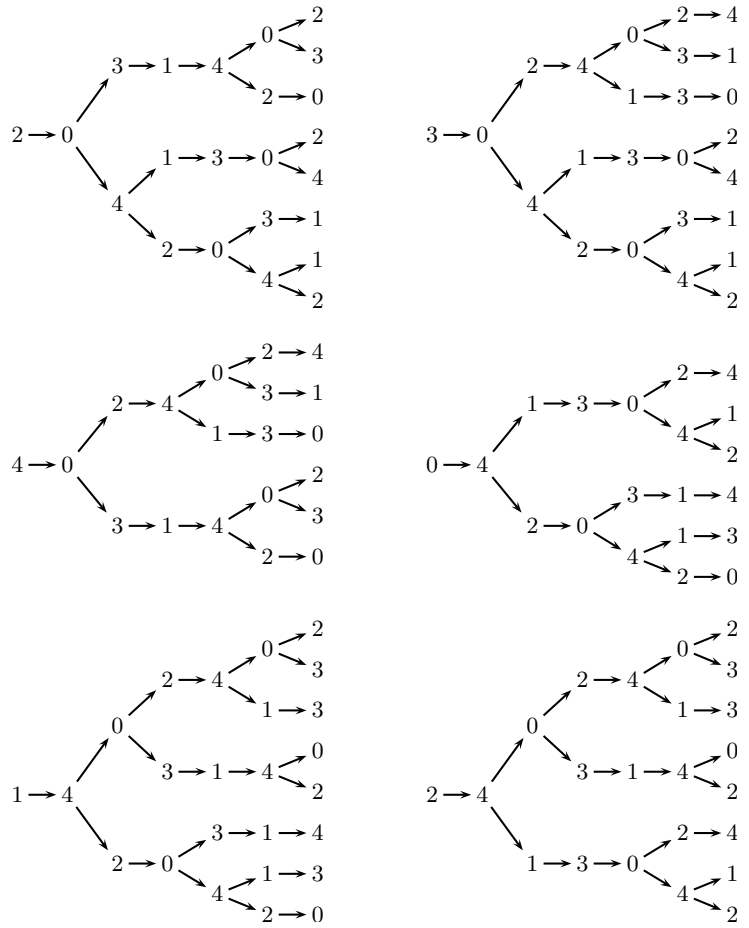


FIGURE 2.6 – Les arbres représentent tous les étiquetages $L(2,1)$ possibles d'un chemin commençant par deux sommets étiquetés a et b et terminant avec deux sommets étiquetés c et d. Rappelons que les seules étiquettes possibles pour b et c sont 0 ou 4 lorsque la règle 4 s'applique. Par exemple, si $f(a) = 2$, $f(b) = 0$, $f(c) = 0$ et $f(d) = 2$, le premier arbre indique qu'il y a deux étiquetages possible pour le chemin $abxyzcd$: 2031402 et 2041302.

* Si v existe et $v \neq u$, nous étiquetons length$(P) + 1$ nouveaux sommets (les sommets non étiquetés de P et les sommets u et v), ce qui donne les récurrences $T(n) = 4T(n-\text{length}(P)-1)$ et $T(n) = \mathcal{O}(4^{\frac{n}{6}}) = \mathcal{O}(1.2600^n)$.

* Si v n'existe pas, nous étiquetons seulement length(P) nouveaux sommets. Néanmoins, dans ce cas, nous considérons seulement deux étiquetages possibles pour x et u, et pour chacun de ces étiquetages nous vérifions s'ils s'étendent à l'étiquetage de P. L'étiquette de y n'a pas d'importance puisque dans ce cas c est de degré 2. Cela donne les récurrences $T(n) = 2T(n - \text{length}(P))$ et $T(n) = \mathcal{O}(2^{\frac{n}{5}}) = \mathcal{O}(1.1487^n)$.

* Si $v = u$, alors u est adjacent à c et donc c a pour degré 3. Dans ce cas, u serait traité par la règle 1 ; $v = u$ n'est donc pas possible à ce stade de l'algorithme.

Le pire des cas est obtenu lorsque $d(c) = 2$ et length$(P) = 3$; ce qui donne $T(n) = \mathcal{O}(1.2600^n)$.

Si aucune des règles 1 à 4 précédentes ne s'applique, alors chaque sommet non étiqueté, adjacent à un sommet étiqueté, appartient à un chemin d'extension ayant une extrémité non étiquetée de degré 3. La dernière règle de branchement traite ce cas.

longueur l du chemin P	nombre maximum de branchements t_1 si $d(b) = 2$	solution de la récurrence $T(n) = t_1T(n-l+1)$	nombre maximum de branchements t_2 si $d(b) = 3$	solution de la récurrence $T(n) = t_2T(n-l)$
1	1	pas de branchement	1	pas de branchement
2	1	pas de branchement	1	pas de branchement
3	2	$\mathcal{O}(2^{\frac{n}{3}}) = \mathcal{O}(1.2600^n)$	2	$\mathcal{O}(2^{\frac{n}{4}}) = \mathcal{O}(1.1893^n)$
4	3	$\mathcal{O}(3^{\frac{n}{4}}) = \mathcal{O}(1.3161^n)$	3	$\mathcal{O}(3^{\frac{n}{5}}) = \mathcal{O}(1.2458^n)$
5	3	$\mathcal{O}(3^{\frac{n}{5}}) = \mathcal{O}(1.2458^n)$	3	$\mathcal{O}(3^{\frac{n}{6}}) = \mathcal{O}(1.2010^n)$
6	5	$\mathcal{O}(5^{\frac{n}{6}}) = \mathcal{O}(1.3077^n)$	6	$\mathcal{O}(6^{\frac{n}{7}}) = \mathcal{O}(1.2918^n)$
7	5	$\mathcal{O}(5^{\frac{n}{7}}) = \mathcal{O}(1.2585^n)$	6	$\mathcal{O}(6^{\frac{n}{8}}) = \mathcal{O}(1.2511^n)$
8	5	$\mathcal{O}(5^{\frac{n}{8}}) = \mathcal{O}(1.2229^n)$	7	$\mathcal{O}(7^{\frac{n}{9}}) = \mathcal{O}(1.2414^n)$

TABLE 2.2 – Branchement sur un chemin d'extension de longueur ≤ 8 avec des contraintes faibles.

Règle 5 - extensions avec des contraintes faibles

- Soit P un chemin d'extension avec une extrémité v non étiquetée de degré 3. Notons w le voisin de v sur P. Notons b l'extrémité étiquetée de P et a le voisin étiqueté de b. Si b est de degré 3, nous notons u le voisin non étiqueté de b qui n'appartient pas à P (voir la figure 2.7). Nous branchons selon les étiquetages possibles pour v, w et (éventuellement) u, tout en étendant cet étiquetage à tout le chemin P (par *programmation dynamique*).

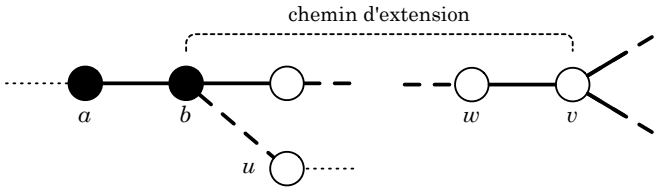


FIGURE 2.7 – Extensions avec des contraintes faibles décrites à la règle 5.

La table 2.2 résume le nombre de branchements pour des chemins de longueur au plus 8. Ces nombres sont obtenus des arbres de recherche de la figure 2.8. Lorsque $d(b) = 2$, nous comptons seulement les étiquetages de v et w qui s'étendent à un étiquetage de P compatibles avec les étiquettes de a et b, puisque l'étiquette attribuée au voisin de b sur P est sans importance. Lorsque $d(b) = 3$, nous comptons les étiquetages de v, w et u qui s'étendent à un étiquetage de P compatible avec les étiquettes de a et b. Bien sûr, les sommets v et b ne peuvent que recevoir 0 ou 4 comme étiquette.

Dans le cas d'un chemin plus long, nous avons au plus 6 étiquetages possibles pour v et w, donnant la récurrence $T(n) = 6T(n - \text{length}(P))$ si $d(b) = 2$. Si $d(b) = 3$, nous avons au plus 12 étiquetages possibles pour v, w et u, donnant alors la récurrence $T(n) = 12T(n - \text{length}(P) - 1)$.

Puisque $\sqrt[9]{6} < \sqrt[4]{3}$ et $\sqrt[10]{12} < \sqrt[4]{3}$, le pire des cas pour la règle 5 est obtenu lorsque $d(b) = 2$, $\text{length}(P) = 5$ et donc $T(n) = \mathcal{O}(3^{\frac{n}{4}}) = \mathcal{O}(1.3161^n)$.

En conséquence de cette analyse, nous obtenons le théorème :

Théorème 2.5 (Havet *et al.* (2011), théorème 5). *L'existence d'un étiquetage $L(2,1)$ de largeur 4 peut être décidée en temps $O^*(1.3161^n)$ et espace polynomial. Si un tel étiquetage existe, il peut être construit dans ce même temps.*

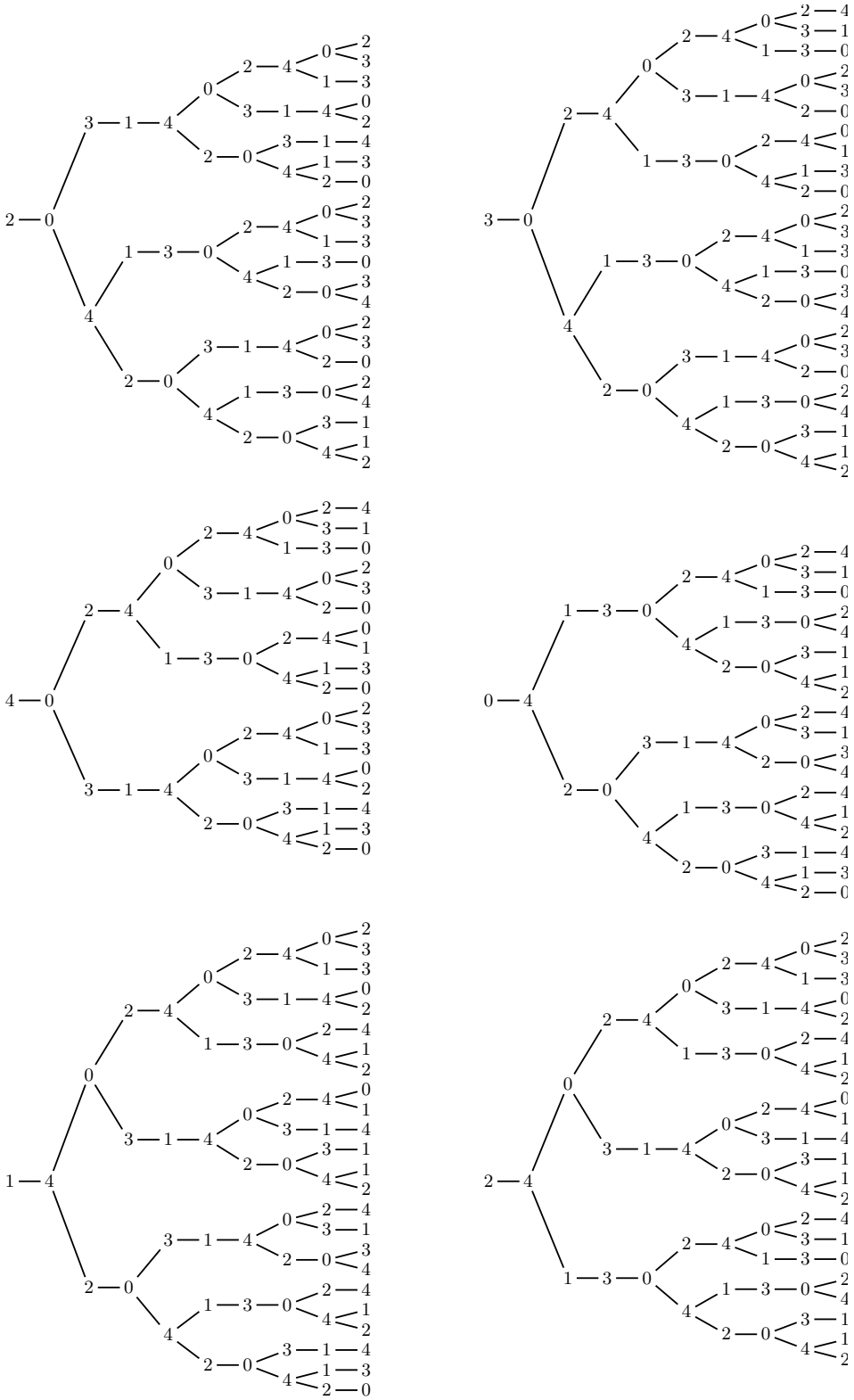


FIGURE 2.8 – Les arbres représentent tous les étiquetages $L(2,1)$ possibles de chemins d'extension de longueur 8. L'arête à la racine de l'arbre correspond aux deux premiers sommets étiquetés.

2.2.1.2 Une analyse raffinée avec mesurer-pour-conquérir

Il est possible d'améliorer (modestement) l'analyse de complexité précédente en utilisant la technique *mesurer-pour-conquérir*. Le lecteur peut se référer aux travaux fondateurs de Fomin *et al.* (2009c) (où la technique est illustrée pour la résolution du stable maximum et du dominant minimum) ainsi qu'à l'ouvrage de Fomin et Kratsch (2010) pour des explications plus pointues sur l'usage de cette technique. Des exemples mettant en oeuvre *mesurer-pour-conquérir* (appliqués de façon plus sophistiquée à des problèmes de domination) abondent également dans mon travail de thèse (Liedloff, 2007). L'idée générale est d'utiliser une mesure *non standard* sur la taille de l'instance, afin de mieux mesurer les progrès faits par l'algorithme à chaque étape de branchement. La mesure doit donc être judicieusement choisie afin de tirer partie de la technique. On obtient des récurrences typiquement plus difficiles à établir et à résoudre. Lorsque celles-ci sont nombreuses, il peut être nécessaire d'implémenter un programme pour générer l'ensemble des récurrences. Eppstein (2006) donne une méthode pour résoudre ces récurrences qui dépendent de plusieurs paramètres. Essentiellement, il montre qu'il suffit de se ramener à des récurrences ne dépendant que d'une seule variable, par une combinaison pondérée des différents paramètres. On obtient ainsi le comportement asymptotique des récurrences à plusieurs paramètres.

Dans la suite de cette section, nous utilisons *mesurer-pour-conquérir* pour améliorer la borne de $O^*(1.3161^n)$ établie au théorème 2.5, sans changer l'algorithme décrit précédemment. Nous établissons le théorème suivant :

Théorème 2.6 (Havet *et al.* (2011), théorème 6)**.** *L'algorithme de la section 2.2.1.1 a un temps d'exécution de* $\mathcal{O}(1.3006^n)$.

Démonstration. Étant donné un graphe G et un étiquetage partiel f de ses sommets, considérons la mesure :

$$\mu = \mu(G, f) = \tilde{n} + \epsilon\hat{n}$$

où $\tilde{n}$ est le nombre de sommets non étiquetés sans voisin étiqueté et $\hat{n}$ est le nombre de sommets non étiquetés avec un voisin étiqueté. Le choix de la constante ϵ sera précisé plus tard, avec $0 \leq \epsilon \leq 1$. Ainsi, $\mu(G, f) \leq n$ où n est le nombre de sommets de G. Ce choix de mesure signifie qu'un sommet a un poids égal à 0 s'il est déjà étiqueté, égal à ϵ s'il est non étiqueté avec un voisin étiqueté, et sinon égal à 1.

L'analyse que nous devons mener est similaire à celle de la section 2.2.1.1, mais nous obtenons des récurrences différentes, qu'il faut expliciter. Nous reprenons chacune des règles, en observant tout d'abord que les règles 1–3 (qui ne demandent aucun branchement) n'ont pas d'influence notable pour cette nouvelle analyse du temps d'exécution.

Règles 1, 2 et 3. Pour tout sous-problème, toute séquence d'application des règles de réductions 1, 2 ou 3, a un temps d'exécution au plus polynomial. En effet, d'une part il est possible de décider en temps polynomial si ces règles peuvent être appliquées au sous-problème ; et d'autre part, chaque séquence d'application a pour longueur au plus n puisque ces règles étiquettent ou suppriment au moins un sommet à chaque application. Finalement, nous pouvons noter que leur application n'augmente pas la mesure μ du sous-problème.

Pour simplifier les notations, nous notons $w(v)$ le poids d'un sommet v. Il peut prendre trois valeurs distinctes :

* $w(v) = 1$ si v est non étiqueté, sans voisin étiqueté ;
* $w(v) = \epsilon$ si v est non étiqueté, avec un voisin étiqueté ;
* $w(v) = 0$ si v est étiqueté.

Par conséquent, on a aussi la relation $\mu(G, f) = \sum_{v \in V} w(v)$. Observons de plus, que le poids d'un sommet ne peut jamais augmenter lors d'un appel récursif à l'algorithme.

Règle 4 - extensions avec des contraintes fortes Reconsidérons un chemin d'extension P ayant les deux extrémités étiquetées, comme celui de la figure 2.9. Nous branchons sur les étiquetages possibles des voisins non étiquetés des extrémités de P (comme indiqué à la section précédente).

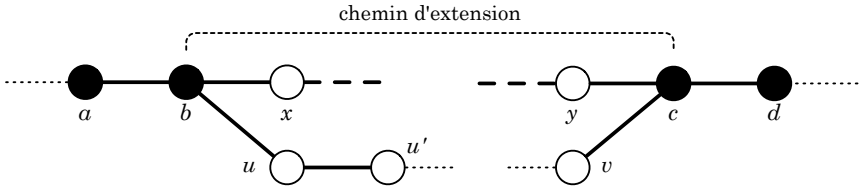


FIGURE 2.9 – Extensions avec des contraintes fortes de la règle 4.

Par application des règles 1 et 2, les degrés de u et v (si ce sommet existe) sont égaux à 2. Notons u' le voisin non étiqueté de u. Le sommet u' est effectivement non étiqueté puisque, dans le cas contraire, la règle 1 (d) aurait été appliquée pour étiqueter u. Le poids $w(u')$ peut soit être égal à 1, soit être égal à ϵ. Regardons ces deux cas :

* Si $w(u') = 1$, alors l'étiquetage de u ferait diminuer le poids de u' à ϵ.
* Si $w(u') = \epsilon$, alors notons u'' un voisin étiqueté de u'. À cause de la règle 1, nous en déduisons que u' doit être de degré 2. Donc l'étiquetage de u créerait un chemin d'extension $P' = uu'u''$ de longueur 2 (u' est le sommet non étiqueté de P') qui peut être étiqueté sans aucun branchement par la règle 4. Ainsi, l'étiquetage de u ferait diminuer le poids de u' à 0.

Par conséquent, si v n'existe pas, l'étiquetage du chemin P et du sommet u ferait diminuer la mesure d'au moins $(2\epsilon + (\text{length}(P) - 3) + \epsilon + \min(1 - \epsilon, \epsilon))$, ce qui donnerait la récurrence

$$T(\mu) \leq t_1 \cdot T(\mu - (2\epsilon + (\text{length}(P) - 3) + \epsilon + \min(1 - \epsilon, \epsilon)))$$

où le nombre t_1 est donné à la table 2.1 pour $\text{length}(P) \leq 4$ et sinon $t_1 = 2$ (se référer à l'analyse de la règle 4 à la section précédente). Si v existe, nous pouvons supposer que les sommets u et v sont différents car dans le cas contraire, la règle 1 (d) aurait étiqueté u sans branchement. Néanmoins, il est tout de même possible que $u' = v$. Ainsi, l'étiquetage du chemin P et des sommets u et v diminuerait la mesure d'au moins $(2\epsilon + (\text{length}(P) - 3) + 2\epsilon)$, ce qui donnerait la récurrence

$$T(\mu) \leq t_2 \cdot T(\mu - (2\epsilon + (\text{length}(P) - 3) + 2\epsilon))$$

où t_2 est le nombre maximum de branchements donné par la table 2.1 pour $\text{length}(P) \leq 4$, et $t_2 = 4$ pour tout $\text{length}(P) \geq 5$ (là encore, on se réfère à l'analyse précédente de la règle 4).

Règle 5 - extensions avec des contraintes faibles Dans cette règle, nous considérons un chemin d'extension P dont l'une des extrémités est non étiquetée de degré 3. Nous branchons sur tous les étiquetages possibles de v, w et (éventuellement) u (voir la section précédente ainsi que la figure 2.10).

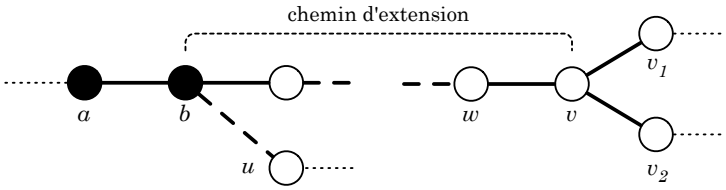


FIGURE 2.10 – Extensions avec des contraintes faibles de la règle 5.

Notons v_1 et v_2 les deux voisins de v qui n'appartiennent pas au chemin P. À cause de la règle 1 (d), ni v_1, ni v_2, ne sont étiquetés ou adjacents à un sommet étiqueté.

Si le sommet u existe, l'étiquetage du chemin P ferait diminuer la mesure d'au moins $(\epsilon + (\text{length}(P) - 1) + 2 - 2\epsilon)$. La récurrence que nous obtenons est donnée par

$$T(\mu) \leq t_1 \cdot T(\mu - (\epsilon + (\text{length}(P) - 1) + 2 - 2\epsilon))$$

où le nombre maximum de branchements, noté t_1, est donné à la table 2.2 pour $\text{length}(P) \leq 8$, et $t_1 = 6$ lorsque $\text{length}(P) \geq 9$ (se référer à l'analyse détaillée de la règle 5 à la section précédente). Si le sommet u existe, nous pouvons supposer que u et v_i ($i \in \{1,2\}$) sont effectivement différents, puisque dans le cas contraire, la règle 1 aurait étiqueté u sans branchement. Ainsi, l'étiquetage du chemin P et du sommet u ferait diminuer la mesure d'au moins $(\epsilon + (\text{length}(P) - 1) + 2 - 2\epsilon + \epsilon)$, en donnant comme récurrence

$$T(\mu) \leq t_2 \cdot T(\mu - (\epsilon + (\text{length}(P) - 1) + 2 - 2\epsilon + \epsilon)),$$

où t_2 est donné à la table 2.2 lorsque $\text{length}(P) \leq 8$, et $t_2 = 12$ dans les autres cas (voir les détails à la section précédente).

Il nous reste finalement à préciser la valeur du paramètre ϵ introduit dans la mesure μ. Cette valeur est choisie de sorte à minimiser la plus grande solution de toutes ces récurrences, pour n'importe quelle longueur de chemin P. En prenant $\epsilon = 0.8190$ et en résolvant les récurrences, nous pouvons établir que le temps d'exécution de notre algorithme est borné par $\mathcal{O}(1.3006^n)$. Cela conclut la preuve du théorème 2.6. □

Bien entendu, le choix d'une autre mesure plus pertinente est une question intéressante, qui pourrait améliorer (peut-être de façon plus significative) notre borne sur le temps d'exécution au pire des cas de l'algorithme.

2.2.1.3 Une borne inférieure sur le temps d'exécution

A la section 2.2.1.1 nous avons présenté un algorithme qui calcule un étiquetage $L(2,1)$ de largeur 4 d'un graphe, si un tel étiquetage existe, en temps $\mathcal{O}(1.3161^n)$. En raffinant notre analyse, par l'introduction d'une autre mesure, nous avons amélioré cette borne à $\mathcal{O}(1.3006^n)$ dans la section 2.2.1.2. Cette borne reste certainement surestimée, comme le sont souvent les bornes des algorithmes de *branchement*, même avec l'utilisation de la technique *mesurer-pour-conquérir*. Il est

donc fort naturel de se demander si la borne $\mathcal{O}(1.3006^n)$ est très éloignée de la meilleure borne supérieure que nous puissions espérer obtenir. Une borne inférieure sur le temps d'exécution au pire des cas, nous donnera une idée sur ce point. Dans la suite de cette section, nous considérons le graphe de la figure 2.11 pour établir $\Omega(1.2290^n)$ comme borne inférieure.

Théorème 2.7 (Havet *et al.* (2011), théorème 7). *Le temps d'exécution au pire des cas de notre algorithme de* branchement *pour calculer un étiquetage $L(2,1)$ de largeur 4 est* $\Omega\left(\left(2+\sqrt{5}\right)^{\frac{n}{7}}\right) = \Omega(1.2290^n)$.

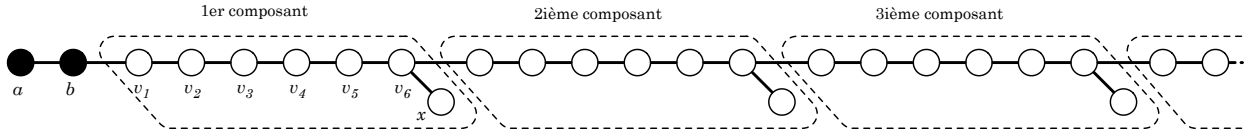


FIGURE 2.11 – Le graphe G_l utilisé pour prouver la borne inférieure.

Démonstration. Nous considérons le graphe $G_l = (V_l, E_l)$, à $7l + 2$ sommets, que nous construisons de la façon suivante (ce graphe est partiellement représenté à figure 2.11).

- l'ensemble des sommets est défini par $V_l = \{a,b\} \cup \bigcup_{\substack{1\le i\le l\\ 1\le j\le 6}} \{v_j^i\} \cup \bigcup_{1\le i\le l}\{x^i\}$;
- l'ensemble des arêtes E_l est défini par $\{a,b\} \cup \{b,v_1^1\} \cup \bigcup_{1\le i\le l} E^i \cup \bigcup_{1\le i<l}\{v_6^i, v_1^{i+1}\}$, où E^i est défini pour tout i, $1 \le i \le l$, par $E^i = \{\{v_1^i, v_2^i\}, \{v_2^i, v_3^i\}, \{v_3^i, v_4^i\}, \{v_4^i, v_5^i\}, \{v_5^i, v_6^i\}, \{v_6^i, x^i\}\}$.

Etant donné un entier i, le sous-graphe induit par $\bigcup_{1\le j\le 6}\{v_j^i\} \cup \{x^i\}$ est appelé le *i ième composant* of G_l. Pour $i > 1$, nous notons P^i le chemin induit par les sommets $\{v_6^{i-1}\} \cup \bigcup_{1\le j\le 6}\{v_j^i\}$.

Étudions une exécution de l'algorithme sur ce graphe G_l. Supposons qu'à l'étape préliminaire de l'algorithme, celui-ci choisisse les deux sommets voisins a et b pour les étiqueter, en leur attribuant respectivement l'étiquette 0 et l'étiquette 4. Nous pouvons facilement observer qu'aucune des règles 1 à 3 ne s'applique. De plus, cet étiquetage partiel ne produit pas de chemin d'extension dont les deux extrémités sont étiquetées ; ce qui implique que la règle 4 ne s'applique pas. L'algorithme utilise donc la règle 5 au chemin d'extension $P^1 = \{b, v_1^1, v_2^1, \ldots, v_6^1\}$ pour continuer l'étiquetage du graphe.

L'idée est d'essayer tous les étiquetages possibles de P^1. Précisément, l'algorithme branche selon les étiquetages possibles de v_5^1 et v_6^1, puis étend ces étiquetages aux autres sommets non étiquetés de P^1. Ensuite, par application de la règle 2, le sommet x est supprimé du graphe (le lemme 2.4 garantit que son étiquette peut facilement être obtenue de l'étiquetage du reste du graphe). Le nombre maximum de branchements pour ce chemin d'extension P^1 de longueur 6 est donné à la table 2.2, *i.e.* 5. Néanmoins, selon l'étiquette de a et b, ce nombre peut être plus petit. La table 2.3 donne le nombre exact de branchement, selon les étiquettes des sommets a et b.

Notons qu'une fois que P_1 est étiqueté, le sommet x est supprimé (pour être étiqueté plus tard, sans branchement), les sommets v_5^1 et v_6^1 sont étiquetés, et v_6^1 a nécessairement pour étiquette 0 ou 4 (puisqu'il a pour degré 3). Ainsi, par induction, nous pouvons réutiliser les mêmes arguments que précédemment en renommant v_5^1 par a et v_6^1 par b, puis en considérant le chemin d'extension P^2 du deuxième composant. Finalement, le graphe G_l est entièrement étiqueté, composant par composant, essentiellement par la règle 5.

valeurs pour $(f(a), f(b))$	$(2,0)$	$(3,0)$	$(4,0)$	$(0,4)$	$(1,4)$	$(2,4)$
valeurs possibles pour $(f(v_5^1), f(v_6^1))$	$(2,4)$ $(0,4)$ $(1,4)$ $(2,0)$	$(4,0)$ $(1,4)$ $(0,4)$ $(2,4)$ $(2,0)$	$(4,0)$ $(1,4)$ $(0,4)$ $(2,4)$	$(4,0)$ $(2,0)$ $(3,0)$ $(0,4)$	$(2,4)$ $(3,0)$ $(2,0)$ $(4,0)$ $(0,4)$	$(2,4)$ $(3,0)$ $(2,0)$ $(4,0)$
nombre de branchements	4	5	4	4	5	4

TABLE 2.3 – Nombre de branchements pour un chemin d'extension de longueur 6 en utilisant la règle 5, selon les étiquettes de a et b. Ces nombres sont obtenus en se référant aux arbres de la figure 2.8.

D'après la table 2.3, le nombre de branchements est soit 4, soit 5, selon les étiquettes de a et b. Pour analyser le temps d'exécution de l'algorithme sur G_l, nous notons $T_4(n)$ le nombre maximum de feuilles dans l'arbre de recherche obtenu de l'exécution de l'algorithme sur le graphe G_l avec n sommets non étiquetés, en supposant que les deux premiers sommets (c'est-à-dire a et b lors de la première étape, puis v_5^i et v_6^i, $1 \leq i < l$, à la $(i+1)^{\text{ème}}$ étape) sont étiquetés $(2,0)$, $(4,0)$, $(0,4)$ ou $(2,4)$. De façon similaire, nous définissons $T_5(n)$ lorsque les deux premiers sommets sont étiquetés $(3,0)$ ou $(1,4)$. Clairement, nous avons la relation $T_5(n) \geq T_4(n)$ pour tout n positif.

Rappelons que chaque fois qu'un chemin d'extension P^iest étiqueté, le chemin devient étiqueté et le sommet x^i est supprimé. Donc les 7 sommets sont soit étiquetés, soit supprimés de G_l. De plus, parmi les 4 ou 5 branchements effectués pour déterminer l'étiquetage de P^i, seulement l'un d'eux ($(3,0)$ ou $(1,4)$) produit un branchement en 5 étiquetages possibles à l'étape suivante pour étiqueter P^{i+1} (voir la table 2.3). On obtient ainsi ces deux récurrences :

$$T_4(n) = 3T_4(n-7) + T_5(n-7) \tag{2.1}$$
$$T_5(n) = 4T_4(n-7) + T_5(n-7). \tag{2.2}$$

En soustrayant (2.1) de (2.2) nous obtenons

$$T_5(n) = T_4(n) + T_4(n-7).$$

Puis, en réinjectant cette expression dans (2.1) nous obtenons

$$T_4(n) = 4T_4(n-7) + T_4(n-14).$$

En résolvant cette dernière récurrence, en substituant $T_4(n) = \alpha_4^n$, nous obtenons

$$T_4(n) = \left(2+\sqrt{5}\,\right)^{\frac{n}{7}} > 1.2290^n$$

(et donc $T_5(n) = \alpha_4^n \cdot (\alpha_4^{-7}+1) \geq T_4(n)$), ce qui démontre le théorème. □

2.2.1.4 Dénombrer les étiquetages $L(2,1)$ de largeur 4

On peut s'intéresser à un problème plus général que décider si un graphe admet un étiquetage $L(2,1)$ de largeur 4 : dénombrer ces étiquetages. Dans (Couturier *et al.*, 2013a), nous montrons le théorème suivant :

Théorème 2.8 (Couturier *et al.* (2013a), théorème 7). *Pour $\epsilon > 0$ quelconque, tous les étiquetages $L(2,1)$ de largeur 4 peuvent être dénombrés en temps $\mathcal{O}^*(6^{(1/15+\epsilon)n})$ (et espace exponentiel).*

Bien sûr, cet algorithme permet de résoudre le problème ÉTIQUETAGE $L(2,1)$ de largeur 4 en temps $\mathcal{O}^*(6^{(1/15+\epsilon)n}) = \mathcal{O}(1.1269^n)$ (en choisissant ϵ suffisamment petit), mais au coût d'un espace exponentiel. Dans la suite nous donnons quelques éléments de la preuve du théorème 2.8.

Éléments de la preuve du théorème 2.8. On observe que si $G = (V, E)$ est un graphe admettant un étiquetage $L(2,1)$ de largeur 4, alors $\Delta(G) \leq 3$ et en particulier le nombre de sommets de degrés égaux à 3 est au plus $\frac{2n}{5}$. Un graphe H auxiliaire est construit à partir de ces sommets de degré 3, dont on calcule une décomposition linéaire (en anglais, *path decomposition*). Par un résultat de Fomin *et al.* (2009a), on obtient que la largeur linéaire de H, notée $pw(H)$, satisfait $pw(H) \leq (\frac{1}{6}+\epsilon)\frac{2n}{5}$, pour $\epsilon > 0$ et H suffisamment grand (la taille requise dépendant de ϵ). Une *programmation dynamique* est ensuite effectuée sur la décomposition linéaire correspondante. Cela nécessite un encodage des étiquetages $L(2,1)$ et le calcul de tables correspondant à chacun des sacs de la décomposition. La taille de ces tables ainsi que leurs calculs s'effectuent en temps et espace bornés par $\mathcal{O}^*(6^{(\frac{1}{6}+\epsilon)\frac{2n}{5}})$. Clairement, si $\Delta(G) < 3$, alors les étiquetages peuvent être dénombrés en temps polynomial et si $\Delta(G) > 3$ alors G n'admet pas d'étiquetage $L(2,1)$ de largeur 4. □

Remarque 3. *L'algorithme en temps $\mathcal{O}(1.3006^n)$ de la section 2.2.1 peut certainement être modifié pour également dénombrer les étiquetages de largeur 4, en espace polynomial. Pour cela un petit travail est nécessaire pour l'étiquetage des chemins d'extension. Lorsque ce chemin est de petite longueur, nous avons analysé les différents cas et établi les étiquetages possibles. Lorsque le chemin est de longueur plus grande, il nous faut une formule qui, étant donnés sa longueur et l'étiquetage de ses extrémités, donne le nombre d'étiquetages possibles de ce chemin. Nous n'avons pas établi cette formule, mais il ne devrait pas être très difficile de l'obtenir.*

2.2.2 Énumération des étiquetages de largeur 5 pour les graphes cubiques

Golovach *et al.* (2010) donnent le théorème suivant :

Théorème 2.9 (Golovach *et al.* (2010), théorème 5)**.** *Tous les étiquetages $L(2,1)$ de largeur 5 d'un graphe connexe cubique à n sommets peuvent être énumérés en temps $\mathcal{O}(1.8613^n)$, et le nombre d'étiquetages $L(2,1)$ de largeur 5 de n'importe quel graphe connexe cubique est $\mathcal{O}(1.8613^n)$.*

Ce théorème est complété par une borne inférieure de $\Omega(2^{n/6}) = \Omega(1.1224)$. Dans (Couturier *et al.*, 2013a), nous améliorons la borne supérieure à $\mathcal{O}(1.7990^n)$ et la borne inférieure à $\Omega(2^{2n/7}) = \Omega(1.2191^n)$. Nous établissons la borne supérieure par un algorithme de *branchement* et la borne inférieure par une construction d'un graphe particulier. Puisque notre algorithme est capable d'énumérer tous les étiquetages $L(2,1)$ de largeur 5, il sera bien évidemment capable de les dénombrer en préservant ce même temps d'exécution. La figure 2.12 donne un exemple de graphe cubique et un étiquetage $L(2,1)$ de largeur 6. On rappelle que déterminer un étiquetage $L(2,1)$ de largeur 6 est équivalent à la recherche d'un homomorphisme localement injectif vers le graphe $\overline{P_6}$ (voir la proposition 2.1).

2.2.2.1 Description de l'algorithme

Nous présentons un algorithme de *branchement* qui énumère tous les étiquetages $L(2,1)$ de largeur 5 d'un graphe connexe cubique. Notons que la recherche des étiquetages $L(2,1)$ de largeur 4 dans un tel graphe ne laisse pas beaucoup d'espoir puisqu'il n'y en a aucun ! En effet, un tel

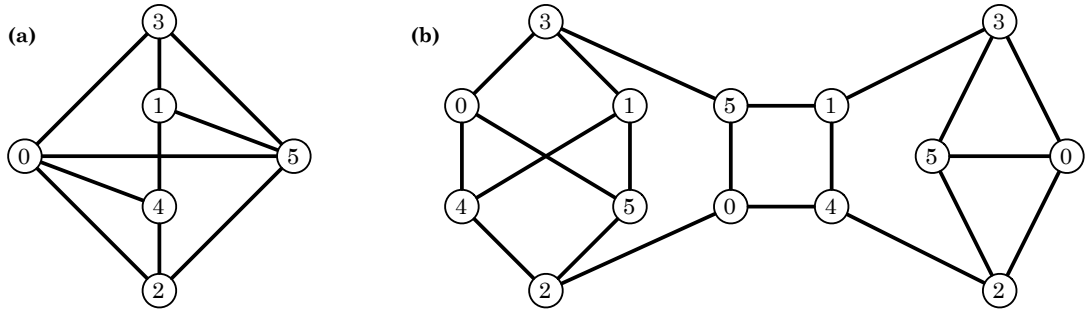


FIGURE 2.12 – (a) Le graphe $\overline{P_6}$. (b) Un graphe cubique G avec un étiquetage $L(2,1)$ de largeur 5 qui correspond à un homomorphisme localement injectif vers le $\overline{P_6}$.

graphe contient au moins un sommet de degré 3 dont les voisins sont également de degré 3 et donc impossible à étiqueter. Il est donc naturel de s'intéresser à un étiquetage de largeur 5.

L'algorithme que nous présentons exploite les propriétés asymétriques de l'étiquetage : étiqueter un sommet x par 0 ou 5 laisse quatre couleurs possibles pour ses voisins, alors qu'étiqueter x par 1, 2, 3 ou 4 laisse seulement trois couleurs possibles pour son voisinage. Nous utilisons cette propriété dans nos règles de branchement. A chaque étape, nous étiquetons un ou plusieurs sommets à la fois. Pour chacun des étiquetages possibles de ces sommets, l'algorithme branche en autant de sous-problèmes. S'il n'existe aucun étiquetage possible pour ces sommets, alors le sous-problème courant est abandonné. A chaque fois que nous étiquetons des sommets, nous conservons la propriété que l'ensemble des sommets étiquetés du graphe forme un sous-graphe connexe. Dans la suite nous décrivons les règles de branchement qui indiquent la façon dont sont choisis les sommets à étiqueter. L'analyse du temps d'exécution fait appel à la résolution de récurrences et nous employons la technique *mesurer-pour-conquérir*, comme à la section 2.2.1.2. Etant donné un graphe G, partiellement étiqueté, nous lui attribuons la mesure

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

où :

- n est le nombre de sommets non étiquetés de G ;
- $|\tilde{M}|$ est la taille maximale d'un couplage induit[4] $\tilde{M}$ dans le graphe biparti $\tilde{G}$, où $\tilde{G}$ est le graphe induit par la bi-partition (X, Y), avec X l'ensemble des sommets d'étiquettes dans $\{1,2,3,4\}$ et Y l'ensemble des sommets non étiquetés de $N(X)$;
- $\epsilon \in [0;1]$ est une constante que nous déterminerons plus tard.

L'utilisation dans la mesure du couplage induit $\tilde{M}$ peut paraître à première vue étrange. En fait, il capture si de possibles applications de l'avantageuse règle [B1] peuvent être effectuées juste après l'emploi des règles [B2], [B3] ou [B4] (ces règles seront définies dans suite). Observons que le choix de la mesure garantit la relation $\mu(G) \leq n$. Ainsi, lorsque nous établirons un temps d'exécution $\mathcal{O}(\alpha^{\mu})$, dépendant de la taille $\mu(G)$ du graphe, nous pourrons immédiatement l'exprimer en fonction de n (puisque $\alpha^{\mu} \leq \alpha^{n}$).

Pour l'analyse du temps d'exécution, nous aurons besoin des deux lemmes suivants. Ils établissent les influences possibles de la taille du couplage $\tilde{M}$ lors de l'extension de l'étiquetage.

[4] Bien que le calcul d'un couplage induit maximum soit NP-difficile, comme nous le verrons dans la suite, cela ne pose pas de problème particulier pour l'analyse de l'algorithme.

Lemme 2.10. *Soit G un graphe partiellement étiqueté. Supposons que $\tilde{M}$ soit un couplage induit maximum du graphe $\tilde{G}$ (on rappelle que $\tilde{G}$ est le graphe induit par la bi-partition (X,Y), où X est l'ensemble des sommets dont les étiquettes sont dans $\{1,2,3,4\}$ et Y est l'ensemble des sommets non étiquetés de $N(X)$). Soit S un sous-ensemble de sommets non étiquetés. Supposons que l'étiquetage de G soit étendu aux sommets de S. Notons $\tilde{M}'$ le couplage induit maximum résultant de cet étiquetage. Alors la taille de $\tilde{M}'$ est au moins $|\tilde{M}| - |S|$.*

Démonstration. Soit $E(S)$ les arêtes de $\tilde{M}$ ayant une extrémité dans S. On observe que $(\tilde{M} \setminus E(S))$ reste un couplage induit, de taille au moins $|\tilde{M}| - |S|$. □

Lemme 2.11. *Soit G un graphe partiellement étiqueté. Supposons que u soit un sommet dont l'étiquette est dans $\{1,2,3,4\}$, ayant deux voisins x et y non étiquetés. Supposons que $\tilde{M}$ soit un couplage induit maximum du graphe $\tilde{G}$. Supposons que l'étiquetage de G soit étendu aux sommets x et y. Notons $\tilde{M}'$ le couplage induit maximum résultant de cet étiquetage. Si $|\tilde{M}'| \leq |\tilde{M}| - 2$, alors x et y ont chacun un voisin distinct étiqueté, différent de u, avec une étiquette dans $\{1,2,3,4\}$.*

Démonstration. Considérons les trois cas suivants :

1. Supposons que ni x, ni y aient un voisin, différent de u, avec une étiquette dans $\{1,2,3,4\}$. Comme $\tilde{M}$ forme un couplage induit, alors on a l'inégalité $|\{\{x,u\},\{y,u\}\} \cap \tilde{M}| \leq 1$. Donc $(\tilde{M} \setminus \{\{x,u\},\{y,u\}\})$ est un couplage induit de taille au moins $|\tilde{M}| - 1$ dans l'étiquetage étendu où x et y sont étiquetés.

2. Supposons que x et y aient un même voisin commun v, différent de u, avec une étiquette dans $\{1,2,3,4\}$. Puisque $\tilde{M}$ est un couplage induit, $|\{\{x,u\},\{y,u\},\{x,v\},\{y,v\}\} \cap \tilde{M}| \leq 1$ Donc $(\tilde{M} \setminus \{\{x,u\},\{y,u\},\{x,v\},\{y,v\}\})$ est un couplage induit de taille au moins $|\tilde{M}| - 1$ dans l'étiquetage étendu où x et y sont étiquetés.

3. Supposons que seulement l'un des sommets x et y ait un voisin v, différent de u, avec une étiquette dans $\{1,2,3,4\}$. Sans perte de généralité, nous supposons que v est le voisin de x. On a encore $|\{\{x,u\},\{y,u\},\{x,v\}\} \cap \tilde{M}| \leq 1$. Donc $(\tilde{M} \setminus \{\{x,u\},\{y,u\},\{x,v\}\})$ est un couplage induit de taille au moins $|\tilde{M}| - 1$ dans l'étiquetage étendu où x et y sont étiquetés.

Nous avons montré que chacun des trois cas implique un couplage induit de taille au moins $|\tilde{M}| - 1$. Donc, si on a $|\tilde{M}'| \leq |\tilde{M}| - 2$, alors x et y doivent avoir chacun un voisin étiqueté distinct, différent du sommet u, avec des étiquettes dans $\{1,2,3,4\}$. □

Dans la suite de cette section, nous présentons les règles de branchement de notre algorithme qui énumère tous les étiquetages de largeur 5 dans un graphe cubique. Pour chaque règle de branchement, nous donnerons les récurrences correspondantes et qui établiront le temps d'exécution de cet algorithme récursif. Les règles sont appliquées dans l'ordre donné, c'est-à-dire qu'une règle n'est appliquée que si les règles décrites précédemment ne peuvent pas s'appliquer. Comme à la section 2.2.1.1, nous commençons par étiqueter deux sommets arbitraires u et v voisins, par les 20 étiquetages possibles et nous branchons sur les 20 sous-problèmes ainsi obtenus.

Règle [B0] – trois voisins étiquetés

- S'il existe un sommet non étiqueté x, voisin de trois sommets u_1, u_2, u_3 étiquetés (voir la figure 2.13), si possible nous étiquetons x (ou nous arrêtons la recherche d'étiquetage pour ce sous-problème).

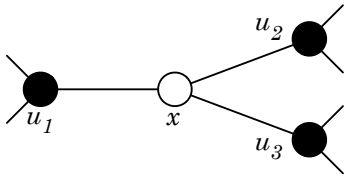


FIGURE 2.13 – Illustration de la règle [B0].

L'étiquette de x est soit déterminée de façon unique, soit il existe deux étiquettes possibles. Supposons d'abord que l'étiquette de x soit forcée. Dans ce cas, nous n'avons pas besoin de brancher. Puisqu'un sommet est étiqueté et que la taille du couplage induit maximum de $\tilde{G}$ ne peut que décroitre de 1 par le lemme 2.10, nous obtenons

$$T(\mu) \leq T(\mu - 1 + \epsilon).$$

Regardons maintenant le cas où nous devons brancher sur x selon ses deux étiquettes possibles. Dans ce cas, l'étiquette d'un autre voisin non étiqueté de u_1, u_2 ou u_3 devient déterminée de façon unique (rappelons que les sommets étiquetés forment une composante connexe et donc u_1, u_2, u_3 ont chacun déjà un voisin étiqueté). Ainsi, l'un des sommets u_i ($i \in \{1,2,3\}$) a une étiquette dans $\{1,2,3,4\}$ et un voisin x_i' non étiqueté, différent de x, car sinon l'étiquette de x est déterminée de façon unique. Si, par l'étiquetage de x et x_i', la taille du couplage induit $\tilde{M}$ décroît d'au plus 1, nous étiquetons x et x_i'. Sinon, la taille de $\tilde{M}$ décroît de 2 et, dans ce cas, u_i ne peut pas être l'extrémité d'une arête de $\tilde{M}$. Il existe donc un indice $j \in \{1,2,3\}$, $j \neq i$, tel que $\{x, u_j\} \in \tilde{M}$. Comme $\{x, u_j\} \in \tilde{M}$, l'étiquette de u_j appartient à $\{1,2,3,4\}$. Puisque l'étiquette de x n'est pas forcée, le sommet u_j a un voisin non étiqueté x_j' différent de x. Ce sommet x_j' ne peut pas être une extrémité d'une arête de $\tilde{M}$, puisque $\tilde{M}$ est un couplage induit contenant déjà $\{x, u_j\}$. Dans ce cas, nous étiquetons les sommets x et x_j'. Cela décroît la taille du couplage induit d'au plus 1. Ainsi, la récurrence correspondante à ce branchement est donnée par

$$T(\mu) \leq 2 \cdot T(\mu - 2 + \epsilon).$$

Règle [B1] – un ou deux voisins étiquetés $1, 2, 3$ ou 4

- S'il existe des sommets non étiquetés adjacents à des sommets étiquetés 1, 2, 3 ou 4, nous les étiquetons en utilisant les règles [B1a] et [B1b].

  [B1a] Supposons que x soit un sommet non étiqueté, voisin d'un sommet u étiqueté 1, 2, 3 ou 4, et que les autres voisins de u soient tous étiquetés (voir la figure 2.14). Alors l'étiquette de x est déterminée de façon unique. L'étiquetage de x diminue de 1 le nombre de sommets non étiquetés et peut diminuer d'au plus 1 la taille de $\tilde{M}$ (d'après le lemme 2.10). On obtient la récurrence

  $$T(\mu) \leq T(\mu - 1 + \epsilon).$$

  [B1b] Supposons que x et y soient deux sommets non étiquetés adjacents à un sommet u étiqueté par $1, 2, 3$ ou 4 (voir la figure 2.15). Il y a deux possibilités d'étiquetages pour x et y, puisque u a déjà un autre voisin u' étiqueté. Si chacun de ces deux étiquetages

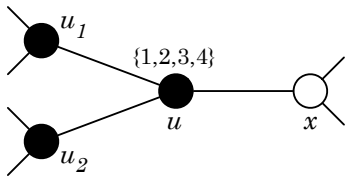


FIGURE 2.14 – illustration de la règle [B1a].

diminue la taille de $\tilde{M}$ d'au plus 1, alors l'algorithme branche sur x et y selon ces deux étiquetages. On obtient alors la récurrence

$$T(\mu) \leq 2 \cdot T(\mu - 2 + \epsilon).$$

Dans le cas où, par cet étiquetage, la taille de $\tilde{M}$ diminue d'au moins 2, le lemme 2.11 garantit que x et y ont chacun un voisin distinct, différent de u, et étiqueté par une étiquette de $\{1,2,3,4\}$. Notons u_1 (respectivement u_2) ce voisin de x (respectivement de y). Puisque la règle [B1a] ne s'applique pas, le sommet u_1 a un autre voisin x' non étiqueté, $x' \neq x$. En étiquetant x, l'étiquette de x' devient forcée et nous étiquetons ce sommet x'. De plus, on observe que l'étiquetage de x, y et x' ne peut faire diminuer le couplage induit $\tilde{M}$ que d'au plus 2. On obtient donc la récurrence

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

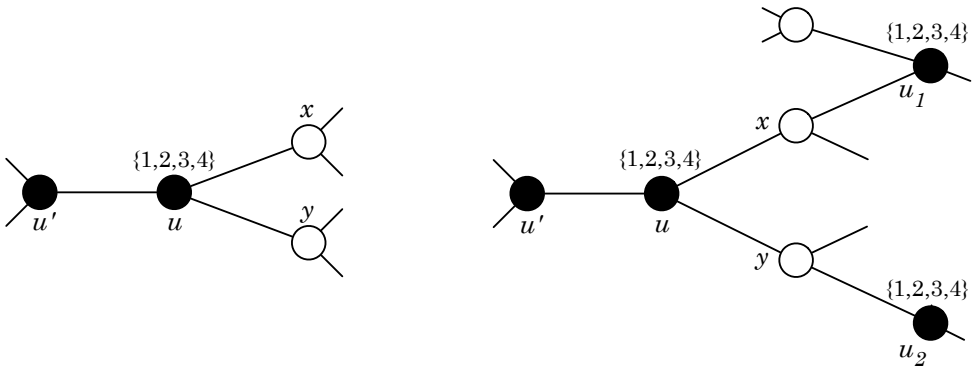


FIGURE 2.15 – Illustration de la règle [B1b].

Si aucune des règles [B0] et [B1] ne s'applique, alors tous les sommets non étiquetés, ayant des voisins étiquetés, ne sont voisins que de sommets étiquetés par 0 ou par 5. Par conséquent l'ensemble $\tilde{M}$ est vide.

Règle [B2] – deux voisins étiquetés 0 ou 5

- S'il existe un sommet non étiqueté x voisin de deux sommets u et v étiquetés, alors l'un d'eux est étiqueté par 0 et l'autre par 5, et x a un voisin y non étiqueté (voir la figure 2.16). Si ce n'est pas le cas, alors l'étiquette de u et v est identique et on arrête l'étiquetage de ce

sous-problème. Nous avons deux possibilités pour l'étiquetage de x et y. Chacune d'elles attribue une étiquette de $\{1,2,3,4\}$ à y. Ainsi, la règle [B1] peut s'appliquer et $\tilde{M}$ sera non vide. Donc la mesure μ diminue d'au moins 2 et nous obtenons la récurrence

$$T(\mu) \leq 2 \cdot T(\mu - 2).$$

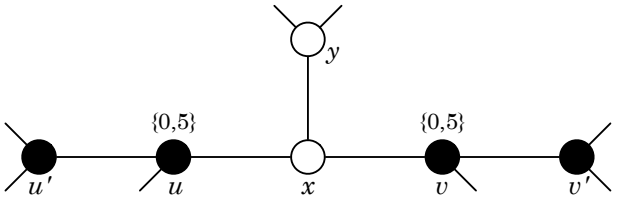


FIGURE 2.16 – Illustration de la règle [B2].

À présent, si aucune des règles [B0], [B1], [B2] ne s'applique, alors chaque sommet non étiqueté est adjacent à au plus un sommet étiqueté 0 ou 5.

Règle [B3] – deux sommets avec un voisin commun étiqueté 0 ou 5

- Si l'on a deux sommets non étiquetés x et y ayant un voisin commun u, alors on utilise [B3a], [B3b] ou [B3c] pour étiqueter x et y (et éventuellement d'autres sommets). On note que par l'application des règles précédentes, le sommet u a pour étiquette 0 ou 5.

  [B3a] Supposons que x et y soient adjacents (voir la figure 2.17). Nous avons au plus 4 possibilités pour étiqueter x et y. Pour chacun de ces étiquetages, la taille du couplage $\tilde{M}$ augmente d'au moins 1. Cela donne la récurrence

$$T(\mu) \leq 4 \cdot T(\mu - 2 - \epsilon).$$

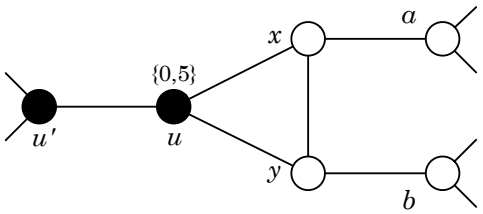


FIGURE 2.17 – Illustration de la règle [B3a].

  [B3b] Supposons que x et y ne soient pas adjacents. Supposons que x et y aient un voisin commun a non étiqueté. Puisque [B2] ne peut être appliquée et que G est un graphe cubique, il s'ensuit que x et y ont chacun un autre voisin non étiqueté. Notons b (respectivement b') cet autre voisin de x (respectivement de y) (voir la figure 2.18). Il y a au plus 6 façons d'étiqueter x et y. Au moins 4 d'entre elles déterminent de façon unique l'étiquette de deux sommets : celle de a et celle de b ou b'. (Certains étiquetages indiquent aussi que l'étiquetage courant ne peut pas être étendu, et nous arrêtons dans

ce cas la recherche des étiquetages pour le sous-problème.) Dans les deux autres cas, au moins une arête est ajoutée à $\tilde{M}$. On obtient la récurrence

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

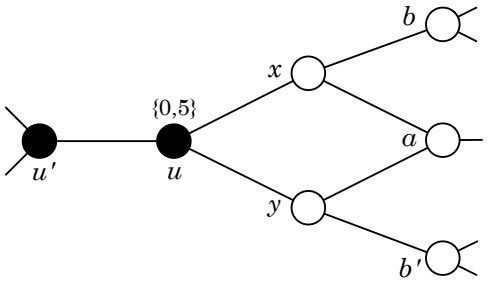


FIGURE 2.18 – Illustration de la règle [B3b].

[B3c] Supposons que x et y ne soient pas adjacents. Nous pouvons maintenant supposer que x et y n'ont pas de voisin commun non étiqueté (voir la figure 2.19). Il y a 6 possibilités d'étiqueter x et y. Au moins 2 d'entre elles attribuent à x et y une étiquette de $\{1,2,3,4\}$. Cela ajoute au moins deux arêtes à $\tilde{M}$. Dans les autres cas, au moins une arête est ajoutée à $\tilde{M}$. D'où la récurrence

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

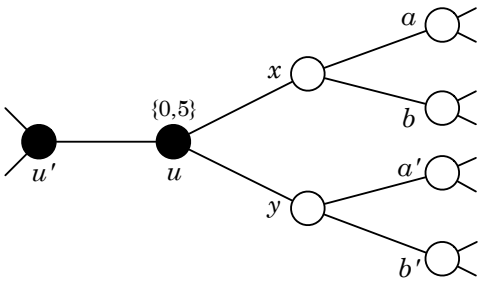


FIGURE 2.19 – Illustration de la règle [B3c].

À présent le graphe ne contient plus de sommets étiquetés ayant deux voisins non étiquetés.

Règle [B4] – un voisin étiqueté 0 ou 5

- Soit x un sommet non étiqueté ayant un voisin u étiqueté par 0 ou 5 (voir la figure 2.20). Il y a deux étiquettes possibles pour x, dont au moins l'une appartient à $\{1,2,3,4\}$ et ajoute une arête à $\tilde{M}$. La récurrence que nous obtenons est

$$T(\mu) \leq T(\mu - 1) + T(\mu - 1 - \epsilon).$$

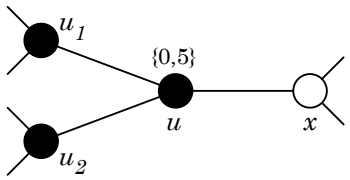


FIGURE 2.20 – Illustration de la règle [B4].

2.2.2.2 Analyse de l'algorithme

La preuve de correction de l'algorithme est facile à expliquer. L'ensemble des cas sont traités dans l'algorithme et celui-ci branche sur toutes les extensions possibles pour étendre l'étiquetage courant à d'autres sommets.

Concernant l'analyse du temps d'exécution, en fixant $\epsilon = 0.819531$, nous établissons que $T(\mu) = \mathcal{O}(1.7990^{\mu})$. La valeur de ϵ a été choisie de sorte à minimiser la base de l'exponentielle dans l'expression du temps d'exécution.

Remarque 4. *Pour $\epsilon = 0.819531$, les récurrences correspondantes aux cas [B0] et [B3c] sont celles ayant le vecteur de branchement le plus important.*

Théorème 2.12 (Couturier *et al.* (2013a), théorème 3)**.** *L'algorithme de* branchement *énumère tous les étiquetages $L(2,1)$ de largeur 5 d'un graphe connexe cubique en temps $\mathcal{O}(1.7990^n)$.*

Une application des algorithmes d'énumération (en temps exponentiel) est l'obtention de bornes combinatoires. Dans notre cas, puisque l'algorithme énumère tous les étiquetages possibles, son temps d'exécution implique immédiatement une borne sur le nombre de tels étiquetages.

Corollaire 2.13 (Couturier *et al.* (2013a), corollaire 3)**.** *Le nombre d'étiquetages $L(2,1)$ de largeur 5 d'un graphe connexe cubique à n sommets est $\mathcal{O}(1.7990^n)$.*

2.2.2.3 Une borne inférieure sur le nombre d'étiquetages $L(2,1)$ de largeur 5

Nous établissons une borne inférieure de $2^{2n/7}$ sur le nombre d'étiquetages $L(2,1)$ de largeur 5. Cette borne semble éloignée de la borne supérieure que nous avons établie au corollaire 2.13. Il faut noter que la borne du corollaire 2.13 est directement obtenue de l'algorithme et de son temps d'exécution établi au théorème 2.12. Or l'algorithme essaye aussi des étiquetages qui finalement, à un moment de l'exécution, ne peuvent pas être étendus (ce qui coûte du temps d'exécution, sans faire augmenter le nombre d'étiquetages valides). D'autres part, il y a de bonnes chances pour que ni la borne du corollaire 2.13, ni celle du théorème 2.14 (donné ci-après) ne soient serrées, dans le sens où la borne du corollaire 2.13 est certainement sur-estimée et celle du théorème 2.14 peut certainement être améliorée par une autre construction. Dans les deux cas, une amélioration n'est pas triviale et demande des efforts.

Théorème 2.14 (Couturier *et al.* (2013a), proposition 2)**.** *Il existe des graphes connexes cubiques à n sommets ayant au moins $2^{2n/7}$ étiquetages $L(2,1)$ de largeur 5.*

Démonstration. Soit G le graphe représenté à la figure 2.21. On commence par construire un graphe F, obtenu d'un K_4 où l'on dénote les sommets par $\{x_1, x_2, x_3, x_4\}$, puis en subdivisant l'arête $\{x_1, x_2\}$ avec un nouveau sommet x_0. Soient $F_1, F_2, ..., F_{2t}$ des copies du graphe F. On y ajoute un cycle $C = \{v_1, v_2, ..., v_{4t}\}$ ainsi que les arêtes $\{v_{4i}, v_{4i-2}\}$ pour $i = 1, 2, ..., t$. On connecte chaque sommet v_{2j-1} avec le sommet x_0 du graphe F_j, pour $j = 1, 2, ..., 2t$. On obtient ainsi le graphe G à $n = 14t$ sommets dont on vérifie facilement qu'il est connexe et cubique.

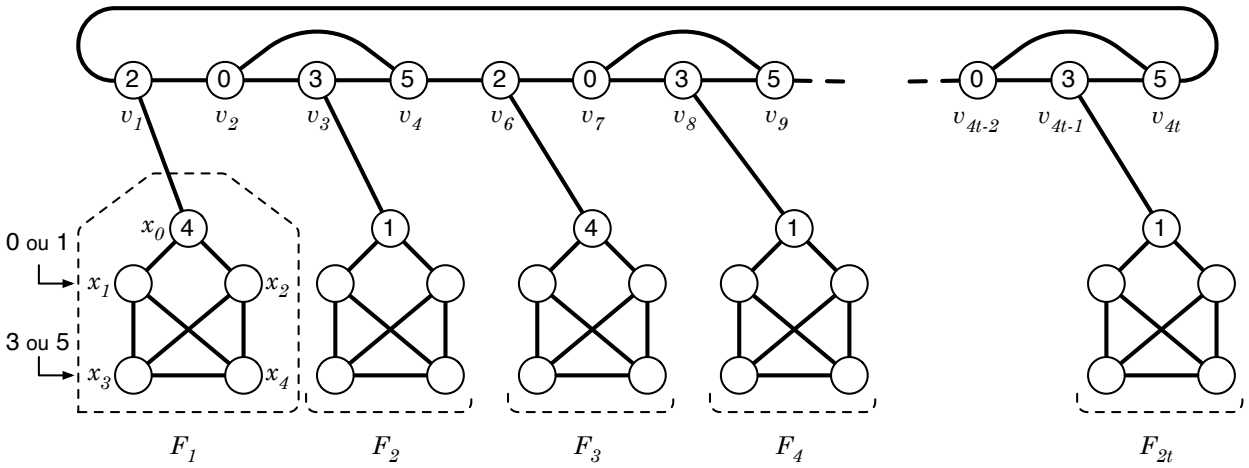


FIGURE 2.21 – Le graphe connexe cubique ayant au moins $2^{2n/7}$ étiquetages $L(2,1)$ de largeur 5. Les sommets x_1 et x_2 ont des étiquettes différentes choisies dans $\{0,1\}$; les sommets x_3 et x_4 ont des étiquettes différentes choisies dans $\{3,5\}$.

Considérons c un étiquetage $L(2,1)$ construit de la façon suivante. Pour tout $i = 1, 2, ..., t$, on pose $c(v_{4i-3}) = 2$, $c(v_{4i-2}) = 0$, $c(v_{4i-1}) = 3$, $c(v_{4i}) = 5$.

Si j est impair alors on étiquette le sommet x_0 de F_j par 4, les sommets de l'ensemble $\{x_1, x_2\}$ par des étiquettes de l'ensemble $\{0,1\}$, les sommets de l'ensemble $\{x_3, x_4\}$ par des étiquettes de $\{3,5\}$.

Si j est pair alors on étiquette le sommet x_0 de F_j par 1, les sommets de l'ensemble $\{x_1, x_2\}$ par des étiquettes de l'ensemble $\{4,5\}$, les sommets de l'ensemble $\{x_3, x_4\}$ par des étiquettes de $\{0,2\}$.

On observe facilement qu'il y a ainsi 4 façons d'étiqueter les sommets de chaque graphe F_j. Donc le nombre total d'étiquetages est $4^{2t} = 4^{n/7}$. □

Le graphe décrit dans la preuve est celui pour lequel nous avons obtenu le plus grand nombre d'étiquetages. Il ne serait pas surprenant qu'il existe des graphes avec davantage d'étiquetages. Néanmoins, la contrainte que ces graphes soient connexes et cubiques rend leur découverte (ou plutôt construction) assez compliquée. D'un autre côté, la borne inférieure (relativement petite) du théorème précédent montre qu'il est difficile de trouver des étiquetages de largeur 5 « à la main » car il y en a peu. D'autre part, cela laisse supposer que décider l'existence d'un étiquetage de largeur 5 dans un graphe cubique doit être beaucoup plus rapide que notre algorithme actuel d'énumération.

2.3 Étiquetage $L(2,1)$ de largeur minimum

À la section 2.2, nous nous sommes intéressés aux étiquetages $L(2,1)$ de largeur 4 et 5. Nous présentons maintenant des algorithmes qui, étant donné un graphe G, calculent la plus petite

valeur k pour laquelle $\lambda_{2,1}(G) = k$. Dans un premier temps, nous donnerons un algorithme dont le temps d'exécution est $\mathcal{O}((k-2.5)^n)$ (Havet *et al.*, 2011). Cet algorithme ressemblera à ceux de la section 2.2, avec des *règles de branchement* et ne nécessitera qu'un espace polynomial. Naturellement, nous avons très vite l'envie d'avoir un algorithme dont le temps d'exécution ne dépende pas (ou peu) de $\lambda_{2,1}(G)$. Nous présenterons différents algorithmes pour lesquels la base de l'exponentielle est une constante indépendante de $\lambda_{2,1}(G)$. Nous présenterons d'abord un algorithme en temps $\mathcal{O}(7.4920^n)$, qui reste en espace polynomial et exploite la stratégie *diviser-pour-régner* (Junosza-Szaniawski *et al.*, 2013b). Puis, nous donnerons un algorithme basé sur de la *programmation dynamique* en temps $\mathcal{O}(3.8730^n)$, mais au coût d'un espace exponentiel (Havet *et al.*, 2011). Finalement, nous présenterons un algorithme en temps $\mathcal{O}(2.6488^n)$ et espace exponentiel (Junosza-Szaniawski *et al.*, 2013a). Celui-ci est basé sur une utilisation astucieuse de la *programmation dynamique*, avec une représentation compacte des étiquetages partiels.

Nous commençons par présenter les algorithmes ne nécessitant qu'un espace polynomial, puis ceux demandant un espace exponentiel.

2.3.1 Espace polynomial

2.3.1.1 Algorithme de branchement

Nous avons mentionné au corollaire 2.3 la possibilité de déterminer un étiquetage $L(2,1)$ de largeur k, s'il en existe, en temps $\mathcal{O}^*((k-2)^n)$. Les détails de cette preuve sont disponibles dans (Havet *et al.*, 2011). Avec quelques règles de branchement, nous pouvons prétendre à un algorithme en temps $\mathcal{O}^*((k-2.5)^n)$ et espace polynomial. Contrairement aux algorithmes de la section 2.2, nous ne visons pas dans cette section à obtenir la meilleure complexité possible, au prix de règles de branchement compliquées et d'une analyse de complexité raffinée. Nous montrons le théorème suivant :

Théorème 2.15 (Havet *et al.* (2011), théorème 8)**.** *Décider si un graphe admet un étiquetage $L(2,1)$ de largeur k, pour un $k \geq 5$ fixé, peut être effectué en temps $\mathcal{O}((k-2.5)^n)$ et espace polynomial.*

Nous décrivons ci-après notre algorithme qui calcule un étiquetage de largeur k. Encore une fois, l'asymétrie des étiquettes va nous servir : les sommets étiquetés 0 ou k autorisent $k-1$ étiquettes possibles dans leur voisinage, alors que les sommets d'étiquettes dans $[1,\dots,k-1]$ n'autorisent que $k-2$ étiquettes possibles dans leur voisinage. Comme à la section 2.2, nous commençons par choisir arbitrairement deux sommets adjacents et nous essayons les (au plus) k^2 étiquetages possibles de ces deux sommets. À chaque étape de branchement, nous conservons pour invariant que l'ensemble des sommets étiquetés forment un sous-graphe connexe. Les trois règles de branchement sont décrites ci-après et appliquées dans cet ordre.

Règle 1 – branchements simples

(a) Si un sommet v a une étiquette dans $[1,\dots,k-1]$ et a au moins un voisin u non étiqueté, alors on branche selon tous les étiquetages possibles de u Puisque v a une étiquette dans $[1,\dots,k-1]$, cela interdit 3 étiquettes pour le sommet u. De plus, v a au moins un voisin étiqueté, ce qui interdit encore une étiquette de plus. Ainsi, on obtient la récurrence

$$T(n) \leq (k-3) \cdot T(n-1).$$

(b) Si un sommet v étiqueté a deux voisins étiquetés et un voisin non étiqueté u, on branche selon tous les étiquetages possibles pour u. Dans ce cas, v interdit au moins 2 étiquettes et ses (au moins) deux voisins étiquetés en interdisent 2 de plus. On a donc $k-3$ étiquettes possibles pour u et la récurrence

$$T(n) \leq (k-3) \cdot T(n-1).$$

(c) Si un sommet v étiqueté a deux voisins non étiquetés u et w, on branche selon les étiquetages possibles pour u et w. Il y a au plus $k-2$ étiquettes possibles pour u et, une fois que u est étiqueté, il reste $k-3$ étiquettes possibles pour w. Cela donne un total de $(k-2)(k-3) < (k-2.5)^2$ possibilités et la récurrence

$$T(n) \leq (k-2.5)^2 \cdot T(n-2).$$

(d) Si un sommet u non étiqueté a deux voisins étiquetés, on branche selon les étiquetages possibles de u. Puisque les voisins étiquetés de u ont des étiquettes différentes, ils interdisent au moins 4 étiquettes pour u. On a donc

$$T(n) \leq (k-3) \cdot T(n-1).$$

(e) Si un sommet u non étiqueté a un voisin étiqueté et deux voisins v, w non étiquetés, on branche selon les étiquetages possibles de u, v, et w. Sans perte de généralité, supposons que le voisin étiqueté de u a pour étiquette 0. Alors u ne peut pas être étiqueté 0 ou 1. Si son étiquette est dans $[2, \ldots, k-1]$, alors il reste $k-3$ étiquettes possibles pour v, puis $k-4$ étiquettes pour w. Il y a donc $(k-2)(k-3)(k-4)$ étiquetages possibles pour u, v, et w. Si u est étiqueté par k, il reste $k-2$ étiquettes possibles pour v, puis $k-3$ pour w. Le nombre total d'étiquetages possibles pour ces trois sommets est donc $(k-2)(k-3)(k-4) + (k-2)(k-3)$ qui est plus petit que $(k-2.5)^3$ pour $k \geq 5$. D'où la récurrence

$$T(n) \leq (k-2.5)^3 \cdot T(n-3).$$

La figure 2.22 représente les configurations correspondant aux cinq cas de la règle 1.

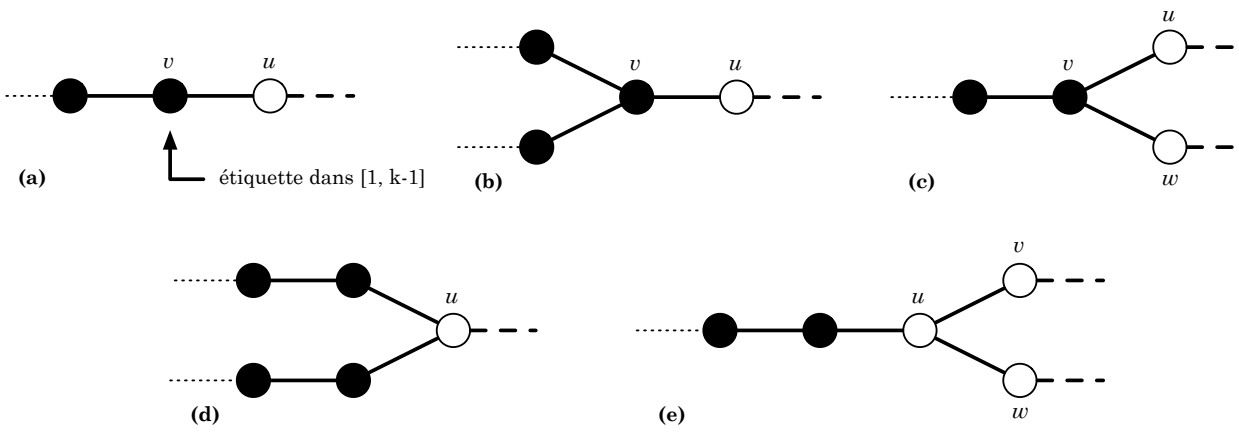


FIGURE 2.22 – Branchements simples de la règle 1.

Si la règle 1 ne peut s'appliquer alors chaque sommet étiqueté ayant un voisin non étiqueté, a pour degré 2, pour étiquette 0 ou k, et seulement un voisin non étiqueté. Ce sommet non étiqueté a seulement un voisin étiqueté et pour degré au plus 2. Il appartient donc à un *chemin d'extension* (un chemin constitué de sommets non étiquetés de degré 2, se terminant avec un sommet non étiqueté de degré 1 ou au moins 3, ou se terminant par un sommet étiqueté de degré 2). Nous traitons ces chemins d'extension (par la règle 2 donnée ci-après) de façon similaire à l'algorithme de la section 2.2.1.1.

Règle 2 – extensions économiques

La figure 2.23 présente les deux cas suivants :

(a) Si un chemin d'extension se termine avec un sommet non étiqueté de degré 1, alors on peut supprimer ce chemin et continuer avec le graphe réduit (il est toujours possible de déterminer à la fin, en temps polynomial, si un étiquetage de ce chemin existe).

(b) Si un chemin d'extension se termine avec un sommet étiqueté, on vérifie par *programmation dynamique* s'il existe un étiquetage du chemin compatible avec l'étiquetage de ses extrémités (et de leur voisinage).

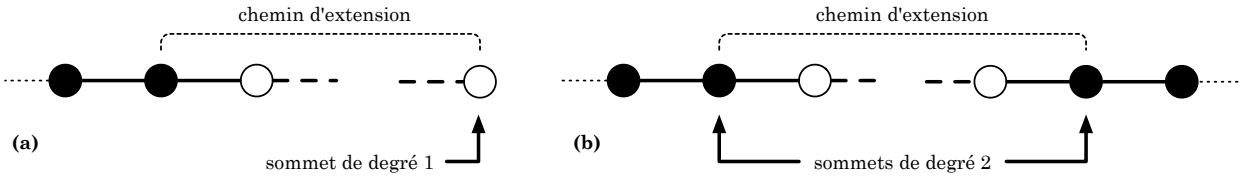


FIGURE 2.23 – Extensions économiques de la règle 2. (a) Un chemin d'extension avec une extrémité non étiquetée de degré 1. (b) Un chemin d'extension dont les deux extrémités sont étiquetées et de degré 2.

Concernant la contribution de la règle 2 au temps d'exécution, celle-ci est au plus polynomiale. En effet, la règle ne demande aucun branchement et n'augmente pas la taille du graphe. De plus, on peut noter qu'elle est appliquée à un sous-problème au plus n fois, puisqu'à chaque fois au moins un sommet est supprimé ou étiqueté.

Règle 3 – branchement sur chemin d'extension

Si un chemin d'extension se termine avec un sommet u non étiqueté de degré au moins 3, alors tous les voisins de u sont non étiquetés (sinon la règle 1 (e) aurait été appliquée). Notons x le voisin de u sur le chemin d'extension et y, z deux autres voisins (voir la figure 2.24). On branche selon tous les étiquetages possibles des sommets u, x, y, z et, pour chacun d'eux, on vérifie par *programmation dynamique* si cet étiquetage peut être étendu au chemin.

Notons $bu_1u_2 \dots u_{t-1}u$ le chemin d'extension, a le voisin étiqueté de b, u l'extrémité non étiquetée de degré au moins 3. Notons y et z deux voisins non étiquetés de u n'appartenant pas au chemin. Nous pouvons supposer que $t \geq 2$, puisque le cas $t = 1$ est traité par la règle 1 (e). Sans perte de généralité, nous pouvons supposer que b est étiqueté 0.

Supposons d'abord que $t = 2$. Si le sommet u est étiqueté par une étiquette de $[2, \dots, k-1]$, alors nous avons au plus $k-3$ étiquettes possibles pour $x = u_1$, $k-3$ étiquettes pour y et

$k-4$ étiquettes pour z. Si u est étiqueté par 1, nous avons $k-2$ étiquettes possibles pour $x = u_1$, $k-3$ étiquettes pour y et $k-4$ étiquettes pour z. Si u est étiqueté par k, nous avons $k-3$ étiquettes possibles pour $x = u_1$, $k-2$ étiquettes pour y et $k-3$ étiquettes pour z. Ainsi, il y a $(k-2)(k-3)(k-3)(k-4) + (k-2)(k-3)(k-4) + (k-3)(k-2)(k-3)$ étiquetages possibles pour u_1, u, y, z (u ne peut pas être étiqueté par 0 qui est l'étiquette de b). On a $(k-2)(k-3)(k-3)(k-4) + (k-2)(k-3)(k-4) + (k-3)(k-2)(k-3) < (k-2.5)^4$ pour $k \geq 5$ et la récurrence

$$T(n) \leq (k-2.5)^4 \cdot T(n-4).$$

Supposons maintenant que $t \geq 3$. Si u est étiqueté par 0 ou k, on obtient à chaque fois $(k-1)(k-2)(k-3)$ possibilités pour étiqueter x, y, z. Si u obtient une étiquette de $[1, \ldots, k-1]$, pour chacun de ces étiquetages nous avons $(k-2)(k-3)(k-4)$ possibilités pour étiqueter x, y, z. Ainsi, le nombre d'étiquetages possibles pour u, x, y, z est $2(k-1)(k-2)(k-3) + (k-1)(k-2)(k-3)(k-4)$, qui est inférieur à $(k-2.5)^5$ pour $k \geq 5$. Pour chacun de ces étiquetages nous vérifions en temps polynomial, s'ils sont compatibles avec les étiquettes de a et b afin de s'étendre à un étiquetage du chemin d'extension. D'où la récurrence

$$T(n) \leq (k-2.5)^5 \cdot T(n-5).$$

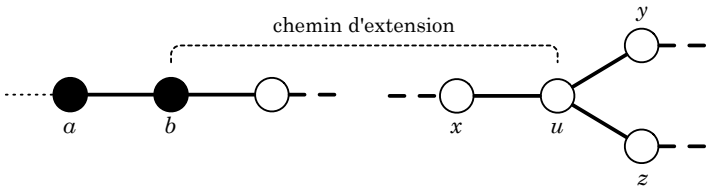


FIGURE 2.24 – Branchement sur un chemin d'extension de la règle 3. Un chemin d'extension avec une extrémité u non étiquetée de degré au moins 3, sans voisin étiqueté.

Puisque l'application de n'importe laquelle des règles donne une récurrence $T(n) \leq (k-2.5)^\ell T(n-\ell)$ pour un certain ℓ, nous établissons que le nombre maximum de feuilles dans l'arbre de recherche correspondant à une exécution de l'algorithme est au plus $\mathcal{O}((k-2.5)^n)$. Cela démontre le théorème 2.15.

2.3.1.2 Stratégie diviser-pour-régner

Nous présentons dans cette section le meilleur algorithme connu pour calculer $\lambda_{2,1}(G)$ en espace polynomial, dont la complexité exponentielle ne dépend pas de $\lambda_{2,1}(G)$. Cet algorithme est basé sur le paradigme *diviser-pour-régner* et l'analyse de son temps d'exécution est donné au théorème suivant :

Théorème 2.16 (Junosza-Szaniawski *et al.* (2013b), théorème 1)**.** *La largeur minimum d'un étiquetage $L(2,1)$ d'un graphe à n sommets peut être calculée en temps $\mathcal{O}(7.4920^n)$ et espace polynomial.*

Notions préliminaires

L'algorithme que nous présentons résout un problème plus général. Avant de définir ce problème[5], nous avons besoin de la notion de *rb-graphes* :

Définition 2.2. Un triplet $G = (V, R, B)$ est un *rb-graphe* si V est un ensemble fini (correspondant aux sommets du graphe) et R, B (correspondant aux arêtes de couleurs rouges et noires) sont des familles disjointes d'ensembles à 2-éléments de V satisfaisant la condition ($\triangle$) :

($\triangle$) si $vw \in B$ et $vu \in B$ alors $uw \in R \cup B$, pour tous sommets u, v, w de V.

Nous pouvons maintenant décrire le problème que nous allons résoudre :

ÉTIQUETAGE L_Q^P

Entrée. Un rb-graphe $G = (V, R, B)$, deux ensembles $P, Q \subseteq V$ et un entier k.

Question. Existe-t-il une application φ de V dans $\{1, 2, \ldots, k\}$ telle que :

1. $|\varphi(u) - \varphi(v)| \geq 2$ pour tout $\{u, v\} \in B$;
2. $|\varphi(u) - \varphi(v)| \geq 1$ pour tout $\{u, v\} \in R$;
3. $Q \cap \varphi^{-1}(1) = \varnothing$;
4. $P \cap \varphi^{-1}(k) = \varnothing$.

Nous notons $\Lambda_Q^P(G)$ le problème d'optimisation qui demande de déterminer un étiquetage L_Q^P de largeur k minimum.

Le problème ÉTIQUETAGE L_Q^P n'est pas sans lien avec le problème ÉTIQUETAGE $L(2,1)$. En effet, définissons la *R-fermeture* d'un graphe $G = (V, E)$ comme le rb-graphe $H = (V, R, B)$ ayant le même ensemble de sommets V et où $B = E$ et $R = E(G^2) \setminus E(G) = \{\{u, w\} : \exists v \in V, \{u, v\} \in E, \{v, w\} \in E, \{u, w\} \notin E\}$. (Voir aussi la figure 2.25 pour un exemple.) On remarque qu'un étiquetage $L_\varnothing^\varnothing$ (c'est-à-dire, où P et Q sont des ensembles vides) de largeur $k+1$ de la R-fermeture H d'un graphe G correspond à un étiquetage $L(2,1)$ de largeur k de G (à 1 près dans les étiquettes, puisque par facilité on n'utilisera pas l'étiquette 0). On obtient ainsi le lemme suivant :

Lemme 2.17. *Soit $G = (V, E)$ un graphe et soit $H = (V, R, B)$ sa R-fermeture. Alors $\Lambda_\varnothing^\varnothing(H) = \lambda_{2,1}(G)+1$.*

Nous pouvons également étendre la notion de *sous-graphe induit* aux rb-graphes. Pour un rb-graphe $G = (V, R, B)$ et un ensemble de sommets $V' \subseteq V$, le sous-graphe induit $G[V']$ est le rb-graphe défini naturellement par $(V', \{e \in R : e \subseteq V'\}, \{e \in B : e \subseteq V'\})$.

Pour établir notre algorithme puis son temps d'exécution, nous avons besoin d'objets combinatoires un peu particuliers (que nous retrouverons également dans l'algorithme de *programmation dynamique* de la section 2.3.2.2). Ces objets sont le pendant des *ensembles stables* utiles pour résoudre le problème COLORATION. En effet, dans une coloration propre d'un graphe G, chaque classe de couleur forme un ensemble stable de G. Pour l'étiquetage $L(2,1)$ (et l'étiquetage L_Q^P), ce ne sont pas les *ensembles stables* qui forment les classes de couleurs, mais les *ensembles stables* dans le graphe G^2. Ces ensembles s'appellent également des 2-packings (ou 2-stables) de G. Pour nos

[5] Pour des raisons de commodité, nous n'utilisons pas l'étiquette 0 dans cette section.

besoins algorithmiques, il nous faut un peu raffiner ces objets. Pour COLORATION, il n'y a pas de dépendance entre les classes de couleurs, c'est-à-dire que nous pouvons facilement permuter les couleurs de deux classes de couleurs. Pour l'étiquetage $L(2,1)$, un sommet d'étiquette i ne peut pas être voisin d'un sommet d'étiquette $i-1$ ou $i+1$. L'interaction entre les classes de couleurs (formées par les sommets ayant la même étiquette) est beaucoup plus forte. En particulier, on ne peut pas nécessairement permuter les étiquettes de deux classes de couleurs. Nous commençons donc par définir de nouveaux objets (dans cette section, les *partitions correctes* seront centrales alors qu'à la section 2.3.2.2, ce seront les *paires propres* que nous définirons). Des bornes combinatoires sur leur nombre maximum seront établies à la section 2.3.3. Elles nous permettront de borner le temps d'exécution de nos algorithmes.

Définition 2.3. On définit les objets combinatoires et propriétés suivantes (voir également la figure 2.25) :

- Un ensemble de sommets Y est un *ensemble stable* s'il ne contient pas de paires de sommets adjacents (s'il s'agit d'un rb-graphe $G(V,R,B)$, peu importe la couleur des arêtes, Y doit être un stable du graphe $(V, R \cup B)$).
- L'ensemble stable Y est *R-maximal* si tout sommet v tel que $N_R(v) = N(v)$ est soit dans Y, soit v a un voisin dans Y (où $N_R(v)$ désigne l'ensemble des voisins de v accessibles par une arête de R). On note que par définition, Y est un stable du graphe $(V, R \cup B)$.
- Une paire (X,Y) d'ensembles disjoints de sommets est une *paire propre R-maximale* si Y est un ensemble stable R-maximal.
- Un triplet (X,Y,Z) est une *partition balancée* si
  1. les ensembles X,Y,Z forment une partition des sommets V ;
  2. l'ensemble Y est stable ;
  3. les ensembles X,Y,Z sont non-vides ;
  4. $|X|, |Z| \leq \frac{|V|}{2}$.
- Le triplet (X,Y,Z) est une *partition correcte* si elle est balancée et Y est un ensemble stable R-maximal.

On peut immédiatement faire l'observation suivante :

Observation 1. *Si (X,Y,Z) est une partition correcte, alors (X,Y) est une paire propre R-maximale. Ainsi, le nombre maximum de paires propres R-maximales dans un rb-graphe G est aussi un majorant sur le nombre de partitions correctes de G.*

À la section 2.3.3, nous établissons une borne supérieure sur le nombre de paires propres R-maximales. Nous allons d'abord observer l'utilité des partitions correctes pour un algorithme de type *diviser-pour-régner*.

Description de l'algorithme

Notre algorithme est basé sur le paradigme classique *diviser-pour-régner*. De façon classique avec cette stratégie, on sépare le problème en (typiquement) deux sous-problèmes, qui sont résolus récursivement, puis on combine les solutions à ces sous-problèmes. Nous utilisons ici cette stratégie

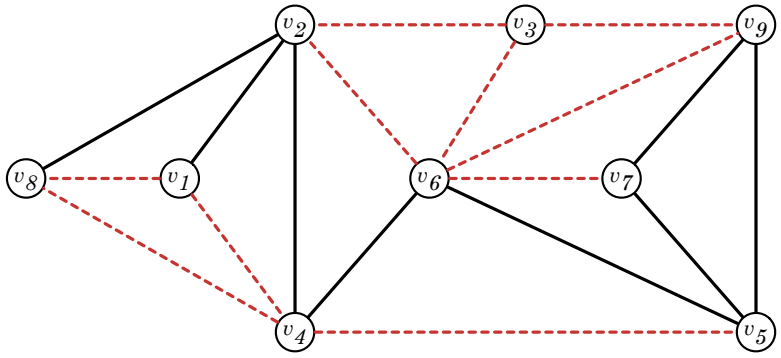


FIGURE 2.25 – Un *rb-graphe* $G = (V, R, B)$ où les arêtes noires sont représentées en traits continus noirs et les arêtes rouges sont représentées en traits discontinus rouges. Ce rb-graphe représente la *R-fermeture* du graphe $H = (V, B)$. L'ensemble $\{v_1, v_7\}$ représente un ensemble *stable* de G et de H. Notons que cet ensemble n'est pas maximal puisqu'il est possible d'y ajouter le sommet v_3. Le seul sommet de ce graphe qui vérifie $N_R(v) = N(v)$ est le sommet v_3. Ainsi, $\{v_1, v_7\}$ n'est pas un *ensemble stable R-maximal*. Par contre, les ensembles $\{v_4, v_9\}$, $\{v_3, v_8\}$ ou $\{v_3, v_8, v_7\}$ forment des *ensembles stables R-maximaux* de G. La paire $(X = \{v_1, v_5, v_9\}, Y = \{v_3, v_8\})$ est une *paire propre R-maximale*. Le triplet $(X = \{v_1, v_3, v_5, v_6\}, Y = \{v_4, v_9\}, Z = \{v_2, v_7, v_8\})$ est l'une des *partitions correctes* de G.

de façon à obtenir un algorithme exponentiel au problème ÉTIQUETAGE $L(2,1)$, dans le but de résoudre le problème (plus restreint) ÉTIQUETAGE $L(2,1)$.

On peut informellement décrire notre algorithme. Supposons que $f : V \to \{0, 1, \ldots, k\}$ soit un étiquetage $L(2,1)$ d'un graphe $G = (V, E)$. Notons $\{\mathcal{C}_0, \mathcal{C}_1, \ldots, \mathcal{C}_k\}$ les classes d'étiquettes, c'est-à-dire $\mathcal{C}_i = \{v \in V \ : \ f(v) = i\}$, pour $i \in \{0, 1, \ldots, k\}$. Pour retrouver cette fonction d'étiquetage f, à chaque étape il nous « suffit » de deviner une classe de couleur $\mathcal{C}_\ell$ telle que $\sum_{i<\ell} |\mathcal{C}_i| \approx \sum_{i>\ell} |\mathcal{C}_i|$, puis de résoudre récursivement notre problème d'étiquetage sur $G[\cup_{i<\ell} \mathcal{C}_i]$ et $G[\cup_{i>\ell} \mathcal{C}_i]$. On s'aperçoit que cela n'est pas si simple car les sommets de $\mathcal{C}_\ell$ (ceux que nous avons devinés et que nous étiquetons par ℓ) ont une influence sur leurs voisins appartenant à $\cup_{i<\ell} \mathcal{C}_i$ et à $\cup_{i>\ell} \mathcal{C}_i$. C'est pour surmonter cette difficulté que nous avons introduit le problème ÉTIQUETAGE $L(P,Q)$ (et ses ensembles P et Q). Nous décrivons maintenant formellement l'algorithme et son idée principale basée sur les lemmes 2.18 et 2.19.

Lemme 2.18. *Soit φ un étiquetage $L(P,Q)$ de largeur k d'un rb-graphe $G = (V, R, B)$. Pour $h \in \{1, \ldots, k\}$, posons $X_h = \cup_{i=1}^{h-1} \varphi^{-1}(i)$, $Y_h = \varphi^{-1}(h)$ et $Z_h = \cup_{i=h+1}^{k} \varphi^{-1}(i)$. Il existe un $\ell \in \{1, \ldots, k\}$ pour lequel l'un des cas suivants est satisfait :*

1. *$X_\ell = \varnothing$ et $|Y_\ell| \geq |V|/2$;*
2. *$Z_\ell = \varnothing$ et $|Y_\ell| \geq |V|/2$;*
3. *le triplet (X_ℓ, Y_ℓ, Z_ℓ) est une partition balancée de G.*

Démonstration. Nous montrons que, si aucun des deux premiers cas n'apparaît, alors il existe un ℓ tel que le troisième cas est satisfait. Tout d'abord, notons que pour tout $h \in \{1, \ldots, k\}$, les ensembles X_h, Y_h, Z_h forment une partition de V et que Y_h est un ensemble stable. Soit ℓ le plus petit nombre tel que $|X_\ell \cup Y_\ell| \geq |V|/2$. Clairement, Y_ℓ est non vide car sinon $\ell - 1$ aurait été choisi. Puisque les deux premiers cas n'apparaissent pas et par le choix de ℓ, les deux ensembles X_ℓ et Z_ℓ sont non vides. Donc $|X_\ell| \leq n/2$, avec $n = |V|$. De plus, comme $|X_\ell \cup Y_\ell| \geq |V|/2$, on a également $|Z_\ell| \leq n/2$. □

Lemme 2.19. *Soit $k = \Lambda_P^Q(G)$ et $\ell \in \{2, \ldots, k-1\}$. Il existe un étiquetage $L(P,Q)$ optimal φ de G tel que l'ensemble $\varphi^{-1}(\ell)$ soit R-maximal.*

Démonstration. Soit φ un étiquetage L_Q^P optimal de $G = (V, R, B)$. Si $\varphi^{-1}(\ell)$ est R-maximal, alors φ est l'étiquetage optimal recherché. Sinon, c'est qu'il existe au moins un sommet v tel que :

- $\varphi(v) \neq h$;
- pour tout $w \in N(v)$, on a $\{v, w\} \in R$ et $\varphi(w) \neq h$.

Notons que dans un tel cas, il ne peut y avoir une arête entre v et un sommet u tel que $\varphi(u) = h$ (car $N_R(v) = N(v)$ et que pour tout $w \in N(v)$, on a $\varphi(w) \neq h$). On peut donc changer l'étiquette de v à h pour obtenir un nouvel étiquetage L_Q^P optimal de G. En appliquant ce ré-étiquetage récursivement, nous obtenons finalement l'étiquetage annoncé au lemme. □

Nous pouvons facilement observer que si le graphe n'est pas connexe, alors il est possible d'étiqueter indépendamment chacune des composantes connexes. Nous pouvons donc supposer que le graphe G est connexe. En particulier, quels que soient le graphe G et les ensembles P et Q, nous avons la relation :

$$\Lambda_Q^P(G) = \max\{\Lambda_Q^P(C) : C \text{ est une composante connexe de } G\}. \tag{2.3}$$

Notre algorithme partitionne les sommets en (tous les) triplets possibles (X, Y, Z) qui forment une partition correcte du graphe G. Les graphes $G[X]$ et $G[Z]$ sont ensuite étiquetés récursivement. L'algorithme considère les cas où $X = \varnothing$ ou $Z = \varnothing$ de façon séparée, à cause des restrictions posées par les ensembles P et Q. Finalement, l'étiquetage du graphe G est construit à partir des étiquetages calculés lors des appels récursifs. Les ensembles d'étiquettes utilisées par $G[X]$ et $G[Z]$ sont séparés les uns des autres par l'étiquette utilisée par l'ensemble stable R-maximal Y. En itérant sur tous les triplets (X, Y, Z) possibles, l'algorithme calcule l'entier $k = \Lambda_Q^P(G)$ minimum pour lequel un étiquetage L_Q^P de largeur k existe. Le pseudo-code de notre algorithme est présenté à l'algorithme 4.

La correction de l'algorithme **Calcule-Lambda** est assurée par les deux lemmes suivants.

Lemme 2.20. *Pour un rb-graphe $G = (V, \varnothing, \varnothing)$ et n'importe quels ensembles P, Q, on a $\Lambda_Q^P(G) \leq 3$.*

Démonstration. En étiquetant tous les sommets par l'étiquette 2, on obtient clairement un étiquetage satisfaisant. □

Lemme 2.21. *Pour tout graphe G et tout ensembles P, Q, un appel à **Calcule-Lambda**(G, P, Q) retourne la valeur $\Lambda_Q^P(G)$.*

Démonstration. La preuve est par induction sur le nombre de sommets du rb-graphe $G = (V, R, B)$.

Si $V(G) = \varnothing$, par définition de $\Lambda_Q^P(\varnothing, \varnothing, \varnothing)$, la valeur correcte est donnée à la ligne 1. Si $\Lambda_Q^P(G) \leq 3$, la valeur est déterminée par une recherche exhaustive à la ligne 3. Notons que si $|V(G)| \leq 1$, alors $P = Q = \varnothing$ et $\Lambda_Q^P(G) \leq 3$ par le lemme 2.20.

Supposons maintenant que l'énoncé du lemme soit vrai pour tout graphe G' et tout couple d'ensembles P', Q' tels que $|V(G')| < n$ et $n > 1$. Soit G un graphe à n sommets et deux ensembles P, Q. Supposons que $\Lambda_Q^P(G) > 3$ (puisque sinon G serait étiqueté par la recherche exhaustive de la ligne 3). Soit k la valeur retournée par un appel à **Calcule-Lambda**(G, P, Q). Pour montrer que $k = \Lambda_Q^P(G)$, il suffit de montrer que $k[C] = \Lambda_Q^P(C)$ (où $k[C]$ est tel que défini dans l'algorithme) pour toute composante connexe C. En ayant montré cela, nous avons que $k = \max\{k[C] : C \text{ est une comp. connexe de } G\} = \max\{\Lambda_Q^P[C] : C \text{ est une comp. connexe de } G\} = \Lambda_Q^P(G)$.

Algorithme 4 : Calcule-Lambda(G, P, Q)

Entrée : *Un rb-graphe $G = (V, R, B)$ et deux ensembles P, Q.*
Sortie : *La valeur de $\Lambda_Q^P(G)$.*

1 **si** $V(G) = \varnothing$ **alors retourner** 0
2 **pour** $k \leftarrow 1$ **à** 3 **faire**
3 **si** *il existe un étiquetage L_Q^P de largeur k de G* **alors retourner** k
4 **pour chaque** *composante connexe C de G* **faire**
5 $k[C] \leftarrow +\infty$
6 **pour chaque** *ensemble stable $Y \subseteq V(C)$ tel que $|Y| \geq |V(C)|/2$* **faire**
7 $k_1 \leftarrow$ **Calcule-Lambda(**$C - Y, N_B(Y), Q$**)**
8 $k_2 \leftarrow$ **Calcule-Lambda(**$C - Y, P, N_B(Y)$**)**
9 **si** $Y \cap P \neq \varnothing$ **alors** $k_1 \leftarrow k_1 + 1$
10 **si** $Y \cap Q \neq \varnothing$ **alors** $k_2 \leftarrow k_2 + 1$
11 $k[C] \leftarrow \min(k[C], k_1 + 1, k_2 + 1)$
12 **pour chaque** *partition correcte (X, Y, Z) de C* **faire**
13 $k_X \leftarrow$ **Calcule-Lambda(**$C[X], N_B(Y), Q$**)**
14 $k_Z \leftarrow$ **Calcule-Lambda(**$C[Z], P, N_B(Y)$**)**
15 $k[C] \leftarrow \min(k[C], k_X + 1 + k_Z)$
16 **retourner** $\Lambda_Q^P(G) = \max\{k[C] : C$ est une composante connexe de $G\}$.

Soit C une composante connexe de G. Nous commençons par montrer que $k[C] \geq \Lambda_Q^P(C)$, c'est-à-dire qu'il existe un étiquetage L_Q^P de largeur $k[C]$ pour C.

Supposons que $k[C]$ ait été défini par la ligne 11 de l'algorithme. Considérons l'ensemble stable Y et l'itération de la boucle aux lignes 6–11 à laquelle k a été défini. Soient k_1' = **Calcule-Lambda**$(C - Y, N_B(Y), Q)$ et k_2' = **Calcule-Lambda**$(C - Y, P, N_B(Y))$. Par hypothèse d'induction, il existe un étiquetage $L_Q^{N_B(Y)}$ de $C - Y$ de largeur k_1', noté φ', et un étiquetage $L_{N_B(Y)}^P$ de $C - Y$ de largeur k_2', noté φ''. Notons que $k[C] \in \{k_1' + 1, k_1' + 2, k_2' + 1, k_2' + 2\}$ et qu'au moins l'un des cas suivants est satisfait :

Cas 1 : $k[C] = k_1' + 1$ et $Y \cap P = \varnothing$. Dans ce cas, nous pouvons étendre φ' de la façon suivante

$$\varphi(v) = \begin{cases} \varphi'(v) & \text{si } v \in V(C) \setminus Y \\ k_1' + 1 & \text{si } v \in Y \end{cases}$$

et obtenir φ, un étiquetage L_Q^P de C de largeur $k[C]$.

Cas 2 : $k[C] = k_1' + 2$ et $Y \cap P \neq \varnothing$. Dans ce cas, nous pouvons étendre φ' de la façon suivante

$$\varphi(v) = \begin{cases} \varphi'(v) & \text{si } v \in V(C) \setminus Y \\ k_1' + 1 & \text{si } v \in Y \end{cases}$$

et obtenir φ, un étiquetage L_Q^P de C de largeur $k[C]$ (notons qu'à cause des restrictions sur P, l'étiquette $k_1' + 2$ est comptée comme étant utilisée, même si aucun sommet ne porte cette étiquette).

Cas 3 : $k[C] = k'_2 + 1$ et $Y \cap Q = \varnothing$. Dans ce cas, nous pouvons étendre φ'' de la façon suivante

$$\varphi(v) = \begin{cases} 1 & \text{si } v \in Y \\ \varphi''(v) + 1 & \text{si } v \in V(C) \setminus Y \end{cases}$$

et obtenir φ, un étiquetage L^P_Q de C de largeur $k[C]$.

Cas 4 : $k[C] = k'_2 + 2$ et $Y \cap Q \neq \varnothing$. Dans ce cas, nous pouvons étendre φ'' de la façon suivante

$$\varphi(v) = \begin{cases} 2 & \text{si } v \in Y \\ \varphi''(v) + 2 & \text{si } v \in V(C) \setminus Y \end{cases}$$

et obtenir φ, un étiquetage L^P_Q de C de largeur $k[C]$.

Supposons maintenant $k[C]$ ait été défini par la ligne 15. Considérons la partition correcte (X, Y, Z) et l'itération de la boucle aux lignes 12–15 à laquelle k a été défini. Soient k_X et k_Z tels que définis aux lignes 13 et 14 de cette itération. Ainsi, $k[C] = k_X + 1 + k_Z$. Par l'hypothèse d'induction, il existe un étiquetage $L^{N_B(Y)}_Q$ de $C[X]$ de largeur k_X, noté φ_X, et un étiquetage $L^P_{N_B(Y)}$ de $C[Z]$ de largeur k_Z, noté φ_Z. Nous pouvons définir un étiquetage L^P_Q de C de largeur $k[C]$ ainsi :

$$\varphi(v) = \begin{cases} \varphi_X(v) & \text{si } v \in X \\ k_X + 1 & \text{si } v \in Y \\ k_X + 1 + \varphi_Z(v) & \text{si } v \in Z. \end{cases}$$

Notons que les ensembles X, Y, Z sont non vides puisque (X, Y, Z) est une partition correcte de C. Donc $k_X, k_Z \geq 1$ et $\varphi^{-1}(1) \cap Q = \varphi^{-1}(k_X + 1 + k_Z) \cap P = \varnothing$.

Montrons maintenant que $k[C] \leq \Lambda^P_Q(C)$. Soit φ un étiquetage L^P_Q de largeur minimum de C. Rappelons que $\Lambda^P_Q(C) > 3$. L'un des cas suivants est satisfait :

Cas 1 : $\varphi^{-1}(1) \geq |V(C)|/2$. Considérons l'itération de la boucle aux lignes 6–11 pour $Y = \varphi^{-1}(1)$. Par l'hypothèse d'induction, l'algorithme **Calcule-Lambda** définit $k_2 = \Lambda^P_{N_B(Y)}(C - Y)$. Notons que $\Lambda^P_{N_B(Y)}(C - Y) = \Lambda^P_Q(C) - 1$ et $\varphi^{-1}(1) \cap Q = \varnothing$. Ainsi, la condition de la ligne 10 n'est pas vérifiée et k_2 est égal à $\Lambda^P_Q(C) - 1$. Par l'affectation de la ligne 11, on obtient $k[C] \leq k_2 + 1 \leq \Lambda^P_Q(C)$.

Cas 2 : $\varphi^{-1}(1) = \varnothing$ et $\varphi^{-1}(2) \geq |V(C)|/2$. Considérons l'itération de la boucle aux lignes 6–11 pour $Y = \varphi^{-1}(2)$. Par l'hypothèse d'induction, l'algorithme **Calcule-Lambda** définit $k_2 = \Lambda^P_{N_B(Y)}(C - Y)$. Remarquons que $\varphi^{-1}(1) \cap Q \neq \varnothing$ puisque sinon on aurait diminué de 1 l'étiquette de chaque sommet, et obtenu un étiquetage L^P_Q de largeur $\Lambda^P_Q(C) - 1$ pour C. Ainsi, $\Lambda^P_{N_B(Y)}(C - Y) = \Lambda^P_Q(C) - 2$ et l'algorithme **Calcule-Lambda** définit k_2 à $\Lambda^P_Q(C) - 2 + 1$ à ligne 10. Par l'affectation de la ligne 11, $k[C] \leq k_2 + 1 \leq \Lambda^P_Q(C) - 2 + 1 + 1 = \Lambda^P_Q(C)$.

Cas 3 : $\varphi^{-1}(\Lambda^P_Q(C)) \geq |V(C)|/2$ est symétrique au cas 1.

Cas 4 : $\varphi^{-1}(\Lambda^P_Q(C)) = \varnothing$ et $\varphi^{-1}(\Lambda^P_Q(C) - 1) \geq |V(C)|/2$ est symétrique au cas 2.

Notons que ni le cas $\varphi^{-1}(1) = \varphi^{-1}(2) = \varnothing$, ni le cas $\varphi^{-1}(\Lambda_Q^P(C)) = \varphi^{-1}(\Lambda_Q^P(C)-1) = \varnothing$ ne peut apparaître puisque l'étiquetage φ est optimal. Le dernier cas qu'il nous reste à considérer est donc :

Cas 5 : $0 < |\varphi^{-1}(1)| \leq |V(C)|/2$ ou $0 < |\varphi^{-1}(1) \cup \varphi^{-1}(2)| \leq |V(C)|/2$ et $0 < |\varphi^{-1}(\Lambda_Q^P(C))| \leq |V(C)|/2$ ou $0 < |\varphi^{-1}(\Lambda_Q^P(C)) \cup \varphi^{-1}(\Lambda_Q^P(C)-1)| \leq |V(C)|/2$.

Soit ℓ tel que défini au lemme 2.18, c'est-à-dire soit $\ell = \min\{j \in \{1,\dots,\Lambda_Q^P(C)\}$ tel que $|\bigcup_{i=1}^{j} \varphi^{-1}(i)| \geq |V(C)|/2\}$. Notons que $1 < \ell < \Lambda_Q^P(C)$, à cause des conditions du cas 5. Le triplet $(\bigcup_{i=1}^{\ell-1} \varphi^{-1}(i), \varphi^{-1}(\ell), \bigcup_{i=\ell+1}^{\Lambda_Q^P(C)} \varphi^{-1}(i))$ est une partition balancée de G.

Soit φ' un étiquetage L_Q^P optimal de C, construit de φ comme dans la preuve du lemme 2.19. Soit $X = \bigcup_{i=1}^{\ell-1} \varphi'^{-1}(i)$, $Y = \varphi'^{-1}(\ell)$ et $Z = \bigcup_{i=\ell+1}^{\Lambda_Q^P(C)} \varphi'^{-1}(i)$.

Notons que

- Y est R-maximal ;
- $\varnothing \neq \varphi^{-1}(\ell) \subseteq Y$ et donc $Y \neq \varnothing$;
- $X \subseteq \bigcup_{i=1}^{\ell-1} \varphi^{-1}(i)$ et donc $|X| \leq |V(C)|/2$;
- $Z \subseteq \bigcup_{i=h+1}^{\Lambda_Q^P(C)} \varphi^{-1}(i)$ et donc $|Z| \leq |V(C)|/2$.

De plus $X \neq \varnothing$ et $Z \neq \varnothing$, car sinon les cas 2 ou 4 seraient apparus. Donc (X,Y,Z) est une partition correcte de C et elle est considérée dans l'itération de la boucle des lignes 12–15. Dans l'itération qui considère (X,Y,Z), l'algorithme définit $k_X = \Lambda_Q^{N_B(Y)}(C(X)) = \ell - 1$ à la ligne 13 et $k_Z = \Lambda_{N_B(Y)}^P(C(Z)) = \Lambda_Q^P(C) - \ell$ à la ligne 14 (par l'hypothèse d'induction). Ainsi, $k[C] \leq k_X + 1 + k_Z = \ell - 1 + 1 + \Lambda_Q^P(C) - \ell = \Lambda_Q^P(C)$. Cela conclut la preuve.

□

Analyse du temps d'exécution

Une estimation rapide donne un temps d'exécution de $\mathcal{O}((9+\epsilon)^n)$. En effet, on observe tout d'abord que vérifier si le graphe est vide (ligne 1 de l'algorithme **Calcule-Lambda**) peut se faire en temps constant. Ensuite, à la ligne 3, l'algorithme vérifie de façon exhaustive s'il existe un étiquetage L_Q^P de G de largeur au plus $k \leq 3$. Clairement, il existe au plus 3^n fonctions $\varphi: V(G) \to \{1,2,3\}$, et donc cette étape peut être effectuée en temps $n^{\mathcal{O}(1)} \cdot 3^n$. Ensuite, le nombre d'ensembles stables Y considérés à la ligne 6 est au plus 2^n et le nombre de partitions correctes (X,Y,Z) considérées à la ligne 12 est au plus 3^n. Ainsi, si on note $T(n)$ le temps d'exécution de l'algorithme **Calcule-Lambda** sur un graphe G à n sommets, le temps d'exécution total est donné par la récurrence

$$T(n) \leq 3^n \, n^{\mathcal{O}(1)} \, T(n/2).$$

La solution de cette récurrence est bornée par $\mathcal{O}(9^n \, n^{\mathcal{O}(1)\log n}) = \mathcal{O}(9^n \, 2^{\mathcal{O}(1)\log^2 n})$, qui est encore borné par $\mathcal{O}((9+\epsilon)^n)$, pour tout $\epsilon > 0$.

En utilisant les bornes que nous démontrerons à la section 2.3.3, nous pouvons améliorer notre analyse du temps d'exécution, comme indiqué dans le théorème suivant :

Théorème 2.22 (Junosza-Szaniawski *et al.* (2013b), lemme 7). *L'algorithme* ***Calcule-Lambda*** *calcule $\Lambda_Q^P(G)$ d'un rb-graphe en temps $\mathcal{O}((8+\epsilon)^n)$ et en espace polynomial, où n est le nombre de sommets de G et $\epsilon > 0$ est une constante arbitrairement petite.*

Démonstration. Les observations précédentes concernant l'analyse restent valables. Néanmoins, par le théorème 2.31 de la page 94 et par l'observation 1 de la page 67, il est possible de mieux estimer le nombre maximum de partitions correctes considérées par l'algorithme **Calcule-Lambda** : il y en a au plus $\sqrt{8}^n$. On obtient ainsi la récurrence suivante sur le temps d'exécution :

$$T(n) \leq \sqrt{8}^n \, n^{\mathcal{O}(1)} \, T(n/2). \tag{2.4}$$

La solution de cette récurrence est bornée par $\mathcal{O}(8^n n^{\mathcal{O}(1)\log n}) = \mathcal{O}(8^n 2^{\mathcal{O}(1)\log^2 n})$, qui est borné par $\mathcal{O}((8+\epsilon)^n)$, pour tout $\epsilon > 0$. □

Revenons à notre problème original qui consiste, étant donné un graphe G, à calculer $\lambda_{2,1}(G)$. Si on se restreint aux R-fermetures de graphes, nous obtenons des rb-graphes particuliers. Nous pouvons alors montrer le théorème 2.16 cité à la page 65 :

Théorème 2.16 (Junosza-Szaniawski *et al.* (2013b), théorème 1). *La largeur minimum d'un étiquetage $L(2,1)$ d'un graphe à n sommets peut être calculée en temps $\mathcal{O}(7.4920^n)$ et espace polynomial.*

Démonstration du théorème 2.16. Nous supposons que le graphe G pour lequel on cherche un étiquetage $L(2,1)$ soit connexe (sinon, on peut étiqueter chacune de ses composantes connexes séparément). Supposons que le graphe initialement donné à l'algorithme **Calcule-Lambda** soit une R-fermeture de G (une R-fermeture peut être calculée en temps polynomial).

On remarque que (X, Y) est une paire propre R-maximale de la R-fermeture de G si et seulement si Y est un 2-stable de G. En effet par définition de (X, Y), l'ensemble Y est un stable du rb-graphe obtenu de la R-fermeture de G, tel que pour tout sommet v pour lequel $N_R(v) = N(v)$, on ait $N[v] \cap Y \neq \emptyset$. Or dans la R-fermeture de G, chaque sommet incident à une arête rouge (*i.e.* de R) est aussi incident à au moins une arête noire (*i.e.* de B). Donc aucun sommet v ne satisfait $N_R(v) = N(v)$, et l'ensemble Y est un stable de G^2, autrement dit Y est un 2-stable de G.

Par conséquent, nous pouvons utiliser le théorème 2.34 établi à la page 97, pour obtenir la relation suivante sur le temps d'exécution :

$$T'(n) \leq 2.6488^n \, n^{\mathcal{O}(1)} \, T(n/2) \tag{2.5}$$

où T est donné par l'inégalité (2.4). (Notons que, dans l'expression de T', nous utilisons $T(n/2)$ et non $T'(n/2)$ car lors des appels récursifs, nous ne pouvons garantir que le graphe donné en paramètre soit encore une R-fermeture d'un certain graphe.) On obtient ainsi la solution $T(n) = \mathcal{O}(7.4920^n)$. □

Remarque 5. *Notons que l'algorithme* ***Calcule-Lambda*** *peut être généralisé pour résoudre le problème* AFFECTATION DE FRÉQUENCES *pour un entier ℓ (voir la page 37). L'entrée est un graphe G et une fonction $w : E \to \{1, 2, \ldots, \ell\}$. L'objectif est de calculer un étiquetage $\psi : V(G) \to \{0, 1, \ldots, k\}$ pour lequel k est minimum et tel que pour toute arête $\{u, v\} \in E$, on ait $|\psi(u) - \psi(v)| \geq w(\{u, v\})$.*

*Pour adapter l'algorithme **Calcule-Lambda**, nous devons considérer la partition des sommets en $\ell+1$ ensembles $X, Y_1, Y_2, \ldots, Y_{\ell-1}, Z$ telle que $|X| \leq n/2$, $|Z| \leq n/2$, et $Y_1, \ldots, Y_{\ell-1}$ soient des stables. L'algorithme est ensuite appliqué récursivement sur $G[X]$ et $G[Z]$. Cette approche en espace polynomial donnerait un temps d'exécution en $\mathcal{O}((\ell+1)^{2n})$.*

Étiquetage petits

Lorsque la largeur k de l'étiquetage est « petite », il est possible d'améliorer la complexité de l'algorithme **Calcule-Lambda**. À nouveau, nous cherchons à résoudre le problème ÉTIQUETAGE L_Q^P, pour k fixé, sur des rb-graphes. Nous notons ÉTIQUETAGE $k-L_Q^P$ ce problème et dénotons pas $T'_k(n)$ le temps nécessaire à sa résolution pour un rb-graphe à n sommets. Observons que ÉTIQUETAGE $k-L_Q^P$ est un cas particulier d'un problème de satisfaction de contraintes : le problème $(d,2)$-CSP que nous pouvons définir comme ci-après.

$(d,2)$-CSP

Entrée.
- Un ensemble V de sommets ;
- un ensemble C_v de couleurs possibles pour chaque sommet v et tel que $|C_v| \leq d$;
- un ensemble $\mathcal{C}$ de contraintes de la forme $\{(u,c_u),(v,c_v)\}$, où $u,v \in V$ et $c_u \in C_u$, $c_v \in C_v$.

Question. Existe-t-il un étiquetage φ des sommets tel que :

1. $\varphi(v) \in C_v$ pour tout $v \in V$,
2. $\{(u,\varphi(u)),(v,\varphi(v))\} \notin \mathcal{C}\}$ pour tout $u,v \in V$?

Le théorème suivant résume les complexités des algorithmes exacts pour le problème $(d,2)$-CSP.

Théorème 2.23 (Beigel et Eppstein (2005), Angelsmark (2005))**.** *Le problème $(d,2)$-CSP peut être résolu en espace polynomial et en temps*

1. $\mathcal{O}(1.3645^n)$ *pour* $d=3$;
2. $\mathcal{O}((0.4518 \cdot d + \epsilon)^n)$ *(où ϵ est une constante positive arbitraire) pour tout $d \geq 4$.*

Il nous reste à expliquer de quelle façon une instance de ÉTIQUETAGE $k-L_Q^P$ peut être modélisée par une instance de $(d,2)$-CSP. À chaque sommet v, nous associons l'ensemble suivant de couleurs :

$$C_v = \begin{cases} \{2,3,\ldots,k-1\} & \text{si } v \in P \cap Q, \\ \{2,3,\ldots,k\} & \text{si } v \in P \setminus Q, \\ \{1,2,\ldots,k-1\} & \text{si } v \in Q \setminus P, \\ \{1,2,\ldots,k\} & \text{sinon.} \end{cases}$$

Nous définissons l'ensemble des contraintes par :

$$\begin{aligned}\mathcal{C} \;=\; & \{\{(u,c),(v,c)\} \text{ pour } uv \in E(G), c \in C_u \cap C_v\} \cup \\ & \{\{(u,c),(v,c+1)\} \text{ pour } uv \in B(G), c \in C_u, c+1 \in C_v\} \cup \\ & \{\{(u,c),(v,c-1)\} \text{ pour } uv \in B(G), c \in C_u, c-1 \in C_v\}.\end{aligned}$$

Pour $k \leq 8$, nous pouvons utiliser directement cette réduction à $(d,2)$-CSP pour résoudre ÉTIQUETAGE $k - L_Q^P$ dans le temps indiqué au théorème 2.23.

Pour $k > 8$, nous procédons comme l'algorithme **Calcule-Lambda**, mais avec une approche un peu différente pour partitionner l'ensemble des sommets en trois sous-ensembles. Par le lemme 2.19, nous pouvons supposer que l'ensemble Y (étiqueté par l'étiquette $\lceil k/2 \rceil$) est R-maximal. Nous partitionnons ensuite les sommets restants en deux ensembles X et Z, qui sont étiquetés respectivement par les étiquettes $1, 2, .., \lceil k/2 \rceil - 1$ et $\lceil k/2 \rceil + 1, .., k$. L'étiquetage de X (respectivement, de Z) est obtenu par l'appel récursif sur l'instance de $L_Q^{N(Y)}$ (respectivement, de $L_{N(Y)}^P$) sur le graphe $G[X]$ (respectivement, $G[Z]$). Notons r_ℓ le nombre maximum d'ensembles stables R-maximaux à ℓ éléments dans un rb-graphes à n sommets. Nous obtenons la formule suivante qui établit la complexité :

$$T'_k(n) \leq \sum_{\ell=0}^{n} r_\ell(n) \sum_{m=0}^{n-\ell} \binom{n-\ell}{m} \left(T'_{\lceil k/2 \rceil - 1}(m) + T'_{\lfloor k/2 \rfloor}(n-\ell-m) \right).$$

Soit α_k un nombre tel que $T'_{\lfloor k/2 \rfloor}(n) \leq \alpha_k^n$ et posons $\beta_k = \alpha_k + 1$. Nous obtenons la formule suivante :

$$\begin{aligned} T'_k(n) &\leq \sum_{\ell=0}^{n} r_\ell(n) \sum_{m=0}^{n-\ell} \binom{n-\ell}{m} \left(T'_{\lceil k/2 \rceil - 1}(m) + T'_{\lfloor k/2 \rfloor}(n-\ell-m) \right) \\ &\leq 2 \sum_{\ell=0}^{n} r_\ell(n) \sum_{m=0}^{n-\ell} \binom{n-\ell}{m} T'_{\lfloor k/2 \rfloor}(n-\ell-m) \\ &\leq 2 \sum_{\ell=0}^{n} r_\ell(n) \sum_{m=0}^{n-\ell} \binom{n-\ell}{m} \alpha_k^{n-\ell-m} \\ &\leq 2 \sum_{\ell=0}^{n} r_\ell(n) \beta_k^{n-\ell}. \end{aligned}$$

Au théorème 2.30 de la page 94, nous démontrons une borne sur $r_\ell(n)$. En particulier, la preuve de ce théorème établit la relation :

$$r_\ell(n) \leq \sum_{\ell'=0}^{\ell} \binom{\lceil n'/2 \rceil}{\ell'} 2^{\ell'} \left(\frac{81}{64} \right)^{\ell-\ell'} \left(\frac{4}{3} \right)^{n-n'}.$$

La borne sur $T'_k(n)$ peut donc encore s'écrire :

$$\begin{aligned}
T'_k(n) &\leq 2\sum_{\ell=0}^{n} r_\ell(n)\beta_k^{n-\ell} \\
&\leq 2\sum_{\ell=0}^{n}\sum_{\ell'=0}^{\ell}\binom{n'/2}{\ell'}2^{\ell'}\left(\frac{81}{64}\right)^{\ell-\ell'}\left(\frac{4}{3}\right)^{n-n'}\beta_k^{n-\ell} \\
&= 2\left(\frac{4}{3}\right)^{n-n'}\beta_k^{n}\sum_{\ell=0}^{n}\sum_{\ell'=0}^{\ell}\binom{n'/2}{\ell'}2^{\ell'}\left(\frac{81}{64}\right)^{\ell-\ell'}\beta_k^{-\ell} \\
&= 2\left(\frac{4}{3}\right)^{n-n'}\beta_k^{n}\sum_{\ell'=0}^{n}\sum_{\ell=\ell'}^{n}\binom{n'/2}{\ell'}2^{\ell'}\left(\frac{81}{64}\right)^{\ell-\ell'}\beta_k^{-\ell} \\
&= 2\left(\frac{4}{3}\right)^{n-n'}\beta_k^{n}\sum_{\ell'=0}^{n}\binom{n'/2}{\ell'}\left(\frac{128}{81}\right)^{\ell'}\sum_{\ell=\ell'}^{n}\Big(\underbrace{\frac{81}{64\beta_k}}_{<1}\Big)^{\ell} \\
&\leq 2\left(\frac{4}{3}\right)^{n-n'}\beta_k^{n}\sum_{\ell'=0}^{n}\binom{n'/2}{\ell'}\left(\frac{128}{81}\right)^{\ell'}(n-\ell')\left(\frac{81}{64\beta_k}\right)^{\ell'} \\
&\leq 2n\left(\frac{4}{3}\right)^{n-n'}\beta_k^{n}\sum_{\ell'=0}^{n}\binom{n'/2}{\ell'}\left(\frac{2}{\beta_k}\right)^{\ell'} \\
&= 2n\left(\frac{4}{3}\right)^{n-n'}\beta_k^{n}\sum_{\ell'=0}^{n'/2}\binom{n'/2}{\ell'}\left(\frac{2}{\beta_k}\right)^{\ell'} \\
&= 2n\left(\frac{4}{3}\right)^{n-n'}\beta_k^{n}\left(1+\frac{2}{\beta_k}\right)^{n'/2} \\
&= 2n\left(\frac{4\beta_k}{3}\right)^{n}\left(\frac{3}{4}\cdot\sqrt{\frac{\beta_k+2}{\beta_k}}\right)^{n'}.
\end{aligned}$$

Dans le cas où $k \geq 9$, la valeur de $\frac{3}{4}\cdot\sqrt{\frac{\beta_k+2}{\beta_k}}$ est plus petite que 1. On obtient donc la borne suivante pour $k \geq 9$:

$$T'_k(n) \leq 2n\left(\frac{4\beta_k}{3}\right)^{n}.$$

La table 2.4 donne des bornes supérieures c_k^n sur $T'_k(n)$ pour de petites valeurs de k (et n suffisamment grand).

k	3	4	5	6	7	8	9	10	...	15	16
c_k	1.3645	1.8072	2.2590	2.7108	3.1626	3.6144	3.7430	4.3454	...	5.5502	6.1526

TABLE 2.4 – Base c_k de l'exposant dans la borne pour $T'_k(n)$. Pour $k \leq 8$, la valeur de c_k est directement obtenue du théorème 2.23. Pour $k \geq 9$, la valeur de c_k est obtenue en utilisant l'approche *diviser-pour-régner* telle que nous venons de la décrire. Par exemple, pour $k = 9$, nous avons $c_9 = \frac{4\beta_9}{3}$ avec $\beta_9 = c_{\lfloor 9/2 \rfloor}$.

Notons $T_k(n)$ le temps d'exécution nécessaire à la résolution du problème ÉTIQUETAGE $L(2,1)$, de largeur k sur un graphe à n sommets. Tout comme au théorème 2.16, nous pouvons supposer que le rb-graphe initial est une R-fermeture d'un graphe connexe. Par conséquent, dans la première étape, nous pouvons choisir l'ensemble Y comme un 2-stable du graphe. Dans la suite, nous

utilisons la borne $w_\ell(n)$ de Junosza-Szaniawski et Rzazewski (2010) sur le nombre maximum de 2-stables à ℓ éléments ; cette borne est rappelée au théorème 2.28 de la page 93.

Rappelons que pour le problème ÉTIQUETAGE $L(2,1)$, nous commençons l'étiquetage avec 0 ; le nombre d'étiquettes disponibles est donc $k+1$. On obtient ainsi, la borne suivante sur la complexité de notre approche :

$$T_k(n) \leq \sum_{\ell=0}^{n} w_\ell(n) \sum_{m=0}^{n-\ell} \binom{n-\ell}{m} \left(T'_{\lceil k/2 \rceil}(m) + T'_{\lfloor k/2 \rfloor}(n-\ell-m) \right)$$

De cela, nous déduisons le majorant suivant (pour n suffisamment grand), où c_k est donné à la table 2.4 et satisfait $T'_k(n) \leq c_k^n$, et $d_k = 1 + c_k$:

$$\begin{aligned} T_k(n) &\leq \sum_{\ell=0}^{n} w_\ell(n) \sum_{m=0}^{n-\ell} \binom{n-\ell}{m} \left(T'_{\lceil k/2 \rceil}(m) + T'_{\lfloor k/2 \rfloor}(n-\ell-m) \right) \\ &\leq 2 \sum_{\ell=0}^{n} w_\ell(n) \sum_{m=0}^{n-\ell} \binom{n-\ell}{m} c_{\lceil k/2 \rceil}^m \\ &= 2 \sum_{\ell=0}^{n} w_\ell(n) d_{\lceil k/2 \rceil}^{n-\ell}. \end{aligned}$$

La table 2.5 présente les bornes sur $T_k(n)$ pour quelques valeurs de k. La seconde colonne donne le temps d'exécution de l'algorithme de *branchement* de la section 2.3.1.1. Au lieu d'utiliser la borne en $O^*((k-2.5)^n)$ de cet algorithme, nous utilisons les récurrences un peu plus précises établies dans la section 2.3.1.1 (typiquement, $\sqrt{(k-2)(k-3)}$ est plus précis que $(k-2.5)$).

k	branchement de la section 2.3.1.1	borne sur $T_k(n)$
4	1.3006	
5	2.4495	
6	3.4642	3.1219
7	4.4722	3.5894
8	5.4773	3.5894
9	6.4808	4.0615
10	7.4834	4.0615
11	8.4853	4.5300
	...	
30	27.4955	7.4316
31	28.4957	8.0420
32	29.4958	8.0420

TABLE 2.5 – Bornes sur $T_k(n)$ pour diverses valeurs de k. La troisième colonne donne la base de l'exposant pour la borne sur $T_k(n)$. Par exemple, pour $k=9$ on utilise $d_{\lceil 9/2 \rceil} = d_5 = 1 + c_5 = 1 + 2.2590$. On calcule ensuite le maximum de $w_\ell(n) \cdot d_5^{n-\ell}$; pour cela, on pose $\ell = \gamma \cdot n$, avec $\gamma \in [0,1]$, et par l'approximation de Stirling, on a $w_\ell(n) \approx \left(\frac{(1-\gamma)^{(1-\gamma)}}{\gamma^\gamma \cdot (1-2\gamma)^{(1-2\gamma)}} \right)^n$.

2.3.2 Espace exponentiel

Dans les deux prochaines sections 2.3.2.1 et 2.3.2.2, nous présentons deux algorithmes exponentiels sensiblement plus rapides que celui de la section 2.3.1, au prix de l'utilisation d'un espace exponentiel. Ces deux algorithmes sont basés sur de la *programmation dynamique* et l'analyse de leur

temps d'exécution demandera l'étude de bornes établies à la section 2.3.3. Le premier algorithme, relativement simple, résout le problème ÉTIQUETAGE $L(2,1)$ en temps $\mathcal{O}(3.8730^n)$. L'analyse de cet algorithme est améliorée par Junosza-Szaniawski et Rzazewski (2010) à $\mathcal{O}(3.5616^n)$. Pour cela, les auteurs établissent essentiellement une meilleure borne combinatoire sur des objets nécessaires à l'algorithme de la section 2.3.2.1 et apportent une légère modification à l'algorithme. Junosza-Szaniawski et Rzazewski (2011) établissent un temps d'exécution de $\mathcal{O}(3.2361^n)$ en considérant des objets combinatoires plus pertinents pour l'algorithme. Ils montrent aussi que $\Omega(3.0739^n)$ est une borne inférieure sur la complexité de l'algorithme au pire des cas. A la section 2.3.2.2, nous présentons un second algorithme, différent du précédent. Il est lui aussi basé sur de la *programmation dynamique*. Grâce à l'étude des *paires propres* (que nous définirons, et dont des bornes combinatoires sont établies à la section 2.3.3.3), nous montrons que son temps d'exécution au pire des cas est borné par $\mathcal{O}(2.6488^n)$, ce qui nous permet de franchir la barre de « $\mathcal{O}^*(3^n)$ ». Il s'agit aujourd'hui du meilleur algorithme connu pour déterminer un étiquetage $L(2,1)$ de largeur minimale dans un graphe à n sommets.

2.3.2.1 Un schéma simple de programmation dynamique

Nous montrons à présent que le calcul d'un étiquetage $L(2,1)$ de largeur minimum peut s'effectuer plus rapidement qu'en temps $\mathcal{O}^*(4^n)$ (qui est par exemple le temps d'exécution de l'algorithme de Král (2006) appliqué au problème ÉTIQUETAGE $L(2,1)$).

Théorème 2.24 (Havet *et al.* (2011), théorème 9)**.** *Le problème* ÉTIQUETAGE $L(2,1)$ *de largeur minimale peut être résolu en temps* $\mathcal{O}^*(15^{n/2}) = \mathcal{O}(3.8730^n)$ *en utilisant un espace exponentiel.*

Démonstration. Soit $G = (V, E)$ un graphe. Pour tout entier $i \in \{0, 1, \ldots, 2n\}$ et tous sous-ensembles $X, Y \subseteq V$ tels que $X \cap Y = \varnothing$, nous définissons une variable booléenne $\texttt{Lab}[X, Y, i]$. Par un schéma de *programmation dynamique*, nous déterminons les valeurs de ces variables telles que :

> $\texttt{Lab}[X, Y, i]$ est *vraie* si et seulement s'il existe un étiquetage $L(2,1)$ partiel de largeur i pour X, noté L (*i.e.* $L : X \to \{0, 1, \ldots, i\}$), tel que chaque sommet de $N(Y) \cap X$ ait au plus $i - 1$ comme étiquette.

Clairement, $\lambda_{2,1}(G) = \min\{i \mid \texttt{Lab}[V(G), \varnothing, i] \text{ est vrai}\}$. Remarquons que $\lambda_{2,1}(G) \leq 2n$ puisque l'étiquetage des sommets par des entiers pairs distincts donne toujours un étiquetage $L(2,1)$ correct.

Une description complète est donnée par l'algorithme 5.

La validité de l'algorithme **Calcule-LambdaDP** est établie par les observations suivantes :

- Par définition des étiquetages $L(2,1)$, chaque ensemble de sommets ayant la même étiquette induit un 2-stable de G (c'est-à-dire un ensemble de sommets situés à distance au moins 2 les un des autres).

- Pour chaque paire d'ensembles disjoints X et Y, l'ensemble X admet un étiquetage partiel de largeur 0, tel que tous les sommets de $X \cap N(Y)$ aient pour étiquette au plus -1 si et seulement si $N(Y) \cap X = \varnothing$ et X induit un 2-stable de G. D'où l'étape d'initialisation de $\texttt{Lab}[X, Y, 0]$ dans l'algorithme.

Algorithme 5 : Calcule-LambdaDP(G)

Entrée : *Un graphe* $G = (V, E)$.
Sortie : *La valeur* $\lambda_{2,1}(G)$.

pour tous les $X, Y \subseteq V(G)$, $X \cap Y = \varnothing$, $i = 0, \ldots, 2n$ **faire**
  $\texttt{Lab}[X, Y, i] \leftarrow$ faux

pour tous les $X, Y \subseteq V(G)$, $X \cap Y = \varnothing$ **faire**
  si X *est un 2-stable de* G *et* $X \cap N(Y) = \varnothing$ **alors**
    $\texttt{Lab}[X, Y, 0] \leftarrow$ vrai

pour $i = 1, \ldots, 2n$ **faire**
  pour tous les $U, A, Y \subseteq V(G)$, $U \cap A = U \cap Y = A \cap Y = \varnothing$ **faire**
    si U *est un 2-stable de* G, $U \cap N(Y) = \varnothing$ *et* $\texttt{Lab}[A, U, i-1]$ **alors**
      $\texttt{Lab}[U \cup A, Y, i] \leftarrow$ vrai

pour $i = 0, \ldots, 2n$ **faire**
  si $\texttt{Lab}[V(G), \varnothing, i]$ **alors**
    retourner « $\lambda_{2,1}(G) = i$ » et s'arrêter

- Soient X et Y deux ensembles disjoints de sommets et soit $i > 0$. Supposons que f soit un étiquetage $L(2,1)$ de X de largeur i tel que $f(u) \leq i-1$ pour tout $u \in N(Y) \cap X$. Soit $U \subseteq X$ l'ensemble des sommets de X étiquetés par i et soit $A = X \setminus U$. Alors U doit être un 2-stable de G et $N(Y) \cap U = \varnothing$. De plus, chaque voisin étiqueté d'un sommet de U doit avoir une étiquette au plus égale à $i-2$. Donc l'étiquetage f restreint aux sommets de A satisfait la condition que $\texttt{Lab}[A, U, i-1]$ soit vraie. D'autre part, l'extension de l'étiquetage de largeur $i-1$ de A aux sommets de X en posant $f(u) = i$ pour tout $u \in U$ donne un étiquetage satisfaisant nos conditions, en supposant $N(Y) \cap U = \varnothing$. Cela justifie le calcul de $\texttt{Lab}[X, Y, i]$ par notre schéma de *programmation dynamique*.

Évaluons le temps d'exécution au pire des cas de **Calcule-LambdaDP**. Le calcul des valeurs $\texttt{Lab}[X, Y, i]$ est effectué en considérant des ensembles $X, Y \subseteq V$ de cardinalités croissantes et des valeurs de i croissantes. Un point crucial pour déterminer le temps d'exécution est l'évaluation du nombre de 2-stables dans un graphe connexe. Notons u_ℓ le nombre de 2-stables de taille ℓ dans un graphe à n sommets. Pour chaque 2-stable U de taille ℓ, l'algorithme considère toutes les paires A, Y d'ensembles disjoints de $V(G) \setminus U$. Puisque chaque sommet de $V(G) \setminus U$ a finalement trois possibilités (soit d'être dans A, soit d'être dans Y, soit de n'être dans aucun de ces deux ensembles), il y a $3^{n-\ell}$ possibilités. Donc le temps d'exécution est donné par

$$O^*\left(\sum_{\ell=0}^{n} u_\ell \, 3^{n-\ell}\right).$$

À la page 91, nous montrons le théorème 2.27 qui nous assure que $u_\ell \leq \binom{\lceil n/2 \rceil}{\ell} \cdot 2^\ell$. De plus, ces ensembles 2-stables peuvent effectivement être énumérés dans ce même temps car la preuve du théorème 2.27 est constructive. (Une autre façon de procéder serait par exemple de considérer ces 2-ensembles comme des ensembles stables de G^2 puis d'utiliser un algorithme à délai polynomial pour les énumérer (Johnson *et al.*, 1988)).

Un simple calcul nous permet de conclure cette preuve :

$$\begin{aligned}
\sum_{\ell=0}^{n} u_\ell \, 3^{n-\ell} &\leq \sum_{\ell=0}^{\lceil n/2 \rceil} \binom{\lceil n/2 \rceil}{\ell} \cdot 2^\ell \cdot 3^{n-\ell} \\
&= 3^{n/2} \sum_{\ell=0}^{\lceil n/2 \rceil} \binom{\lceil n/2 \rceil}{\ell} \cdot 2^\ell \cdot 3^{n/2-\ell} \\
&= 3^{n/2} \, 3^{n/2} \sum_{\ell=0}^{\lceil n/2 \rceil} \binom{\lceil n/2 \rceil}{\ell} \cdot \left(\frac{2}{3}\right)^\ell \\
&= 3^n \cdot \left(1 + \frac{2}{3}\right)^{\lceil n/2 \rceil} \\
&\leq \left(3 \cdot \sqrt{1 + \frac{2}{3}}\right)^n \cdot \sqrt{1 + \frac{2}{3}} \\
&= \frac{\sqrt{15}}{3} \cdot \sqrt{15}^n.
\end{aligned}$$

Ainsi, le temps d'exécution est borné par $O^*((\sqrt{15})^n) = \mathcal{O}(3.8730^n)$. □

Dans (Junosza-Szaniawski et Rzazewski, 2010) puis dans (Junosza-Szaniawski et Rzazewski, 2011), cet algorithme est successivement amélioré jusqu'à établir un temps d'exécution en $\mathcal{O}(3.2361^n)$. Ce nouveau résultat est obtenu à la fois par de meilleurs bornes combinatoires mais aussi par l'énumération d'objets plus pertinents. Dans (Junosza-Szaniawski et Rzazewski, 2011), les auteurs énumèrent des « triplets légaux » dans la boucle principale de l'algorithme, ce qui permet de considérer les ensembles U, A et Y de façon moins indépendante qu'à la présente section. À la section suivante, nous présentons un algorithme en $\mathcal{O}(2.6488^n)$.

2.3.2.2 Algorithme plus rapide que $O^*(3^n)$

Nous présentons dans cette section l'un des résultats majeurs du chapitre, à savoir un algorithme plus rapide que $O^*(3^n)$ pour le problème ÉTIQUETAGE $L(2,1)$:

Théorème 2.26 (Junosza-Szaniawski *et al.* (2013a), théorème 6)**.** *La largeur minimum d'un étiquetage $L(2,1)$ d'un graphe connexe peut être déterminée en temps $\mathcal{O}(2.6488^n)$ et espace exponentiel.*

L'une des idées clés de notre algorithme est l'emploi de *manipulations algébriques*, comme on peut les trouver dans les algorithmes rapides de multiplication de matrices, comme celui de Strassen (1969). Étant données deux matrices A et B de dimension $2^k \times 2^k$, il est possible de les diviser chacune en quatre blocs de taille identique. Le produit de matrices $A \cdot B$ peut être ensuite calculé facilement par huit multiplications de matrices de taille $2^{k-1} \times 2^{k-1}$. En répétant cette opération récursivement, nous obtenons un temps d'exécution en $\mathcal{O}(n^3)$, tout comme l'algorithme naïf. Il est possible d'améliorer ce temps d'exécution en utilisant seulement sept multiplications de matrices et obtenir un algorithme en $\mathcal{O}(n^{\log_2(7)+o(1)}) = \mathcal{O}(n^{2.8074})$. Le génie de cet algorithme réside dans des manipulations algébriques qui requièrent moins d'opérations arithmétiques. Nous redonnons quelques détails de cet algorithme dans la suite.

Néanmoins, pour résoudre le problème ÉTIQUETAGE $L(2,1)$, cette technique appliquée seule n'est pas suffisante. Nous utilisons en plus une autre astuce : une représentation *compacte* des étiquetages $L(2,1)$ partiels dans notre schéma de *programmation dynamique*. Comme nous allons

le voir, nous utiliserons en fait deux représentations des étiquetages partiels. Une telle idée a également été utilisée par van Rooij *et al.* (2009) (voir aussi van Rooij (2011)).

Multiplication de matrices par l'algorithme de Strassen

Faisons une petite parenthèse en rappelant l'idée de l'algorithme de Strassen (1969). Supposons que A et B soient deux matrices de taille $2^k \times 2^k$, composées chacune de quatre sous-matrices :

$$A = \begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \qquad B = \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix}.$$

Notons C la matrice obtenue du produit $A \cdot B$

$$C = \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}.$$

où

$$\begin{aligned} C_{1,1} &= A_{1,1} \cdot B_{1,1} + A_{1,2} \cdot B_{2,1} \\ C_{1,2} &= A_{1,1} \cdot B_{1,2} + A_{1,2} \cdot B_{2,2} \\ C_{2,1} &= A_{2,1} \cdot B_{1,1} + A_{2,2} \cdot B_{2,1} \\ C_{2,2} &= A_{2,1} \cdot B_{1,2} + A_{2,2} \cdot B_{2,2} \end{aligned}$$

Les produits de matrices apparaissant dans $C_{1,1}, C_{1,2}, C_{2,1}$ et $C_{2,2}$ sont ensuite calculés récursivement. On observe que 8 multiplications de matrices de taille $2^{k-1} \times 2^{k-1}$ sont alors nécessaires. Si on note $T(n)$ le temps d'exécution de cette approche pour calculer des matrices de taille $n \times n$, on obtient la récurrence

$$T(n) = 8 \cdot T(n/2)$$

dont la solution est $T(n) = \mathcal{O}(n^3)$.

Strassen (1969) propose de calculer la matrice C par les calculs algébriques suivants :

$$\begin{aligned} M_1 &= (A_{1,1} + A_{2,2}) \cdot (B_{1,1} + B_{2,2}) \\ M_2 &= (A_{2,1} + A_{2,2}) \cdot B_{1,1} \\ M_3 &= A_{1,1} \cdot (B_{1,2} - B_{2,2}) \\ M_4 &= A_{2,2} \cdot (B_{2,1} - B_{1,1}) \\ M_5 &= (A_{1,1} + A_{1,2}) \cdot B_{2,2} \\ M_6 &= (A_{2,1} - A_{1,1}) \cdot (B_{1,1} + B_{1,2}) \\ M_7 &= (A_{1,2} - A_{2,2}) \cdot (B_{2,1} + B_{2,2}) \end{aligned}$$

puis la matrice C est obtenue par :

$$\begin{aligned} C_{1,1} &= M_1 + M_4 - M_5 + M_7 \\ C_{1,2} &= M_3 + M_5 \\ C_{2,1} &= M_2 + M_4 \\ C_{2,2} &= M_1 - M_2 + M_3 + M_6. \end{aligned}$$

Cette fois, malgré l'apparente complexité des calculs, seulement 7 multiplications de matrices de taille $2^{k-1} \times 2^{k-1}$ sont nécessaires (les autres opérations d'addition et de soustraction de matrices sont moins coûteuses et peuvent être réalisées en temps $\mathcal{O}(n^2)$ pour des matrices $n \times n$. À présent le temps d'exécution est décrit par la récurrence

$$T(n) = 7 \cdot T(n/2)$$

qui admet pour solution $T(n) = \mathcal{O}(n^{2.807})$, donc plus rapide que l'algorithme (naïf) en $\mathcal{O}(n^3)$.

Remarque 6. *L'algorithme actuellement le plus rapide pour la multiplication de matrices est dû à Le Gall (2014) et a pour temps d'exécution $\mathcal{O}(n^{2.3728639})$.*

Une représentation compacte

Comme indiqué, notre algorithme pour résoudre ÉTIQUETAGE $L(2,1)$ cherche à la fois à limiter le nombre d'opérations (comme pour l'algorithme de Strassen), mais il utilise aussi une représentation efficace des étiquetages partiels.

Soit ℓ un entier tel que $0 \leq \ell \leq 2n$) et soit $c_\ell : V \to \{0, 1, \ldots, \ell\} \cup \{\cdot\}$. La fonction c_ℓ est une *fonction d'étiquetage* valide si pour tout sommet v tel que $c_\ell(v) \neq \cdot$, on a :

- pour tout voisin u de v, soit $c_\ell(u) = \cdot$, soit $|c_\ell(u) - c_\ell(v)| \geq 2$;
- pour tout sommet u tel que $d(u,v) \geq 2$, $c_\ell(u) \neq c_\ell(v)$.

Etant donné un entier ℓ, une façon pratique de représenter les étiquetages partiels qui utilisent (au plus) les ℓ premières étiquettes est de stocker toutes les fonctions d'étiquetages valides. Une telle fonction peut être vue comme un vecteur $(c_\ell(v_1), c_\ell(v_2), \ldots, c_\ell(v_n))$ si on dénote par $v_1, v_2, \ldots, v_n$ les sommets du graphe. Chaque fonction d'étiquetage valide c_ℓ est ensuite étendue à la fonction $c_{\ell+1}$, jusqu'à obtenir un étiquetage complet du graphe. La figure 2.26 donne toutes les fonctions d'étiquetage c_0, c_1, c_2 (vues comme des vecteurs colonnes) pour un certain graphe à 5 sommets. On observe aisément qu'une telle représentation des étiquetages partiels demande de retenir beaucoup d'informations (près de 60 fonctions d'étiquetages c_2 valides).

Nous allons maintenant donner une représentation plus compacte, qui conserve toute l'information nécessaire à l'étiquetage d'un graphe. Soit $G = (V, E)$ un graphe, où $V = \{v_1, \ldots, v_n\}$. Notre algorithme calcule successivement les tables $T_0, T_1, \ldots, T_{2n} \subseteq \{0, \bar{0}, 1, \bar{1}\}^n$, où chaque vecteur $\mathbf{a} \in T_\ell$ correspond à un étiquetage partiel. Formellement, la table T_ℓ contient un vecteur $\mathbf{a} \in \{0, \bar{0}, 1, \bar{1}\}^n$ si et seulement s'il existe un étiquetage partiel $\varphi : V \to \{0, \ldots, \ell\}$ tel que :

1. $a_i = 0$ ssi v_i n'est pas étiqueté par φ et il n'y a pas de voisin u de v_i tel que $\varphi(u) = \ell$;
2. $a_i = \bar{0}$ ssi v_i n'est pas étiqueté par φ et il existe un voisin u de v_i tel que $\varphi(u) = \ell$;
3. $a_i = 1$ ssi $\varphi(v_i) < \ell$;
4. $a_i = \bar{1}$ ssi $\varphi(v_i) = \ell$.

Avec une telle représentation, la table de la figure 2.26 (c) devient beaucoup plus compacte comme en témoigne la figure 2.27.

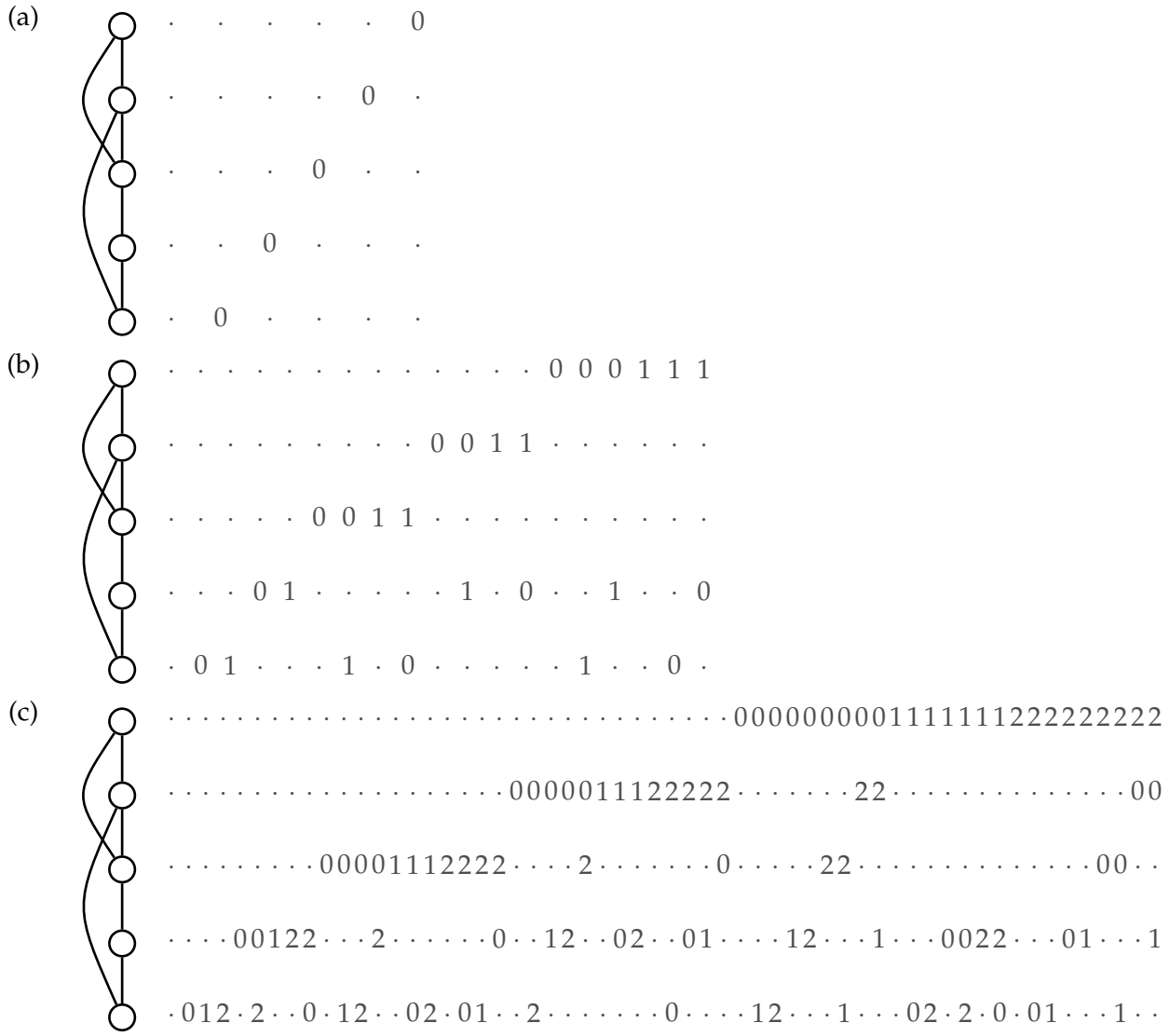


FIGURE 2.26 – Étiquetages partiels du graphe représenté à gauche en utilisant les étiquettes $\{0\}$ (fig. a), $\{0,1\}$ (fig. b) et $\{0,1,2\}$ (fig. c). Un « · » indique l'absence d'étiquette pour le sommet dans l'étiquetage partiel.

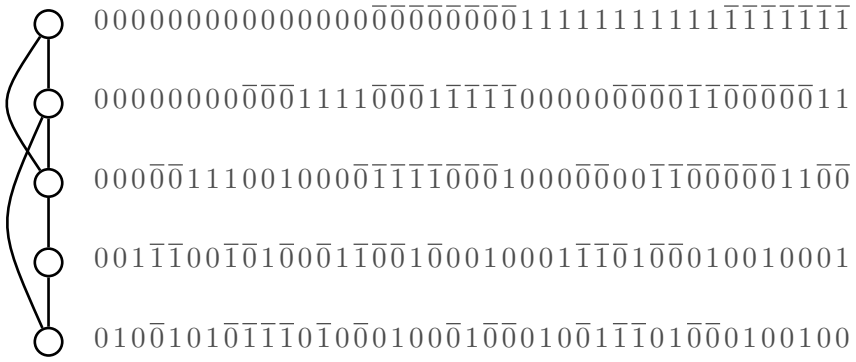


FIGURE 2.27 – Étiquetages partiels du graphe représenté à gauche (où $n = 5$) en utilisant la représentation compacte $\{0, \bar{0}, 1, \bar{1}\}^n$ pour chaque étiquetage partiel.

Calcul des tables

Il nous reste maintenant à expliquer comment on passe de la table T_ℓ (contenant tous les vecteurs $\mathbf{a} \in \{0, \bar{0}, 1, \bar{1}\}^n$ correspondant à des étiquetages partiels $\varphi : V \to \{0, \ldots, l\}$) à la table $T_{\ell+1}$.

Nous commençons par définir la fonction partielle $\oplus : \{0, \bar{0}, 1, \bar{1}\} \times \{0, 1\} \to \{0, 1, \bar{1}\}$ comme suit :

$\oplus$	0	$\bar{0}$	1	$\bar{1}$
0	0	0	1	1
1	$\bar{1}$	–	–	–

L'entrée « – » signifie que la fonction $\oplus$ n'est pas définie pour cette entrée.

Nous généralisons la fonction $\oplus$ aux vecteurs par :

$$a_1 a_2 \ldots a_n \oplus b_1 b_2 \ldots b_n = \begin{cases} (a_1 \oplus b_1) \ldots (a_n \oplus b_n) & \text{si } a_i \oplus b_i \text{ est défini pour tout} \\ & i \in \{1, \ldots, n\}, \\ - & \text{sinon,} \end{cases}$$

ainsi qu'aux ensembles de vecteurs $\mathcal{A} \subseteq \{0, \bar{0}, 1, \bar{1}\}^n$, $\mathcal{B} \subseteq \{0, 1\}^n$ par :

$$\mathcal{A} \oplus \mathcal{B} = \{\, \mathbf{a} \oplus \mathbf{b} \mid \mathbf{a} \in \mathcal{A},\ \mathbf{b} \in \mathcal{B},\ \mathbf{a} \oplus \mathbf{b} \text{ est défini} \,\}.$$

Notons par $P \subseteq \{0, 1\}^n$ les encodages de tous les 2-stables de G. Formellement, $\mathbf{p} \in P$ si et seulement s'il existe un 2-stable $S \subseteq V$ tel que pour tout i, $1 \le i \le n$, $p_i = 1$ si et seulement si $v_i \in S$.

Notre stratégie est de calculer $T_{\ell+1}$ à partir de $T_\ell \oplus P$. Si $\mathbf{a} \in T_\ell$ est un vecteur correspondant à un étiquetage $L(2,1)$ partiel φ et $\mathbf{p} \in P$ correspond à un 2-stable C de G, nous souhaitons étendre φ en posant $\varphi(u_i) = \ell + 1$ pour tout $u_i \in C$. Une telle extension est valide si et seulement si $p_i = 1$ implique $a_i = 0$, et on obtient ainsi $a_i \oplus p_i = \bar{1}$. L'ensemble de vecteurs $T_\ell \oplus P$ est presque égal à la table $T_{\ell+1}$: $\mathbf{a} \in T_{\ell+1}$ si et seulement s'il existe un $\mathbf{a}' \in T_l \oplus P$ tel que

1. $a_i = 0$ ssi $a'_i = 0$ et il n'y a pas de $v_j \in N(v_i)$ avec $a'_j = \bar{1}$;
2. $a_i = \bar{0}$ ssi $a'_i = 0$ et il y a un $v_j \in N(v_i)$ avec $a'_j = \bar{1}$;

3. $a_i = 1$ ssi $a'_i = 1$;
4. $a_i = \bar{1}$ ssi $a'_i = \bar{1}$.

Le calcul de $T_{\ell+1}$ à partir de $T_l \oplus P$ est donc facile : il suffit de regarder chaque vecteur de $T_l \oplus P$ et de déterminer pour chaque composante 0 si elle reste à 0 ou doit être changée à $\bar{0}$.

Calculer « $\oplus$ » rapidement

Il nous reste à trouver une méthode pour calculer $T_l \oplus P$ rapidement.

Nous avons besoin d'une notation supplémentaire. Soient $\mathbf{w}$ un vecteur et A un ensemble de vecteurs. On note

$$A_{\mathbf{w}} = \{\, \mathbf{v} \mid \mathbf{wv} \in A \,\}$$

l'ensemble des vecteurs $\mathbf{v}$ qui appartiennent à A si on les préfixe par $\mathbf{w}$. (La notation $\mathbf{wv}$ désigne la concaténation des vecteurs $\mathbf{w}$ et $\mathbf{v}$.)

Étant donnés deux ensembles $\mathcal{A}$ et $\mathcal{B}$ de vecteurs de taille $n > 1$, l'opération $\oplus$ sur les vecteurs de $\mathcal{A}$ et $\mathcal{B}$ peut être réalisée de la façon suivante (nous pouvons omettre les termes pour lesquels l'opération $\oplus$ n'est pas définie) :

$$\mathcal{A} \oplus \mathcal{B} = (0 \oplus 0)(\mathcal{A}_0 \oplus \mathcal{B}_0) \cup (0 \oplus 1)(\mathcal{A}_0 \oplus \mathcal{B}_1) \cup (\bar{0} \oplus 0)(\mathcal{A}_{\bar{0}} \oplus \mathcal{B}_0) \cup (1 \oplus 0)(\mathcal{A}_1 \oplus \mathcal{B}_0) \cup (\bar{1} \oplus 0)(\mathcal{A}_{\bar{1}} \oplus \mathcal{B}_0),$$

ce qui peut encore s'écrire :

$$\begin{aligned} \mathcal{A} \oplus \mathcal{B} = \quad & 0((\mathcal{A}_0 \cup \mathcal{A}_{\bar{0}}) \oplus \mathcal{B}_0) \\ \cup \quad & 1((\mathcal{A}_1 \cup \mathcal{A}_{\bar{1}}) \oplus \mathcal{B}_0) \\ \cup \quad & \bar{1}(\mathcal{A}_0 \oplus \mathcal{B}_1). \end{aligned}$$

Comme pour la multiplication de matrices, nous pouvons nous servir de cette formule pour calculer $\mathcal{A} \oplus \mathcal{B}$ de façon récursive. En utilisant la formule précédente, 3 applications de $\oplus$ sont nécessaires sur des ensembles contenant des vecteurs de taille $n-1$. En effet, puisque $\mathcal{A}$ et $\mathcal{B}$ contiennent des vecteurs de taille n, les ensembles $(\mathcal{A}_0 \cup \mathcal{A}_{\bar{0}})$, $(\mathcal{A}_1 \cup \mathcal{A}_{\bar{1}})$, $\mathcal{A}_0$, $\mathcal{B}_0$ et $\mathcal{B}_1$ contiennent chacun des vecteurs de taille $n-1$.

Si on note par $T(n)$ le temps nécessaire au calcul de $\mathcal{A} \oplus \mathcal{B}$ (par application de la méthode récursive que nous venons de décrire), lorsque les ensembles $\mathcal{A}, \mathcal{B}$ contiennent des vecteurs de taille n, $T(n)$ est décrit par la récurrence

$$T(n) = 3 \cdot T(n-1).$$

Malheureusement, cela n'est pas suffisant pour obtenir un algorithme plus rapide que $\mathcal{O}^*(3^n)$ puisque qu'on observe que la solution de cette récurrence est exactement $T(n) = 3^n$.

Supposons maintenant que nous réalisions l'opération $\oplus$ sur deux vecteurs $\mathbf{a} = (a_1, a_2)$ et $\mathbf{b} = (b_1, b_2)$ de taille 2. Supposons de plus que les deux composantes de ces vecteurs correspondent à des sommets adjacents. L'opération $\oplus$ est alors définie par :

$\oplus$	00	$0\bar{0}$	01	$0\bar{1}$	$\bar{0}0$	$\bar{0}\bar{0}$	$\bar{0}1$	$\bar{0}\bar{1}$	10	$1\bar{0}$	11	$1\bar{1}$	$\bar{1}0$	$\bar{1}\bar{0}$	$\bar{1}1$	$\bar{1}\bar{1}$
00	00	00	01	01	00	00	01	01	10	10	11	11	10	10	11	-
01	$0\bar{1}$	$\sim$	$\frown$	$\frown$	$0\bar{1}$	$\sim$	$\frown$	$\frown$	$1\bar{1}$	$\sim$	$\frown$	$\frown$	$1\bar{1}$	$\sim$	$\frown$	$\frown$
10	$\bar{1}0$	$\bar{1}0$	$\bar{1}1$	$\bar{1}1$	$\sim$	$\sim$	$\sim$	$\sim$	$\frown$	$\frown$	$\frown$	$\frown$	$\frown$	$\frown$	$\frown$	$\frown$
11	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Dans cette table, la fonction $\oplus$ appliquée à deux sommets adjacents n'est pas définie pour plusieurs entrées. Cela s'explique intuitivement par les cas suivants :

- lorsque le symbole « – » apparaît, la fonction n'est pas définie car on tente d'étiqueter avec la même étiquette les deux sommets voisins ;
- lorsque le symbole « ⌢ » apparaît, la fonction n'est pas définie car $a_1 \oplus b_1$ ou $a_2 \oplus b_2$ ne sont pas définis (la signification est que typiquement, on tente d'étiqueter un sommet déjà étiqueté) ;
- lorsque le symbole « ~ » apparaît, la fonction n'est pas définie car $a_1 \oplus b_1$ ou $a_2 \oplus b_2$ ne sont pas définis (typiquement, on tente d'étiqueter un sommet non étiqueté par un certain $\ell + 1$, mais il a déjà un voisin étiqueté par ℓ)

De façon beaucoup plus intéressante, on observe qu'une fois l'opération $\oplus$ effectuée, le préfixe $\overline{1}\overline{1}$ n'apparait pas. Lorsque nous prenons comme préfixe deux sommets adjacents, le calcul $\mathcal{A} \oplus \mathcal{B}$ s'écrit alors :

$$\begin{aligned}
\mathcal{A} \oplus \mathcal{B} = \quad & 00((\mathcal{A}_{00} \cup \mathcal{A}_{0\overline{0}} \cup \mathcal{A}_{\overline{0}0} \cup \mathcal{A}_{\overline{0}\overline{0}}) \oplus \mathcal{B}_{00}) \\
\cup \quad & 01((\mathcal{A}_{01} \cup \mathcal{A}_{0\overline{1}} \cup \mathcal{A}_{\overline{0}1} \cup \mathcal{A}_{\overline{0}\overline{1}}) \oplus \mathcal{B}_{00}) \\
\cup \quad & 10((\mathcal{A}_{10} \cup \mathcal{A}_{1\overline{0}} \cup \mathcal{A}_{\overline{1}0} \cup \mathcal{A}_{\overline{1}\overline{0}}) \oplus \mathcal{B}_{00}) \\
\cup \quad & 11((\mathcal{A}_{11} \cup \mathcal{A}_{1\overline{1}} \cup \mathcal{A}_{\overline{1}1}) \oplus \mathcal{B}_{00}) \\
\cup \quad & 0\overline{1}((\mathcal{A}_{00} \cup \mathcal{A}_{\overline{0}\overline{0}}) \oplus \mathcal{B}_{01}) \\
\cup \quad & 1\overline{1}((\mathcal{A}_{10} \cup \mathcal{A}_{\overline{1}0}) \oplus \mathcal{B}_{01}) \\
\cup \quad & \overline{1}0((\mathcal{A}_{00} \cup \mathcal{A}_{0\overline{0}}) \oplus \mathcal{B}_{10}) \\
\cup \quad & \overline{1}1((\mathcal{A}_{01} \cup \mathcal{A}_{0\overline{1}}) \oplus \mathcal{B}_{10}).
\end{aligned}$$

Seulement 8 opérations $\oplus$ sont à présent nécessaires sur des ensembles contenants des vecteurs de taille $n-2$. On obtient alors la récurrence

$$T(n) = 8 \cdot T(n-2)$$

qui a *le bon goût* d'avoir pour solution $T(n) = 8^{n/2} < 2.8285^n$.

Calculer « $\oplus$ » sur un sous-graphe connexe

Poussons un peu plus loin notre approche. Au lieu de considérer le résultat de l'opération $\oplus$ sur deux sommets voisins (et d'observer qu'ils ne peuvent simultanément être étiquetés avec une même étiquette), considérons un sous-graphe connexe (suffisamment grand, disons de taille $k' = \mathcal{O}(1)$) du graphe qu'on souhaite étiqueter. Le théorème suivant nous garantit qu'un graphe connexe peut toujours être couvert par des sous-graphes connexes de taille assez grande, et dont l'intersection entre ces sous-graphes est relativement petite.

Théorème 2.25 (Junosza-Szaniawski *et al.* (2013a), théorème 5)**.** *Soit G un graphe connexe à n sommets et soit $k < n$ un entier positif. Alors il existe des sous-graphes connexes $G_1, G_2, \ldots, G_q$ de G tels que :*

(i) chaque sommet de G appartient à au moins l'un de ces sous-graphes ;

(ii) la taille de chacun des sous-graphes $G_1, G_2, \ldots, G_{q-1}$ est au moins k et au plus $2k$ (pour G_q, nous garantissons seulement que $|V(G_q)| \leq 2k$) ;

(iii) la somme du nombre de sommets de tous ces sous-graphes est au plus $n\left(1+\frac{1}{k}\right)$.

Démonstration. Soit r un sommet arbitraire d'un graphe G. Considérons un arbre DFS T de G (c'est-à-dire, obtenu par une exploration en profondeur du graphe G), enraciné en r. Pour tout sommet v, notons $T(v)$ le sous-arbre enraciné en v. Si $|T(r)| \leq 2k$ alors on ajoute G à l'ensemble des sous-graphes désirés et on s'arrête. S'il existe un sommet v tel que $k \leq |T(v)| \leq 2k$, alors on ajoute $G[V(T(v))]$ à l'ensemble des sous-graphes désirés et on recommence récursivement ce processus avec $G \setminus V(T(v))$. Sinon, c'est qu'il existe un sommet v tel que $|T(v)| > 2k$ et pour chacun de ses fils u, $|T(u)| < k$. Dans un tel cas, on calcule un sous-ensemble $\{u_1, \ldots, u_j\}$ de ses fils tel que $k \leq |T(u_1)| + \cdots + |T(u_j)| \leq 2k-1$. On ajoute $G[\{v\} \cup V(T(u_1)) \cup \cdots \cup V(T(u_j))]$ à l'ensemble des sous-graphes désirés et on recommence récursivement avec le graphe $G \setminus (V(T(u_1)) \cup \cdots \cup V(T(u_j)))$. Cette procédure termine après au plus $\frac{n}{k}$ étapes et pour chacune d'elles, nous avons laissé au plus un sommet (v) du sous-graphe connexe désiré lors de l'appel récursif. On peut aussi remarquer que dans cette construction, le graphe $G[V(G_i) \cup V(G_2) \cup \cdots \cup V(G_q)]$ est connexe pour tout $i \in \{1, \ldots, q\}$. □

Fixons une constante k (dont nous spécifierons la valeur plus tard). Soit $G_1, G_2, \ldots, G_q$ une couverture du graphe G en sous-graphes connexes tels que garantis par le théorème précédent. Notons k' le nombre de sommets de G_1. Tant que $q > 1$, le théorème garantit $k \leq k' \leq 2k$.

Nous allons calculer $\mathcal{A} \oplus \mathcal{B}$ pour $\mathcal{A} \subseteq \{0, \bar{0}, 1, \bar{1}\}^n$, $\mathcal{B} \subseteq \{0,1\}^n$ et $n > k'$. Rappelons que $k' = \mathcal{O}(1)$. Nous pouvons calculer $\mathcal{A} \oplus \mathcal{B}$ de la façon expliquée ci-après. À première vue, cette façon de procéder peut sembler compliquée, mais elle se révèlera très utile. Cela généralise simplement l'approche expliquée précédemment pour $n = 1, 2$.

$$\begin{aligned}\mathcal{A} \oplus \mathcal{B} &= \bigcup_{\substack{\mathbf{u} \in \{0,\bar{0},1,\bar{1}\}^{k'} \\ \mathbf{v} \in \{0,1\}^{k'} \\ \text{t.q. } \mathbf{u} \oplus \mathbf{v} \text{ est défini}}} (\mathbf{u} \oplus \mathbf{v})(\mathcal{A}_{\mathbf{u}} \oplus \mathcal{B}_{\mathbf{v}}) \\ &= \bigcup_{\substack{\mathbf{v} \in \{0,1\}^{k'} \\ \mathbf{w} \in \{0,1,\bar{1}\}^{k'}}} \mathbf{w}\Big[\Big(\bigcup_{\substack{\mathbf{u} \in \{0,\bar{0},1,\bar{1}\}^{k'} \\ \text{t.q. } \mathbf{u} \oplus \mathbf{v} = \mathbf{w}}} \mathcal{A}_{\mathbf{u}}\Big) \oplus \mathcal{B}_{\mathbf{v}}\Big]\end{aligned}$$

Analysons le temps nécessaire au calcul de $\mathcal{A} \oplus \mathcal{B}$ par la formule précédente. En particulier, il nous faut évaluer le nombre d'opérations $\oplus$ sur des ensembles de vecteurs de taille $n - k'$. Nous pouvons omettre ce calcul lorsque l'ensemble $\bigcup_{\substack{\mathbf{u} \in \{0,\bar{0},1,\bar{1}\}^{k'} \\ \text{t.q. } \mathbf{u} \oplus \mathbf{v} = \mathbf{w}}} \mathcal{A}_{\mathbf{u}}$ est vide. Nous devons donc borner le nombre de paires $\mathbf{v}, \mathbf{w}$ telles qu'il y ait au moins un $\mathbf{u}$ avec $\mathbf{u} \oplus \mathbf{v} = \mathbf{w}$.

Si nous fixons $\mathbf{v}$, alors évidement pour tout i où $v_i = 1$, cela implique $w_i = \bar{1}$. Donc pour un $\mathbf{v}$ fixé, il y a $2^{k' - \|\mathbf{v}\|}$ vecteur $\mathbf{w}$, où $\|\mathbf{v}\|$ représente le nombre de positions i pour lesquelles $v_i = 1$.

Le nombre total de paires $\mathbf{v}, \mathbf{w}$ telles que $\mathbf{w} = \mathbf{v} \oplus \mathbf{u}$ pour un certain $\mathbf{u}$ (et donc qui contribuent à un ensemble $\bigcup_{\substack{\mathbf{u} \in \{0,\bar{0},1,\bar{1}\}^{k'} \\ \text{t.q. } \mathbf{u} \oplus \mathbf{v} = \mathbf{w}}} \mathcal{A}_{\mathbf{u}}$ non vide) est donc au plus

$$\sum_{\mathbf{v} \in \{0,1\}^{k'}} 2^{k' - \|\mathbf{v}\|}.$$

Utilisation des paires propres et d'une borne sur leur nombre

Un ensemble S de sommets d'un graphe $G = (V, E)$ est un 2-*packing* (ou 2-stable) si la distance entre deux sommets de S est au moins 3, c'est-à-dire si S est un stable de G^2. Une paire (S, X) de sous-ensembles de $V(G)$ est une *paire propre* si $S \cap X = \varnothing$ et S est un 2-stable de G. Si on note $\mathrm{pp}(G)$ le nombre de paires propres de G, alors par définition nous avons

$$\mathrm{pp}(G) = \sum_{\substack{S \subseteq V(G) \\ S \text{ est un 2-stable}}} 2^{n-|S|}.$$

Au théorème 2.34 (que nous démontrons à la page 97), nous établissons que le nombre maximum de paires propres dans un graphe connexe à n sommets, noté $\mathrm{pp}(n)$, est au plus $\mathcal{O}(2.6488^n)$.

Par conséquent, si les vecteurs $\mathbf{v}$ d'un ensemble $\mathcal{B}$ représentent tous les 2-stables d'un graphe connexe, alors nous pouvons trouver au plus $\mathrm{pp}(k')$ paires $\mathbf{v}, \mathbf{w}$ telles que $\mathbf{w} = \mathbf{v} \oplus \mathbf{u}$ pour un certain $\mathbf{u}$. Par conséquent,

$$\sum_{\mathbf{v} \in \{0,1\}^{k'}} 2^{k' - \|\mathbf{v}\|} \leq \mathrm{pp}(k')$$

et nous avons donc besoin de faire seulement (au plus) $\mathrm{pp}(k')$ calculs de $\oplus$ récursivement sur des vecteurs de taille $n - k'$.

Puisque nous pouvons entièrement couvrir le graphe donné en entrée par des sous-graphes dont les tailles sont comprises entre k et $2k$, ces calculs récursifs sont effectivement possibles. De plus, le théorème 2.25 implique que la longueur totale des vecteurs est au plus $n' \leq n(1 + 1/k)$.

Puisque la longueur totale des vecteurs respecte aussi $n' \geq n$, cela implique que plusieurs coordonnées correspondent à un même sommet. Cela n'est pas réellement un problème pour notre algorithme : par consistance, nous exigeons que la valeur de chaque coordonnée correspondant à un même sommet soit égale, sinon l'algorithme supprime immédiatement un tel vecteur de la table courante.

Le calcul récursif et son temps d'exécution

Pour résumer, à chaque appel récursif nous devons :

- préparer jusqu'à $\mathrm{pp}(k')$ paires[6] d'ensembles de vecteurs de longueur $n' - k'$ (où $k \leq k' \leq 2k$) ;
- calculer récursivement $\oplus$ sur ces paires ;
- de ce résultat, on obtient la prochaine table $T_{\ell+1}$ en temps linéaire.

Préparer les appels récursifs et combiner leurs résultats demande un temps linéaire en la taille de $\mathcal{A}$ et de $\mathcal{B}$. La taille de $\mathcal{B}$ est clairement au plus $\mathcal{O}(n2^{n'})$ bits. La taille de $\mathcal{A}$ est au plus $\mathcal{O}(n\,\mathrm{pp}(n'))$ bits : les « $\bar{1}$ » forment un 2-stable et il reste deux possibilités pour les autres sommets (1 ou $0/\bar{0}$).

Nous obtenons la récurrence suivante sur le temps d'exécution :

$$T_{n'} = \mathcal{O}\big(n\ \mathrm{pp}(n') + \mathrm{pp}(k')T_{n'-k'}\big) \quad \text{pour } k \leq k' \leq 2k.$$

[6] Ce sont les paires $(\mathcal{A}_{\mathbf{u}}, \mathcal{B}_{\mathbf{v}})$ où $\mathcal{A}_{\mathbf{u}}$ et $\mathcal{B}_{\mathbf{v}}$ sont des ensembles de vecteurs de longueur $n' - k'$.

Nous pouvons montrer par induction sur n' que toute solution de cette récurrence satisfait $T_{n'} = \mathcal{O}\big(nn' \operatorname{pp}(n')\big)$: en utilisant l'hypothèse d'induction sur $t_{n'-k'}$, on obtient

$$\begin{aligned} T_{n'} &= \mathcal{O}\big(n \operatorname{pp}(n')\big) + \operatorname{pp}(k')\,\mathcal{O}\big(n(n'-k')\operatorname{pp}(n'-k')\big) \\ &= \mathcal{O}\big(n \operatorname{pp}(n') + n(n'-k')\operatorname{pp}(n')\big) = \mathcal{O}\big(nn' \operatorname{pp}(n')\big). \end{aligned}$$

Finalement, en choisissant la constante k suffisamment grande pour que $2.64876^{1+1/k} \leq 2.6488$, nous obtenons $T_n \leq T_{n'} = \mathcal{O}^*(\operatorname{pp}(n(1+1/k))) \leq \mathcal{O}^*(2.6488^n)$ (la valeur 2.64876 provient de la preuve du théorème 2.34 à la page 97, puisque la racine positive de l'équation $\tau^5 = 16\tau + 88$ est $2.64875..$). Cela nous permet d'établir le théorème :

Théorème 2.26 (Junosza-Szaniawski *et al.* (2013a), théorème 6)**.** *La largeur minimum d'un étiquetage $L(2,1)$ d'un graphe connexe peut être déterminée en temps $\mathcal{O}(2.6488^n)$ et espace exponentiel.*

Pseudocode de l'algorithme

Afin de formaliser au mieux l'algorithme, nous donnons dans le reste de cette section une description en pseudocode de notre algorithme.

Soit $G = (V, E)$ un graphe donné en entrée et soit $G_1, \ldots, G_q$ une couverture de G par des sous-graphes connexes, comme assurée par le théorème 2.25. Notons d_i le nombre de sommets de G_i, pour $1 \leq i \leq q$. Notons n' la somme du nombre de sommets, c'est-à-dire $n' = d_1 + \cdots + d_q$. Soit $\mathcal{A} \subseteq \{0, \bar{0}, 1, \bar{1}\}^{n'}$ and $\mathcal{B} \subseteq \{0,1\}^{n'}$. Nous commençons par donner la description de l'algorithme **MultPlus** qui calcule $\mathcal{A} \oplus \mathcal{B}$, où

$$\mathcal{A} \oplus \mathcal{B} = \{\, \mathbf{a} \oplus \mathbf{b} \mid \mathbf{a} \in \mathcal{A},\ \mathbf{b} \in \mathcal{B},\ \mathbf{a} \oplus \mathbf{b} \text{ est défini} \,\},$$

en utilisant l'approche décrite précédemment dans cette section 2.3.2.2 (voir l'algorithme 6).

Clairement, le corps de la boucle la plus interne est exécutée exactement $24^{k'}$ fois. Toutes les opérations peuvent être réalisées en temps constant, exceptés l'union d'ensembles et l'appel récursif à **MultPlus**. Une union d'ensembles $X \cup Y$ (de vecteurs de taille n) demande au plus $n(|X| + |Y|)$ étapes si l'on implémente naïvement les ensembles comme des tableaux et qu'on les trie avec un tri par base (voir par exemple le livre de Cormen *et al.* (2009)). Sans compter les appels récursifs, le temps d'exécution est donc $\mathcal{O}(n(|\mathcal{A}| + |\mathcal{B}|))$ si $d_1 = k' = \mathcal{O}(1)$. Dans le cas (de base) où $q = 1$, le calcul de $\mathcal{A} \oplus \mathcal{B}$ est effectué naïvement en temps $\mathcal{O}(n \cdot |\mathcal{A}| \cdot |\mathcal{B}|) = \mathcal{O}(n8^{k'})$, qui est constant si $d_1 = k' = \mathcal{O}(1)$.

Soient $(v_1, \ldots, v_{n'})$ les sommets de G (où l'on autorise les doublons, comme déjà discuté) tels que

$$(v_1, \ldots, v_{n'}) = (u_{11}, \ldots, u_{1d_1},\ u_{21}, \ldots, u_{2d_2},\ \ldots\ldots,\ u_{q1}, \ldots, u_{qd_q})$$

et $G_i = (u_{i1}, \ldots, u_{id_i})$, pour $1 \leq i \leq q$. Soit k une constante suffisamment grande pour que $\operatorname{pp}(1 + 1/k) = \tau^{1+1/k} < 2.6488$, où $\tau = 2.6487..$ est la racine positive de l'équation $\tau^5 = 16\tau + 88$ comme donnée dans la preuve du théorème 2.34 à la page 97. Puisque la preuve du théorème 2.25 est constructive et fournit même un algorithme polynomial pour calculer la couverture de G en sous-graphes $G_1, \ldots, G_q$, nous pouvons effectivement calculer la décomposition de G de sorte que $k \leq d_i \leq 2k$ pour tout $1 \leq i < q$. Le théorème 2.25 nous assure également que $n' \leq n(1 + 1/k)$. L'obtention d'une telle décomposition est importante pour établir le temps d'exécution de notre algorithme, qui est intimement relié aux valeurs $\operatorname{pp}(G[G_i])$. Nous pouvons garantir que celles-ci

Algorithme 6 : MultPlus$(\mathcal{A}, \mathcal{B}, d_1, \ldots, d_q)$

Entrée : *Deux ensembles $\mathcal{A}, \mathcal{B}$ de vecteurs de taille n' ; $d_1, \ldots, d_q$ les cardinalités de $G_1, \ldots, G_q$.*
Sortie : *L'ensemble $\mathcal{A} \oplus \mathcal{B}$.*

si $q = 1$ **alors retourner** $\mathcal{A} \oplus \mathcal{B}$

$k' \leftarrow d_1$
pour chaque $\mathbf{v} \in \{0,1\}^{k'}$ **faire**
  $R \leftarrow \varnothing$
  pour chaque $\mathbf{w} \in \{0,1,\bar{1}\}^{k'}$ **faire**
    $\mathcal{A}' \leftarrow \varnothing$
    pour chaque $\mathbf{u} \in \{0,\bar{0},1,\bar{1}\}^{k'}$ **faire**
      si $\mathbf{w} = \mathbf{u} \oplus \mathbf{v}$ **alors**
        $\mathcal{A}' \leftarrow \mathcal{A}' \cup \mathcal{A}_{\mathbf{u}}$
    si $\mathcal{A}' \neq \varnothing$ **alors**
      $R \leftarrow R \cup$ MultPlus$(\mathcal{A}', \mathcal{B}_{\mathbf{v}}, d_2, \ldots, d_q)$
retourner R

sont (suffisamment) petites uniquement si les $G[G_i]$ sont connexes. Nous avons déjà établi l'analyse du temps d'exécution au théorème 2.26. Nous terminons en donnant l'algorithme **CalculeTables** qui calcule successivement les tables $T_1, \ldots, T_{2n}$ (voir l'algorithme 7).

Une fois que nous avons toutes ces tables, il est facile de trouver le plus petit ℓ tel que la table T_ℓ contienne au moins un vecteur de $\{1,\bar{1}\}^n$. Un tel vecteur correspond à (au moins) une solution où tous les sommets sont étiquetés. On en déduit alors que $\lambda_{2,1}(G) = \ell$.

2.3.3 Bornes combinatoires auxiliaires

Dans cette section, nous montrons des bornes combinatoires sur des objets introduits aux sections 2.3.1.2 et 2.3.2.2. Ces objets étant au cœur de nos algorithmes, de bonnes bornes supérieures sur leur nombre maximum sont indispensables pour garantir les temps d'exécution annoncés aux théorèmes 2.16 et 2.26.

En particulier, nous étudions aux deux prochaines sections 2.3.3.1 et 2.3.3.2 le nombre maximum d'*ensembles stables R-maximaux* et de *paires propres R-maximales* dans un *rb-graphe*. Ces deux objets combinatoires, ainsi que les rb-graphes, ont été définis à la section 2.3.1.2 (voir les pages 66 à 67) et ont été particulièrement utiles pour l'algorithme **Calcule-Lambda**, de type *diviser-pour-régner* de la section 2.3.1.2. Nous redonnons la définition de ces objets au prochain paragraphe.

À la section 2.3.3.3, nous présentons une borne sur le nombre maximum de *paires propres* dans un graphe. Les paires propres nous ont également été nécessaires à l'analyse de l'algorithme de la section 2.3.1.2, mais plus particulièrement encore pour l'analyse de la *programmation dynamique* présentée à la section 2.3.2.2. Avant de redonner quelques définitions, voici les deux résultats importants démontrés dans cette section :

Théorème 2.31 (Junosza-Szaniawski *et al.* (2013b), théorème 5)**.** *Le nombre maximum de paires propres R-maximales dans un rb-graphe à n sommets est $\Theta(\sqrt{8}^n)$.*

Algorithme 7 : CalculeTables$(G, n, d_1, \ldots, d_q, v_1, \ldots, v_{n'})$

Entrée : *Un graphe G à n sommets ; $d_1, \ldots, d_q$ les cardinalités de $G_1, \ldots, G_q$; les sommets $v_1, \ldots, v_{n'}$ avec les éventuels doublons issus de la décomposition.*

Sortie : *Les tables $T_1, \ldots, T_{2n}$ correspondant à des étiquetages partiels de G.*

$T_0 \leftarrow \{0^{n'}\}$
$P \leftarrow \varnothing$
pour chaque $\boldsymbol{x} \in \{0,1\}^{n'}$ **faire**
  si $\{ v_i$ *t.q.* $x_i = 1 \}$ *est un 2-stable* **alors**
    $P \leftarrow P \cup \{\boldsymbol{x}\}$
pour $\ell \leftarrow 1$ **à** $2n$ **faire**
  $R \leftarrow \text{MultPlus}(T_{\ell-1}, P, d_1, \ldots, d_q)$
  $T_\ell \leftarrow \varnothing$
  pour chaque $\boldsymbol{x} \in R$ **faire**
    pour $i, j \leftarrow 1$ **à** n' **faire**
      si $x_i = 0$, $x_j = \bar{1}$ *et* $v_i \in N(v_j)$ **alors**
        $x_i \leftarrow \bar{0}$
    $T_\ell \leftarrow T_\ell \cup \{\boldsymbol{x}\}$
retourner $T_1, \ldots, T_{2n}$

Théorème 2.34 (Junosza-Szaniawski *et al.* (2013a), théorème 2)**.** *Le nombre de paires propres* $\text{pp}(n)$ *dans un graphe connexe à n sommets est au plus $\mathcal{O}(2.6488^n)$.*

Rappelons qu'un *rb-graphe* $G = (V, R, B)$ est un graphe pour lequel les arêtes sont partitionnées en deux sous-ensembles R, B (rouge et noir) tels que si $vw \in B$ et $vu \in B$ alors $uw \in R \cup B$ (condition ($\triangle$)). Un ensemble stable Y est *R-maximal* si tout sommet v tel que $N_R(v) = N(v)$ est soit dans Y, soit v a un voisin dans Y. Une paire (X, Y) d'ensembles disjoints de sommets est une *paire propre R-maximale* si Y est un ensemble stable R-maximal. Pour un rb-graphe G, nous notons G_B le graphe induit par les sommets appartenant à des arêtes noires, c'est-à-dire $G[\bigcup_{e \in B(G)} e]$. De façon similaire, nous notons G_R le sous-graphe induit $G[V(G) \setminus V(G_B)]$. Nous dirons qu'un graphe G est *noir* (respectivement, *rouge*) s'il ne contient que des arêtes noires (respectivement, rouges), c'est-à-dire si $G = G_B$ (respectivement, $G = G_R$). Pour un graphe $G = (V, E)$, nous dirons que (S, X) est une *paire propre* si S et X sont des sous-ensembles de sommets disjoints, et si S est un 2-stable de G.

2.3.3.1 Sur le nombre d'ensembles stables R-maximaux

Soit $r_\ell(n)$ le nombre maximum d'ensembles stables R-maximaux à ℓ éléments dans un rb-graphe à n sommets. Pour établir la borne, nous allons utiliser les théorèmes 2.27 et 2.29 suivants.

Nous commençons par étendre le lemme 10 de Havet *et al.* (2011), qui n'était établi que pour le cas d'un nombre n pair de sommets.

Théorème 2.27. *Le nombre maximum $u_\ell(n)$ d'ensembles 2-stables à ℓ éléments dans un graphe à n sommets sans sommet isolé est au plus $\binom{\lceil n/2 \rceil}{\ell} \cdot 2^\ell$.*

Démonstration. Soit $G = (V, E)$ un graphe à n sommets. Pour simplifier la démonstration, supposons tout d'abord que n soit pair. Nous partitionnons les sommets de G en r étoiles à $a_1, a_2, \ldots, a_r$ sommets, telles que $a_1 + a_2 + \cdots + a_r = n$ et $a_i \geq 2$ pour tout $1 \leq i \leq r$ Ainsi, $r \leq \frac{n}{2}$. Pour obtenir une telle partition, il suffit de considérer un arbre couvrant T de G. Notons u et v respectivement l'extrémité et son unique voisin d'un plus long chemin de la racine à une feuille de T. Tous les voisins de v dans G (excepté au plus un voisin) sont des feuilles de T. Ces feuilles (dont u fait partie) forment avec v une étoile d'au moins 2 sommets. En la supprimant, nous obtenons un arbre plus petit T' que nous pouvons traiter de la même façon. Cette procédure se termine avec un arbre vide et produit la partition recherchée en étoiles de taille au moins 2.

Chaque 2-stable ne peut contenir qu'au plus un sommet de chaque étoile. Ainsi,

$$u_\ell \leq \sum_{1 \leq i_1 < i_2 < \cdots < i_\ell \leq r} a_{i_1} a_{i_2} \ldots a_{i_\ell}. \tag{2.6}$$

On obtient une borne supérieure sur u_ℓ en maximisant cette somme sous la condition que les nombres $a_1, a_2, \ldots, a_r$ soient des entiers, supérieurs ou égaux à 2, dont la somme est égale à n. Pour notre estimation, nous oublions un instant que ces nombres proviennent du partitionnement d'un graphe et nous regardons les a_i comme des variables ; leur nombre r varie également.

On observe que pour $a_i \geq 4$, la somme qui apparaît dans (2.6) ne décroît pas si a_i est remplacé par deux nouvelles variables $a_i' = 2$ et $a_i'' = a_i - 2$. En effet, notons $g_\ell(A) = \sum_{1 \leq i_1 < i_2 < \cdots < i_\ell \leq r} a_{i_1} a_{i_2} \ldots a_{i_\ell}$ pour $A = \{a_1, a_2, \ldots, a_r\}$. Soit $\widetilde{A} = A \setminus \{a_i\}$ et $A' = \{2, a_i - 2\} \cup \widetilde{A}$. Alors

$$\begin{aligned} g_\ell(A') &= 2\,(a_i - 2)\, g_{\ell-2}(\widetilde{A}) + 2\, g_{\ell-1}(\widetilde{A}) + (a_i - 2)\, g_{\ell-1}(\widetilde{A}) + g_\ell(\widetilde{A}) \\ &\geq a_i\, g_{\ell-1}(\widetilde{A}) + g_\ell(\widetilde{A}) \\ &= g_\ell(A). \end{aligned}$$

De façon similaire, la somme qui apparaît dans l'inégalité (2.6) n'augmente pas lorsque deux nombres a_i et a_j, tous deux égaux à 3, sont remplacés par trois nouveaux nombres égaux à 2. Puisque n est pair, on obtient que la somme de (2.6) atteint son maximum (sous l'hypothèse que les variables soient supérieures ou égales à 2 et que leur somme soit égale à n) lorsque $a_1 = a_2 = \cdots = a_r = 2$, $r = n/2$. On a donc $u_\ell \leq \binom{n/2}{\ell} \cdot 2^\ell$.

Si n est impair, alors (par les transformations précédentes) la somme de (2.6) est maximisée lorsque l'un des a_i est égal à 3 et tous les autres sont égaux à 2. Nous vérifions que la borne sur u_ℓ est encore vraie.

Sans perte de généralité, supposons que $a_1 = 3$ et que $a_i = 2$ pour $2 \leq i \leq r$, où $r = \lfloor n/2 \rfloor$. On rappelle que a_1 correspond à une étoile de taille 3, c'est-à-dire contenant trois sommets. Étant donné un 2-stable, au plus l'un de ces trois sommets y appartient. Ainsi, en notant $\hat{A} = A \setminus \{a_1\}$, nous avons la relation :

$$g_\ell(A) = 3\, g_{\ell-1}(\hat{A}) + g_\ell(\hat{A}). \tag{2.7}$$

Puisque la somme des éléments de $\hat{A}$ est paire et que pour n pair nous avons montré que $u_\ell \leq \binom{n/2}{\ell} \cdot 2^\ell$, par l'équation (2.7), nous pouvons écrire :

$$\begin{aligned} g_\ell(A) &\leq 3\binom{\frac{n-3}{2}}{\ell-1}2^{\ell-1} + \binom{\frac{n-3}{2}}{\ell}2^\ell \\ &= \frac{3\,(\frac{n-3}{2})!\,2^{\ell-1}\,\ell}{\ell!\,(\frac{n-3}{2}-\ell+1)!} + \frac{(\frac{n-3}{2})!\,2^\ell\,(\frac{n-3}{2}-\ell+1)}{\ell!\,(\frac{n-3}{2}-\ell+1)!} \\ &= \frac{(\frac{n-3}{2})!\,2^{\ell-1}\,(\ell-1+n)}{\ell!\,(\frac{n-3}{2}-\ell+1)!} \qquad (\star_1) \end{aligned}$$

On observe que lorsque n est impair, on a $\frac{n-3}{2} = \frac{n+1}{2} - 2 = \lceil \frac{n}{2} \rceil - 2$. Donc,

$$(\star_1) = \frac{(\frac{n-3}{2})!\,2^{\ell-1}\,(\ell-1+n)\,(\lceil \frac{n}{2} \rceil - \ell)}{\ell!\,(\lceil \frac{n}{2} \rceil - \ell)!}. \qquad (\star_2)$$

De plus $(\frac{n+1}{2})! = (\frac{n+1}{2})\,(\frac{n+1}{2}-1)\,(\frac{n+1}{2}-2)!$; donc $(\frac{n-3}{2})! = (\frac{n+1}{2}-2)! = (\frac{n+1}{2})!\,\frac{2}{n+1}\,\frac{2}{n-1}$. Comme n est impair, on a aussi $\lceil \frac{n}{2} \rceil = \frac{n+1}{2}$. Ainsi,

$$\begin{aligned} (\star_2) &= \frac{(\frac{n+1}{2})!\,\frac{2}{(n+1)\,(n-1)}\,2^\ell\,(\ell-1+n)\,(\lceil \frac{n}{2} \rceil - \ell)}{\ell!\,(\lceil \frac{n}{2} \rceil - \ell)!} \\ &= \frac{\lceil \frac{n}{2} \rceil!\,\frac{1}{n^2-1}\,2^\ell\,(n^2 - \ell n + 3\ell - 2\ell^2 - 1)}{\ell!\,(\lceil \frac{n}{2} \rceil - \ell)!}. \qquad (\star_3) \end{aligned}$$

Or, pour tout entier ℓ tel que $0 \leq \ell \leq n$, on a $n^2 - \ell n + 3\ell - 2\ell^2 - 1 \leq n^2 - 1$. On peut donc majorer la dernière expression pour obtenir :

$$\begin{aligned} (\star_3) &\leq \frac{\lceil \frac{n}{2} \rceil!\,2^\ell}{\ell!\,(\lceil \frac{n}{2} \rceil - \ell)!} \\ &= \binom{\lceil \frac{n}{2} \rceil}{\ell}2^\ell. \end{aligned}$$

□

Nous pouvons observer que la borne du théorème 2.27 est optimale ; il suffit pour cela de considérer un graphe constitué d'arêtes disjointes. Si on se restreint aux graphes connexes, comme l'ont observé Junosza-Szaniawski et Rzazewski (2010), cette borne peut-être améliorée. Néanmoins, nous n'utilisons pas leur borne pour démontrer le théorème 2.30, puisque notre preuve demande de considérer des sous-graphes non nécessairement connexes.

Théorème 2.28 (Junosza-Szaniawski et Rzazewski (2010))**.** *Le nombre maximum $w_\ell(n)$ de 2-stables à ℓ éléments dans un graphe connexe à n sommets est au plus $\binom{n-\ell+1}{\ell}$.*

Théorème 2.29 (Eppstein (2003a))**.** *Le nombre maximum $mis_\ell(n)$ d'ensembles stables maximaux à ℓ éléments dans un graphe à n sommets est au plus $3^{4\ell-n}4^{n-3\ell} = (81/64)^\ell(4/3)^n$.*

Nous pouvons maintenant borner la valeur de $r_\ell(n)$, qui représente le nombre maximum d'ensembles stables R-maximaux à ℓ éléments dans un rb-graphe à n sommets :

Théorème 2.30. *Soit G un rb-graphe à n sommets ayant $r_\ell(n)$ ensembles stables R-maximaux et soit $n' = |V(G_B)|$. Alors $r_\ell(n) \leq (4/3)^{n-n'}\left(\frac{81}{64}\right)^\ell \sum_{\ell'=0}^{\ell}\binom{\lceil n'/2\rceil}{\ell'}\left(\frac{128}{81}\right)^{\ell'}$.*

Démonstration. Nous pouvons construire les ensembles stables R-maximaux Y en deux étapes[7] :

1. Dans G_B, en choisissant un ensemble 2-stable Y_B, qui n'ait pas d'arête rouge (dans G) entre les sommets de Y_B.
2. Dans G_R, en choisissant un ensemble stable maximal Y_R tel que $Y_R \cap N_G(Y_B) = \varnothing$.

On obtient ainsi $Y = Y_B \cup Y_R$.

Remarquons que, comme le graphe G est un rb-graphe, par la condition ($\triangle$), si $vw \in B$ et si $vu \in B$ alors uw est une arête de G (de couleur rouge ou noire). La recherche d'un sous-ensemble $Y_B \subseteq Y$ des sommets de G_B qui soit aussi un ensemble stable dans G revient donc à rechercher un 2-stable Y_B de G_B, puis à vérifier que celui-ci n'admet pas d'arête rouge entre ses sommets. Cette dernière vérification est nécessaire car certaines arêtes rouges pourraient ne pas provenir de la condition ($\triangle$), et donc le 2-stable pourrait contenir deux sommets a, b reliés par une arête rouge.

Puisque G_B n'a aucun sommet isolé, nous pouvons appliquer la borne du théorème 2.27. Cela nous donne alors la borne :

$$\begin{aligned} r_\ell(n) &\leq \sum_{\ell'=0}^{\ell} u_{\ell'}(n') \cdot mis_{\ell-\ell'}(n-n') \\ &\leq \sum_{\ell'=0}^{\ell}\binom{\lceil n'/2\rceil}{\ell'}2^{\ell'}\left(\frac{81}{64}\right)^{\ell-\ell'}\left(\frac{4}{3}\right)^{n-n'} \\ &= (4/3)^{n-n'}\left(\frac{81}{64}\right)^{\ell}\sum_{\ell'=0}^{\ell}\binom{\lceil n'/2\rceil}{\ell'}\left(\frac{128}{81}\right)^{\ell'}. \end{aligned}$$

□

2.3.3.2 Sur le nombre de paires propres R-maximales

Notons $\rho(G)$ le nombre de paires propres R-maximales dans un rb-graphe G. Soit $\rho(n)$ la valeur maximale de $\rho(G)$ sur tous les rb-graphes à n sommets. Nous allons montrer le théorème suivant :

Théorème 2.31 (Junosza-Szaniawski *et al.* (2013b), théorème 5)**.** *Le nombre maximum de paires propres R-maximales dans un rb-graphe à n sommets est $\Theta(\sqrt{8}^n)$.*

Nous montrons tout d'abord la borne pour les graphes rouges, c'est-à-dire un rb-graphe dont toutes les arêtes sont rouges.. Soit $\rho_R(n)$ la valeur maximale de $\rho(G)$ sur tous les graphes rouges à n sommets. On remarque que les ensembles stables R-maximaux dans de tels graphes correspondent simplement aux ensembles stables maximaux. Ainsi, les paires propres R-maximales dans un graphe rouge sont les paires (X, Y) d'ensembles disjoints, où Y est un ensemble stable maximal.

Théorème 2.32. *Si le graphe G est rouge alors $\rho(G) = \mathcal{O}\left(\sqrt[5]{80}^n\right) = \mathcal{O}(2.4023^n)$, où n est le nombre de sommets de G.*

[7] Les définitions de G_B et G_R sont données à la page 91.

La preuve de ce théorème est inspirée d'une élégante preuve de Wood (2011) sur le nombre d'ensembles stables maximaux.

Démonstration. Nous montrons le théorème par induction sur le nombre n de sommets. Si $n \leq 2$, alors le nombre de paires propres R-maximales est au plus $4 < \sqrt[5]{80}^n$. Supposons que $n \geq 3$ et que la propriété soit vraie pour tous les graphes rouges ayant moins de n sommets.

Soit G un graphe rouge à n sommets tel que $\rho(G) = \rho_R(n)$. Soit v un sommet de G étant de plus petit degré $\delta(G)$. Notons que pour toute paire propre R-maximale (X,Y), au moins l'un des sommets de $N[v]$ appartient à Y (puisque Y est maximal). Soit $w \in N[v] \cap Y$. Comme l'ensemble Y est stable, aucun sommet de $N(w)$ n'appartient également à Y. Néanmoins, ils peuvent éventuellement appartenir à l'ensemble X. On obtient ainsi la récurrence suivante :

$$\rho_R(n) \leq \sum_{w \in N[v]} 2^{\deg(w)} \rho_R(n - \deg(w) - 1).$$

Soit d l'entier de $\{\delta(G), \ldots, n-1\}$ qui maximise $2^d \rho_R(n-d-1)$. Alors

$$\rho_R(n) \leq \sum_{w \in N[v]} 2^d \rho_R(n-d-1) = (\delta+1)2^d \rho_R(n-d-1) \leq (d+1)2^d \rho_R(n-d-1).$$

Nous obtenons alors que $\rho_R(n) = \mathcal{O}\left(\left(\sqrt[d+1]{(d+1)2^d}\right)^n\right)$. On peut facilement vérifier que cette expression est maximisée pour $d = 4$ et l'on peut alors établir que $\rho_R(n) = \mathcal{O}\left(\sqrt[5]{80}^n\right) = \mathcal{O}(2.4023^n)$. □

En considérant le graphe rouge H_k contenant k copies disjointes du graphe K_5, on observe que $\rho(H_k) = \Theta\left((5 \cdot 2^4)^k\right) = \Theta\left(\sqrt[5]{80}^n\right)$. Cela montre que la borne du théorème 2.32 est optimale et $\rho_R(n) = \Theta\left(\sqrt[5]{80}^n\right)$. On peut aussi observer que cette borne inférieure s'applique aux graphes rouges connexes. Pour cela, il suffit d'ajouter à H_k un sommet adjacent à précisément l'un des sommets de chaque K_5.

Intéressons-nous maintenant à borner le nombre de paires propres R-maximales dans un rb-graphe connexe noir, c'est-à-dire un rb-graphe dont toutes les arêtes sont noires.

Lemme 2.33. *Le nombre maximum de paires propres R-maximales dans un graphe noir G à n sommets, sans sommet isolé, est au plus $\mathcal{O}(\sqrt{8}^n) = \mathcal{O}(2.8285^n)$.*

Démonstration. On peut remarquer que dans un graphe noir, les paires propres R-maximales sont simplement des paires propres (c'est-à-dire des paires (X,Y) où Y est un 2-stable). Considérons les paires propres (X,Y) et posons $\ell = |Y|$. Puisque le graphe G n'a pas de sommet isolé, par le théorème 2.27 il y a $u_\ell(n)$ possibilités de choisir l'ensemble Y. Ensuite, chaque sommet qui n'appartient pas à Y peut être inclus dans X ou non. On obtient ainsi la formule :

$$\rho(G) = \sum_{\ell=0}^{n} u_\ell(n) \cdot 2^{n-\ell} = \sum_{\ell=0}^{n} \binom{\lceil n/2 \rceil}{\ell} \cdot 2^\ell \cdot 2^{n-\ell} = \mathcal{O}(\sqrt{8}^n) = \mathcal{O}(2.8285^n).$$

□

Nous pouvons à présent conclure par la preuve du théorème 2.31.

Démonstration. Il est encore une fois possible de construire toutes les paires propres R-maximales (X,Y) en deux étapes :

1. Dans G_B, en choisissant un ensemble 2-stable Y_B, qui n'ait pas d'arête rouge (dans G) entre les sommets de Y_B, et un ensemble X_B disjoint de Y_B.
2. Dans G_R, en choisissant un ensemble stable maximal Y_R tel que $Y_R \cap N(Y_B) = \varnothing$, et un ensemble X_R disjoint de Y_R.

On obtient ainsi $X = X_B \cup X_R$ et $Y = Y_B \cup Y_R$.

On observe que les paires (X,Y) ainsi construites sont exactement les paires propres R-maximales de G. Puisque le graphe G_B n'a pas de sommet isolé, pour $|V(G_B)| = n'$, on obtient la formule :

$$\rho(n) = \rho(G) = \mathcal{O}\left(\sqrt{8}^{n'} \cdot \rho_R(n-n')\right) = \mathcal{O}\left(\sqrt{8}^{n'} \cdot 2.4023^{n-n'}\right) = \mathcal{O}\left(\sqrt{8}^{n}\right) = \mathcal{O}(2.8285^n).$$

Pour montrer que cette borne est la meilleure possible, considérons le graphe M_k constitué de k arêtes noires disjointes et d'un sommet v connecté avec une arête rouge à l'un des sommets de chaque arête noire (voir la figure 2.28). Pour un tel graphe, on a $\rho(M_k) = \Theta(8^k) = \Theta(8^{n/2})$. Ainsi, $\rho(n) = \Theta(\sqrt{8}^n)$.

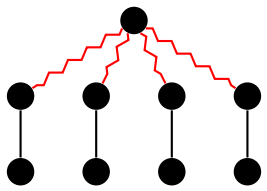


FIGURE 2.28 – Le graphe M_4, les arêtes verticales sont noires et celles en zigzag sont rouges.

□

Remarque 7. *Les paires propres R-maximales dans les graphes noirs* connexes *sont exactement les paires propres que nous allons considérer à la section suivante. Notons que nous n'utilisons pas la nouvelle borne de la section suivante pour établir le lemme 2.33, puisque ce lemme est utilisé dans la preuve du théorème 2.31 pour des graphes noirs qui ne sont pas nécessairement connexes. En particulier, pour le graphe de la figure 2.28 qui nous sert à démontrer l'optimalité de la borne supérieure sur le nombre de paires propres R-maximales dans un rb-graphe, on observe que graphe noir G_B (le graphe induit par les arêtes noires de M_k) n'est pas connexe.*

2.3.3.3 Sur le nombre de paires propres

On rappelle qu'un ensemble S de sommets d'un graphe $G = (V,E)$ est un 2-*packing* (ou 2-stable) si la distance entre deux sommets de S est au moins 3. Autrement dit S est un stable de G^2. Une paire (S,X) de sous-ensembles de $V(G)$ est une *paire propre* si $S \cap X = \varnothing$ et S est un 2-stable de G. Nous notons $\mathrm{pp}(G)$ le nombre de paires propres de G. Par définition, nous avons

$$\mathrm{pp}(G) = \sum_{\substack{S \subseteq V(G) \\ S \text{ est un 2-stable}}} 2^{n-|S|}.$$

Finalement, nous définissons le nombre maximum de paires propres dans un graphe connexe à n sommets par

$$\mathrm{pp}(n) = \max_{\substack{H \text{ est connexe} \\ |V(H)|=n}} \mathrm{pp}(H).$$

Nous montrons à présent la borne supérieure suivante :

Théorème 2.34 (Junosza-Szaniawski *et al.* (2013a), théorème 2). *Le nombre de paires propres* $\mathrm{pp}(n)$ *dans un graphe connexe à* n *sommets est au plus* $\mathcal{O}(2.6488^n)$.

Démonstration. Soit $G = (V, E)$ un graphe connexe à n sommets tel que $\mathrm{pp}(G) = \mathrm{pp}(n)$. On observe que si S est un 2-stable de G, alors pour toute arête e de G, l'ensemble S reste un 2-stable du graphe $G = (V, E \setminus \{e\})$. Puisque la suppression d'une arête n'augmente pas le nombre de paires propres et que l'on peut supprimer des arêtes tant que le graphe reste connexe, nous pouvons sans perte de généralité supposer que G est un arbre.

($*$) Supposons qu'il y a deux feuilles v_1 et v_2 de G ayant un voisin commun v_3. On observe que toute paire propre de G est également propre dans le graphe obtenu en supprimant l'arête v_1v_3 et en ajoutant l'arête v_1v_2 (voir la figure 2.29). Puisque cette opération ne réduit pas le nombre de paires propres, nous pouvons supposer qu'il n'existe pas deux (ou plusieurs) feuilles avec un voisin commun dans G.

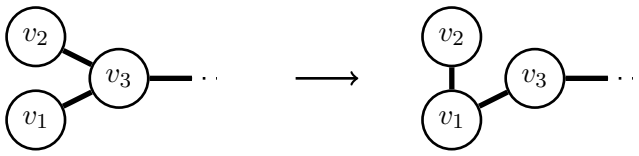


FIGURE 2.29 – Transformation de deux feuilles avec un voisin commun.

On observe facilement que $\mathrm{pp}(0) = 1$ (puisque $(\varnothing, \varnothing)$ est la seule paire propre possible du graphe vide), $\mathrm{pp}(1) = 3$ et $\mathrm{pp}(2) = 8$. Supposons que $|V(G)| \geq 3$ et notons P un plus long chemin dans G. Soit v une extrémité de P, u son voisin sur P et c ($c \neq v$) le voisin de u sur P. Par la transformation ($*$), nous pouvons supposer que $\deg(u) = 2$.

(A) Si $\deg(c) \leq 2$, alors nous pouvons partitionner les paires propres (S, X) en deux sous-ensembles : celles pour lesquelles $v \notin S$ et celles pour lesquelles $v \in S$ (voir la figure 2.30).

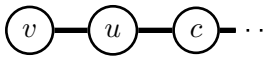


FIGURE 2.30 – Cas (A) avec $\deg(c) \leq 2$.

Notons que si $v \notin S$, alors v peut éventuellement appartenir à X . Si $v \in S$, alors aucun des deux sommets $\{u, c\}$ ne peut également appartenir à S. Chacun d'eux peut éventuellement appartenir à X Puisque les graphes $G - v$ et $G - \{v, u, c\}$ sont connexes, nous obtenons la récurrence suivante :

$$\mathrm{pp}(G) \leq 2\ \mathrm{pp}(G - v) + 4\ \mathrm{pp}(G - \{v, u, c\}) \tag{2.8}$$

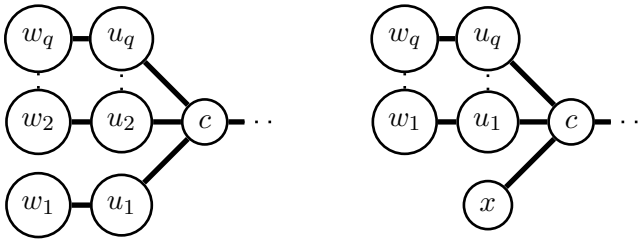
et donc

$$\mathrm{pp}(n) \leq 2\ \mathrm{pp}(n-1) + 4\ \mathrm{pp}(n-3). \tag{2.9}$$

(B) Si $\deg(c) > 2$, notons d le voisin de c (avec $d \neq u$) sur P. Soit $U = N_G(c) \setminus \{d\}$ et soit $W = N_G(U) \setminus \{c\}$. On observe que tous les sommets de W sont des feuilles de G (car sinon P ne serait pas un plus long chemin). De plus, tous les sommets de U, excepté au plus un (qui serait de degré 1), sont de degré 2 (par la transformation ($*$)). On a donc l'un des cas suivants :

(B0) Aucun sommet de U n'est une feuille de G (comme sur la figure 2.31a).

(B1) Il existe un sommet $x \in U$ qui est une feuille de G (comme sur la figure 2.31b).



(a) Cas (B0). (b) Cas (B1).

FIGURE 2.31 – Les cas (B0) et (B1).

Nous pouvons partitionner les paires propres (S, X) en celles pour lesquelles $S \cap (W \cup U) = \varnothing$ et les autres.

Si $S \cap (W \cup U) = \varnothing$, alors chaque sommet de $W \cup U$ peut appartenir à X ou être en dehors de $S \cup X$. Si $S \cap (W \cup U) = \widehat{S}$ (avec $\widehat{S} \neq \varnothing$), alors $\widehat{S}$ doit être un 2-stable de G. Ainsi, on observe que le nombre de paires propres $(\widehat{S}, \widehat{X})$ dans $G[W \cup U \cup \{c\}]$, telles que $\widehat{S} \neq \varnothing$ et $c \notin \widehat{S}$ est égal à :

1. $\underbrace{(3^q - 2^q)2^{q+1}}_{S\cap(W\cup U)=\varnothing} + \underbrace{q \cdot 3^{q-1}2^{q+1}}_{S\cap(W\cup U)\neq\varnothing} = 3^{q-1}2^{q+1}(3+q) - 2^{2q+1}$ pour $q \geq 2$ dans le cas (B0).
2. $\underbrace{(3^q - 2^q)2^{q+2}}_{S\cap(W\cup U)=\varnothing} + \underbrace{q \cdot 3^{q-1}2^{q+2}}_{S\cap(W\cup U\setminus\{x\})\neq\varnothing} + \underbrace{3^q 2^{q+1}}_{x\in S} = 3^{q-1}2^{q+1}(9+2q) - 2^{2q+2}$ pour $q \geq 1$ dans le cas (B1).

Chaque sommet de $(W \cup U \cup \{c\}) \setminus \widehat{S}$ est soit dans X, soit n'appartient pas à $S \cup X$.

Puisque les graphes $G - (W \cup U)$ et $G - (W \cup U \cup \{c\})$ sont connexes, on obtient les récurrences :

$$\mathrm{pp}(n) \leq 2^{2q}\ \mathrm{pp}(n-2q) + (3^{q-1}2^{q+1}(3+q) - 2^{2q+1})\ \mathrm{pp}(n-2q-1) \tag{2.10}$$

$$\mathrm{pp}(n) \leq 2^{2q+1}\ \mathrm{pp}(n-2q-1) + (3^{q-1}2^{q+1}(9+2q) - 2^{2q+2})\ \mathrm{pp}(n-2q-2). \tag{2.11}$$

On montre par induction sur n que pour $n \geq 0$, on a :

$$\mathrm{pp}(n) \leq 2 \cdot \tau^n \tag{2.12}$$

où $\tau = 2.6487..$ est la racine positive de l'équation $\tau^5 = 16\tau + 88$.

Il est facile de vérifier que l'inégalité (2.12) est correcte pour $n \leq 2$. Supposons donc que l'inégalité soit correcte pour toutes valeurs plus petites que n et regardons les différents cas.

(A)

$$\begin{aligned}
\mathrm{pp}(n) &\leq 2\ \mathrm{pp}(n-1) + 4\ \mathrm{pp}(n-3) \leq 4\tau^{n-1} + 8\tau^{n-3} \\
&= 4(\tau^2+2)\tau^{n-3} \\
&< 2 \cdot \tau^3 \cdot \tau^{n-3} \\
&= 2 \cdot \tau^n.
\end{aligned}$$

(B0)

$$\begin{aligned}\mathrm{pp}(n) &\leq 2^{2q}\,\mathrm{pp}(n-2q) + (3^{q-1}2^{q+1}(3+q) - 2^{2q+1})\,\mathrm{pp}(n-2q-1)\\ &\leq 2(2^{2q}\cdot\tau^{n-2q} + (3^{q-1}2^{q+1}(3+q) - 2^{2q+1})\cdot\tau^{n-2q-1})\\ &= 2\cdot\tau^n(2^{2q}\cdot\tau^{-2q} + (3^{q-1}2^{q+1}(3+q) - 2^{2q+1})\cdot\tau^{-2q-1})\\ &= 2\cdot\tau^n\Big(\Big(\frac{2}{\tau}\Big)^{2q} - \Big(\frac{2}{\tau}\Big)^{2q+1} + \frac{4(3+q)}{\tau^3}\Big(\frac{6}{\tau^2}\Big)^{q-1}\Big).\end{aligned}$$

On peut vérifier que la fonction $h_0(x) = \big(\frac{2}{\tau}\big)^{2x} - \big(\frac{2}{\tau}\big)^{2x+1} + \frac{4(3+x)}{\tau^3}\big(\frac{6}{\tau^2}\big)^{x-1}$ est décroissante pour tout réel $x > 2$ et $h_0(2) = 1$. Donc

$$\mathrm{pp}(n) \leq 2\cdot\tau^n\Big(\Big(\frac{2}{\tau}\Big)^{2q} - \Big(\frac{2}{\tau}\Big)^{2q+1} + \frac{4(3+q)}{\tau^3}\Big(\frac{6}{\tau^2}\Big)^{q-1}\Big) \leq 2\cdot\tau^n.$$

(B1)

$$\begin{aligned}\mathrm{pp}(n) &\leq 2^{2q+1}\,\mathrm{pp}(n-2q-1) + (3^{q-1}2^{q+1}(9+2q) - 2^{2q+2})\,\mathrm{pp}(n-2q-2)\\ &\leq 2(2^{2q+1}\tau^{n-2q-1} + (3^{q-1}2^{q+1}(9+2q) - 2^{2q+2})\tau^{n-2q-2})\\ &= 2\cdot\tau^n\Big(\Big(\frac{2}{\tau}\Big)^{2q+1} - \Big(\frac{2}{\tau}\Big)^{2q+2}) + \frac{4(9+2q)}{\tau^4}\Big(\frac{6}{\tau^2}\Big)^{q-1}\Big).\end{aligned}$$

Puisque la fonction $h_1(x) = \big(\frac{2}{\tau}\big)^{2x+1} - \big(\frac{2}{\tau}\big)^{2x+2} + \frac{4(9+2x)}{\tau^4}\big(\frac{6}{\tau^2}\big)^{x-1}$ est décroissante pour tout réel $x > 1$ et $h_1(1) < 1$, on obtient :

$$\mathrm{pp}(n) \leq 2\cdot\tau^n\Big(\Big(\frac{2}{\tau}\Big)^{2q+1} - \Big(\frac{2}{\tau}\Big)^{2q+2} + \frac{4(9+2q)}{\tau^4}\Big(\frac{6}{\tau^2}\Big)^{q-1}\Big) < 2\cdot\tau^n.$$

Nous avons montré que quelle que soit la structure de G, la fonction $2\cdot\tau^n$ est une borne supérieure sur le nombre de paires propres de G. Donc $\mathrm{pp}(n) = \mathcal{O}(\tau^n) = \mathcal{O}(2.6488^n)$. □

Par la construction de la preuve, on pourrait être tenté de conjecturer que le pire des cas est obtenu pour un chemin P_n à n sommets. Un simple calcul montre que $\mathrm{pp}(P_n) = \Theta(2.5943..^n)$ (la récurrence (2.9) donne $\mathrm{pp}(n) \leq 2\,\mathrm{pp}(n-1) + 4\,\mathrm{pp}(n-3)$). Néanmoins, l'exemple du théorème suivant donne une plus grande borne inférieure.

Théorème 2.35. *La valeur de* $\mathrm{pp}(n)$ *est bornée inférieurement par* $\Omega(2.6117^n)$.

Démonstration. Pour montrer le théorème, nous construisons un graphe et montrons qu'il admet $\Theta(2.6117..^n)$ paires propres. Considérons les graphes A_k, B_k, C_k et D_k de la figure 2.32.

Notons a_k, b_k et c_k le nombre de paires propres respectives dans les graphes A_k, B_k et C_k. Notons d_k le nombre de paires propres (S,X) dans le graphe D_k, pour lesquelles le 2-stable S ne contient pas le sommet barré.

En considérant séparément le nombre de paires propres (S,X), pour lesquelles S contient ou pas les sommets marqués (ceux qui sont entourés sur la figure 2.32), nous obtenons le système de récurrences :

$$\begin{cases} a_k = 2\,b_{k-1} + 4\,a_{k-1}\\ b_k = 2\,c_k + 2\,d_k\\ c_k = 2\,a_k + 12\,d_{k-1}\\ d_k = 4\,d_{k-1} + 12\,a_{k-1}.\end{cases}$$

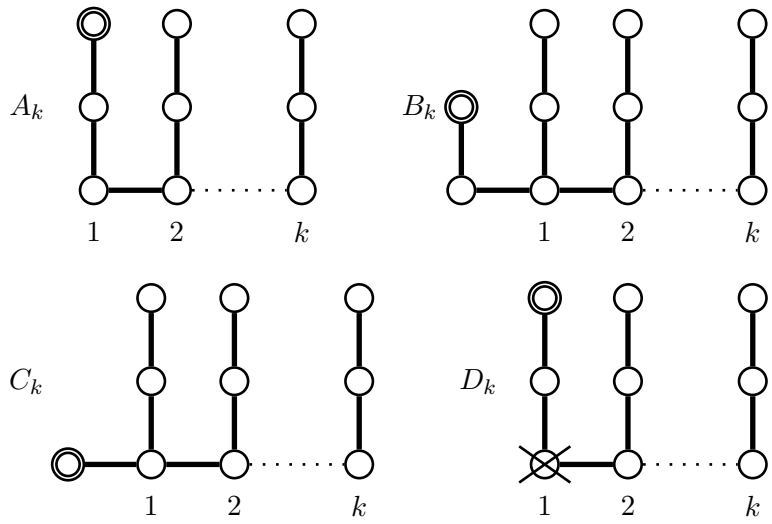


FIGURE 2.32 – Les graphes A_k, B_k, C_k et D_k de la preuve du théorème 2.35.

Nous pouvons encore réécrire ce système sous la forme suivante :

$$\begin{cases} a_k = 4\,a_{k-1} + 2\,b_{k-1} \\ b_k = 40\,a_{k-1} + 8\,b_{k-1} + 32\,d_{k-1} \\ c_k = 8\,a_{k-1} + 4\,b_{k-1} + 12\,d_{k-1} \\ d_k = 12\,a_{k-1} + 4\,d_{k-1} \end{cases}$$

autrement dit,

$$\begin{pmatrix} a_k \\ b_k \\ c_k \\ d_k \end{pmatrix} = M \cdot \begin{pmatrix} a_{k-1} \\ b_{k-1} \\ c_{k-1} \\ d_{k-1} \end{pmatrix}$$

où M est la matrice de passage définie par

$$M = \begin{pmatrix} 4 & 2 & 0 & 0 \\ 40 & 8 & 0 & 32 \\ 8 & 4 & 0 & 12 \\ 12 & 0 & 0 & 4 \end{pmatrix}.$$

Les valeurs propres de M sont les racines du polynôme caractéristique

$$p(X) = \det(M - X \cdot Id) = X^4 - 16X^3 - 576X.$$

On établit ainsi que $a_k = \Theta(\alpha^k)$, où $\alpha = 17.8149..$ est la plus grande racine positive du polynôme p. Puisque $k = n/3$, le graphe A_k contient $a_k = \Theta(17.8149..^{n/3}) = \Theta(2.6117..^n)$ paires propres. □

2.4 Conclusion

Nous avons étudié dans ce chapitre le problème ÉTIQUETAGE $L(2,1)$ sous plusieurs aspects : décision, énumération et optimisation. Nous avons proposé un algorithme en espace polynomial pour décider si un graphe admet un étiquetage $L(2,1)$ de largeur 4. Cet algorithme s'exécute en temps $\mathcal{O}(1.3161^n)$ et en utilisant la technique *mesurer-pour-conquérir* nous avons amélioré l'analyse pour montrer un temps $\mathcal{O}(1.3006^n)$. Une borne inférieure de $\Omega(1.2290^n)$ sur le temps d'exécution au pire des cas a aussi été établie. Avec une approche basée sur la décomposition linéaire, il est possible de dénombrer ces étiquetages de largeur 4 en temps $\mathcal{O}(1.1269^n)$ mais espace exponentiel. Pour la largeur 5, nous montrons qu'il est possible d'énumérer tous les étiquetages d'un graphe cubique en temps $\mathcal{O}(1.7990^n)$ et que notre algorithme de *branchement* demande au moins un temps $\Omega(1.2191^n)$ au pire des cas. Là encore, l'analyse du temps d'exécution considère une mesure non standard. Nous avons ensuite proposé des algorithmes pour déterminer la largeur minimum d'un étiquetage $L(2,1)$. Nous avons d'abord proposé un algorithme de *branchement* en espace polynomial et dont la complexité dépend de la largeur, puis nous avons donné un algorithme en temps $\mathcal{O}(7.4920^n)$ basé sur la stratégie *diviser-pour-régner*. Au prix d'un espace exponentiel, par des schémas de *programmation dynamique*, nous avons donné un algorithme en $\mathcal{O}(3.8730^n)$ puis un algorithme sophistiqué en $\mathcal{O}(2.6488^n)$ qui est le meilleur algorithme connu actuellement. L'algorithme de type *diviser-pour-régner* et celui de *programmation dynamique* demandent de considérer des objets combinatoires, en particulier les paires propres, pour lesquels nous avons établi des bornes supérieures sur leur nombre maximum.

Des extensions de nos algorithmes sont possibles. Il est à la fois pertinent de s'intéresser à des généralisations comme les étiquetages $L(h,k)$ ou $L(p_1,p_2,\ldots,p_q)$, ou au problème AFFECTATION DE FRÉQUENCES. Par ailleurs, le problème ÉTIQUETAGE $L(2,1)$ est NP-complet pour de nombreuses classes de graphes. Les bornes combinatoires sont peut-être améliorables lorsqu'on se restreint à certaines de ces classes.

Par exemple, la borne du théorème 2.27 peut être améliorée si dans la preuve nous pouvons garantir une certaine taille sur les étoiles utilisées. En définissant les graphes *d-bien partitionnés* (graphes pouvant être partitionnés en ensembles de taille au moins d, contenant chacun une étoile comme sous-graphe), nous pouvons améliorer la borne du théorème et ainsi garantir que l'algorithme **Calcule-LambdaDP** de la page 79 s'exécute en temps $\mathcal{O}((3(\frac{3+d}{3})^{1/d})^n)$ (voir Havet *et al.* (2011) pour les détails). Nous avions également posé la question « est-ce $\delta(G)$ grand implique un graphe bien partitionné ? ». Alon et Wormald (2010) y ont répondu par l'affirmative. Par ce résultat, l'algorithme **Calcule-LambdaDP** a un temps d'exécution qui peut devenir arbitrairement proche de $\mathcal{O}^*(3^n)$ selon que la valeur du degré minimum $\delta(G)$. D'une façon un peu plus générale, pour les graphes admettant un ensemble dominant de taille r, nous obtenons dans (Junosza-Szaniawski *et al.*, 2013a) une meilleure borne sur le nombre maximum de paires propres et pouvons résoudre ÉTIQUETAGE $L(2,1)$ en temps $\mathcal{O}(2^{n-r}(2+\frac{n}{r})^r)$. D'un autre côté, pour les graphes sans griffe (c'est-à-dire sans $K_{1,3}$ induit), nous montrons également dans (Junosza-Szaniawski *et al.*, 2013a) que le nombre de paires propres est au plus $\Theta(2.5943^n)$ et par conséquent que l'algorithme **CalculeTables** de la page 91 s'exécute en temps $\mathcal{O}(2.5943^n)$ sur cette classe de graphes.

Le principe d'*inclusion-exclusion* est utilisé dans l'algorithme en $\mathcal{O}^*(2^n)$, qui est le meilleur connu pour résoudre COLORATION (Björklund *et al.*, 2009). Un résultat de Cygan et Kowalik (2011) utilise également ce principe et a pour conséquence la résolution de ÉTIQUETAGE $L(2,1)$ en temps $\mathcal{O}(3^n)$. Est-il possible de diminuer cette complexité à $\mathcal{O}(2^n)$ grâce à *inclusion-exclusion* ? Cette question n'est pas triviale car, contrairement au problème COLORATION, l'interaction entre les classes de couleurs (c'est-à-dire, les ensembles de sommets ayant la même étiquette) est beaucoup plus forte pour le problème ÉTIQUETAGE $L(2,1)$.

Aussi, COLORATION peut être résolu en temps $\mathcal{O}(k^{k+\mathcal{O}(1)}n)$ sur les graphes de largeur arborescente k. Fiala *et al.* (2008a) (voir aussi Fiala *et al.* (2005)) ont montré que ÉTIQUETAGE $L(2,1)$ est déjà NP-complet sur les graphe de largeur arborescente 2. Est-il possible de définir une décomposition, peut-être un peu différente de la décomposition arborescente, où chaque paire de sommets situés à distance 2 se retrouverait dans un même sac, et ainsi obtenir un algorithme efficace paramétré par la largeur de cette nouvelle décomposition ? Bien sûr, étant donné un graphe G, la décomposition arborescente de G^2 pourrait nous aider, mais malheureusement la largeur de la décomposition de G^2 pourrait être bien plus grande que celle de G.

Chapitre 3

Autour de la domination

Ce chapitre étudie deux problèmes liés à la domination dans les graphes : DOMINATION AVEC CAPACITÉS et l'énumération des ensembles dominants minimaux. Le problème DOMINATION AVEC CAPACITÉS généralise le problème classique DOMINATION : chaque sommet se voit affecté d'une borne sur le nombre maximum de voisins qu'il peut dominer. On propose un algorithme en $\mathcal{O}(n^{20}\,1.8573^n)$ pour calculer un tel ensemble de plus petite taille possible, améliorant ainsi l'algorithme précédemment connu en $\mathcal{O}^*(1.89^n)$ (Cygan *et al.*, 2011). De façon surprenante, il est possible d'établir un compromis entre les facteurs polynomiaux et exponentiels dans l'expression du temps d'exécution, avec des temps en $O(n^d \cdot c^n)$, où $1.8463 \leq c \leq 1.8844$ et $10 \leq d \leq 40\,000$. Pour parvenir à ce résultat, nous analysons plus finement et étendons un algorithme de *programmation dynamique* de Koivisto (2009) pour un problème de partitionnement.

Dans un second temps, on s'intéresse à l'énumération des ensembles dominants minimaux dans les graphes splits et d'intervalles. Nous montrons que leur nombre est borné par $3^{n/3} \approx 1.4422^n$ et qu'ils peuvent être énumérés en temps $\mathcal{O}(3^{n/3})$. Ces résultats sont optimaux, puisqu'il existe des graphes appartenant à ces deux classes admettant un tel nombre d'ensembles dominants minimaux. Ces résultats apportent une réponse positive à une conjecture de Couturier *et al.* (2012).

Sommaire

3.1 Introduction et motivations

Déterminer un sous-ensemble d'au plus k sommets qui dominent tous les autres sommets d'un graphe est un problème classique connu sous le nom DOMINATION (en anglais, DOMINATING SET). Durant ces deux dernières décennies, un nombre considérable de publications lui a été consacré, ainsi que deux monographies (Haynes *et al.*, 1998b,a) et plusieurs thèses dont celle de l'auteur du présent manuscrit (van Rooij, 2011; Liedloff, 2007). À travers ce chapitre, nous témoignons de l'intérêt pour ce problème en s'intéressant à une généralisation et à l'énumération de certains ensembles dominants. Commençons par définir formellement le problème et par lister quelques résultats autour du problème.

DOMINATION

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

Question. Existe-t-il un sous-ensemble de sommets $D \subseteq V$ tel que $|D| \leq k$ et que chaque sommet $u \in V \setminus D$ ait au moins un voisin dans D, autrement dit tel que $N[D] = V$?

On note habituellement $\gamma(G)$ la plus petite taille d'un ensemble dominant d'un graphe G. Étant donné un graphe G, le problème d'optimisation correspondant à DOMINATION demande de calculer le paramètre $\gamma(G)$. Ce problème est déjà présent dans la longue liste de problèmes NP-complets du livre de Garey et Johnson (1979). Il reste NP-complet même lorsqu'on se restreint aux graphes cercles, cordaux, planaires, parfaits, de comparabilité, splits, bipartis et pour beaucoup d'autres classes de graphes. D'un autre côté, des algorithmes polynomiaux lui sont connus pour les graphes de co-comparabilité, d'intervalles, de permutation ou les cographes.

Le problème résiste à beaucoup d'approches : il est $W[2]$-complet et non approchable avec un facteur constant. La preuve de $W[2]$-complétude de Downey et Fellows (1999) implique qu'il est très improbable de résoudre DOMINATION avec un algorithme à paramètre fixé (sauf si $W[2] = \text{FPT}$). Johnson (1974) a montré que le problème est approchable avec un facteur $1 + \log n$, c'est-à-dire qu'il est possible de construire un ensemble dominant D tel que $\frac{|D|}{\gamma(G)} \leq 1 + \log n$ en temps polynomial. Comme résultat négatif, Raz et Safra (1997) et Feige (1998) ont montré qu'il n'est pas possible d'approximer une solution avec un facteur $c \log n$, où $c < 1$ est une constante positive, sauf si $\text{NP} \subseteq \text{DTIME}(n^{\mathcal{O}(\log \log n)})$.

Ces résultats montrent la difficulté du problème. La dernière décennie a vu le développement de plusieurs algorithmes modérément exponentiels pour le résoudre. Le premier algorithme s'exécutant plus rapidement que la recherche exhaustive est dû à Fomin *et al.* (2004) et s'exécute en temps $\mathcal{O}(1.9379^n)$. Le problème a un grand intérêt et une succesion d'algorithmes a été proposée : Grandoni (2004), Schiermeyer (2008) (voir aussi Randerath et Schiermeyer (2004)), Liedloff (2008), Fomin *et al.* (2009c), ou encore van Rooij et Bodlaender (2011). Certains de ces papiers ont introduit des techniques remarquables et ont fait progresser ce domaine de recherche, comme le papier de Fomin *et al.* (2009c) avec la technique *mesurer-pour-conquérir*. L'algorithme avec la meilleure borne au pire des cas est actuellement établi par Iwata (2011). Cet algorithme a un temps d'exécution de $\mathcal{O}(1.4864^n)$; au prix d'un espace exponentiel (dû à un pré-calcul par *programmation dynamique*), le temps d'exécution peut être réduit à $\mathcal{O}(1.4689^n)$.

Notons que Nederlof *et al.* (2014) ont également développé un algorithme qui permet de dénombrer les ensembles dominants, d'une taille donnée, en temps $\mathcal{O}(1.5673^n)$ (et $\mathcal{O}(1.5002^n)$

en espace exponentiel). Cet algorithme utilise à la fois du *branchement* et le principe d'*inclusion-exclusion*.

Quelques variantes et aussi des généralisations du problème DOMINATION ont été étudiées, à la fois sous l'angle des algorithmes exponentiels, mais aussi des algorithmes polynomiaux ou des preuves de NP-difficulté pour certaines classes de graphes. Dans la thèse de Liedloff (2007) des résultats sont consacrés à des variantes et généralisations. Parmi les variantes bien connues, on peut requérir que l'ensemble dominant satisfasse une propriété supplémentaire :

- être connexe (Fernau *et al.*, 2011; Abu-Khzam *et al.*, 2011) ;
- être une clique (Kratsch et Liedloff, 2007) ;
- être un stable (Gaspers et Liedloff, 2012) ;
- être total (l'algorithme de Fomin *et al.* (2009c) peut résoudre cette variante) ;
- ...

Diverses généralisations du problème DOMINATION ont également été proposées et notamment celle de Telle (1994). Ce problème, appelé (σ, ϱ)-DOMINATION, se définit ainsi : étant donnés deux ensembles $\sigma, \varrho \subseteq \mathbb{N}$ et un graphe $G = (V, E)$, un ensemble $S \subseteq V$ est un ensemble (σ, ϱ)-dominant si $|N(v) \cap S| \in \sigma$ pour tout $v \in S$ et $|N(v) \cap S| \in \varrho$ pour tout $v \in V \setminus S$ (voir la figure 3.1). Pour beaucoup de choix de paramètres pour σ et ϱ, nous obtenons des problèmes NP-difficiles bien connus, tels que DOMINATION ENSEMBLE STABLE, ENSEMBLE STABLE DOMINANT, COUPLAGE INDUIT, ... Si on restreint les ensembles σ et ϱ, nous montrons qu'il est possible de résoudre (et même d'en énumérer les solutions) ce problème en temps $O^*(c^n)$, pour un certain $c < 2$ (Fomin *et al.*, 2011a). Nos algorithmes sont basés sur une technique, inspirée des preuves par *déchargement*, et que nous avons appelée *brancher-et-recharger*. D'autres résultats sont obtenus pour quelques cas particuliers d'ensembles σ et ϱ grâce à la technique peu utilisée *trier-et-chercher*.

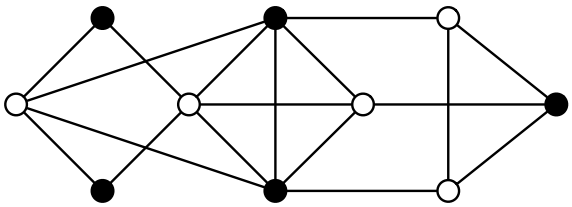


FIGURE 3.1 – Pour $\sigma = \{0, 1\}$ et $\varrho = \{2, 3, 4\}$, les sommets noirs forment un ensemble (σ, ϱ)-dominant du graphe.

Dans ce chapitre, nous présentons deux résultats. En premier lieu, on s'intéresse à une autre généralisation de DOMINATION : le problème DOMINATION AVEC CAPACITÉS. Dans ce problème, chaque sommet est affecté d'une borne sur le nombre de voisins qu'il peut dominer. La question de l'existence d'un algorithme plus rapide que celui par recherche exhaustive en $\mathcal{O}^*(2^n)$ pour DOMINATION AVEC CAPACITÉS a été pour la première fois soulevée à IWPEC 2008, puis réitérée par Johan van Rooij à Dagstuhl (voir (Fomin *et al.*, 2008c)). Le premier algorithme plus rapide que $\mathcal{O}^*(2^n)$ est dû à Cygan *et al.* (2011) et s'exécute en temps $\mathcal{O}^*(1.89^n)$ grâce au calcul d'un *couplage maximum*. Nous verrons que le problème peut d'une part effectivement être résolu par un calcul de *couplage maximum* mais aussi par une approche de *programmation dynamique* d'autre part. En exploitant des propriétés structurelles des instances ne pouvant être résolues efficacement

par un calcul de *couplage*, et en « occultant » des coûts additionnels dûs à des sous-ensembles de grandes tailles dans le schéma de *programmation dynamique*, nous obtenons un algorithme en temps $\mathcal{O}(n^d \cdot c^n)$, où $1.8463 \leq c \leq 1.8844$ et $10 \leq d \leq 40\,000$. Ce temps d'exécution est étonnant car il traduit un compromis entre le facteur polynomial et celui exponentiel : le degré d du facteur polynomial varie de façon inversement proportionnelle à c. Ce premier résultat que nous présentons dans ce chapitre a été publié dans l'article suivant :

> **(Liedloff *et al.*, 2014)** LIEDLOFF, M., TODINCA, I. et VILLANGER, Y. (2014). Solving Capacitated Dominating Set by using covering by subsets and maximum matching. *Discrete Applied Mathematics*, 168:60–68

La deuxième partie du chapitre est consacrée à l'énumération des ensembles dominants minimaux (par l'inclusion) dans les graphes splits et d'intervalles. Couturier *et al.* (2013b) montrent que pour les graphes splits et pour les graphes d'intervalles propres, leur nombre d'ensembles dominants minimaux est au plus 1.4656^n. Ils proposent également une borne inférieure de $3^{n/3} \approx 1.4422^n$. Ils conjecturent que la vraie borne supérieure correspondrait à la borne inférieure, c'est-à-dire $3^{n/3}$. Dans la deuxième partie de ce chapitre, nous répondons par l'affirmative à cette conjecture et montrons que le nombre maximum d'ensembles dominants minimaux dans les graphes splits et les graphes d'intervalles (pas seulement propres) est borné par $O^*(3^{n/3}) = O(1.4423^n)$. Ce résultat est le meilleur possible puisqu'il existe des graphes appartenant à ces deux classes ayant effectivement un tel nombre d'ensembles dominants minimaux. Notre preuve est « algorithmique » et se base sur l'analyse de complexité d'algorithmes d'énumération que nous construirons. L'algorithme pour les graphes splits ressemblera typiquement à un algorithme de *branchement* ; celui pour les graphes d'intervalles sera quelque peu différent et laissera moins apparaître la notion de branchement. Ces résultats sont extraits de la publication suivante, où ils sont complétés par une borne pour les graphes co-bipartis (ce dernier résultat a été récemment amélioré par Golovach *et al.* (2015)) :

> **(Couturier *et al.*, 2015)** COUTURIER, J., LETOURNEUR, R. et LIEDLOFF, M. (2015). On the number of minimal dominating sets on some graph classes. *Theor. Comput. Sci.*, 562:634–642

3.2 Domination avec capacités

3.2.1 Définition du problème et résultats connus

Nous considérons le problème DOMINATION AVEC CAPACITÉS qui est une généralisation du problème DOMINATION. Dans ce problème, chaque sommet se voit affecté d'une borne sur le nombre de voisins qu'il est capable de dominer en plus de lui-même. Le problème est défini formellement de la façon suivante :

> **DOMINATION AVEC CAPACITÉS**
>
> ***Entrée.*** Un graphe $G = (V, E)$, une fonction de capacités $c : V \to \mathbb{N}$ et un entier k.
>
> ***Question.*** Existe-t-il un sous-ensemble de sommets $D \subseteq V$ et une fonction $f : V \setminus D \to D$ tels que $|D| \leq k$ et

- $f(u) \in N(u) \cap D$ pour tout $u \in V \setminus D$;
- $|f^{-1}(v)| \leq c(v)$ pour tout $v \in D$?

L'ensemble D recherché est appelé un ensemble dominant avec capacités.

Si on affecte comme borne à chaque sommet son nombre de voisins, le problème DOMINATION AVEC CAPACITÉS est équivalent au problème classique DOMINATION. Par conséquent, tous les résultats de difficultés de ce dernier problème se transposent au problème DOMINATION AVEC CAPACITÉS (difficulté d'approximation, NP-complétude, W[2]-difficulté). Comme en témoignent les résultats suivants, le problème est plus difficile que DOMINATION. Par exemple, Bodlaender *et al.* (2009) ont montré que le problème DOMINATION AVEC CAPACITÉS est W[1]-difficile sur les graphes planaires, alors que DOMINATION est FPT sur ces graphes. Dom *et al.* (2008) ont quant à eux montré que le problème est W[1]-complet lorsqu'il est paramétré par la largeur de la décomposition arborescente plus la taille de la solution.

Le premier algorithme non trivial modérément exponentiel pour résoudre DOMINATION AVEC CAPACITÉS est dû à Cygan *et al.* (2011). Ils montrent que le problème peut être résolu en temps $\mathcal{O}(1.89^n)$. Pour cela, ils conçoivent tout d'abord un algorithme qui, étant donné un ensemble U de sommets, calcule en temps polynomial un ensemble dominant minimum D où $U \subseteq D$ et tel que seuls les élèments de U soient autorisés à dominer deux ou plusieurs sommets à l'extérieur de D. Nous donnerons une description détaillée de cet algorithme qui utilise une réduction au problème du couplage maximum. Ils obtiennent ensuite leur algorithme pour DOMINATION AVEC CAPACITÉS en devinant l'ensemble U, dont on peut vérifier qu'il est nécessairement de taille au plus $n/3$.

Une approche alternative pour résoudre le problème est l'utilisation de la *programmation dynamique* sur les sous-ensembles. De façon un peu brutale, on pourrait imaginer un algorithme qui parcourrait tous les sous-ensembles de sommets $X \subseteq V$, stockerait pour chacun d'eux la taille $w(X)$ d'une solution optimale à DOMINATION AVEC CAPACITÉS pour le graphe $G[X]$. La valeur de $w(X)$ pourrait s'obtenir en prenant la valeur $w(X \setminus Z) + 1$ minimale sur tous les sous-ensembles $Z \subset X$, où l'ensemble Z pourrait être dominé par un unique sommet de capacité $|Z| - 1$. On obtiendrait alors une borne de $O^*(3^n)$ sur le temps d'exécution (puisque tous les sous-ensembles X de V devraient être considérés, ainsi que tous les sous-ensembles Z de X).

Koivisto (2009) propose une approche plus astucieuse pour le problème suivant, où typiquement des ensembles de taille bornée (par une constante) sont considérés :

PARTITION EN SOUS-ENSEMBLES DE CARDINALITÉS BORNÉES

Entrée. Une famille $\mathcal{F}$ de sous-ensembles sur un univers V.

Question. Une partition de V en un nombre minimum d'éléments de $\mathcal{F}$.

Un temps d'exécution de $\mathcal{O}((2-\epsilon)^n)$ peut être obtenu si tous les ensembles de $\mathcal{F}$ sont petits, c'est-à-dire lorsque leur taille est bornée par une constante. Par exemple, si la constante c est égale à 2, le temps d'exécution est $\mathcal{O}(1.6181^n)$, pour $c = 3$, il devient $\mathcal{O}(1.8271^n)$ et pour $c = 20$, il est de $\mathcal{O}(1.9654^n)$. Par conséquent, on obtient un algorithme en temps $\mathcal{O}((2-\epsilon)^n)$ pour DOMINATION AVEC CAPACITÉS si toutes les capacités sont bornées par une constante k, pour un certain $\epsilon > 0$ (il suffit de prendre pour la famille $\mathcal{F}$ tous les sous-ensembles $X \subseteq N[v]$ tels que $|X| \leq c(v) + 1$, pour tous les sommets v du graphe).

Pour les approches de Cygan *et al.* (2011) et de Koivisto, il existe des instances qui forcent leurs algorithmes à les résoudre respectivement en temps $\mathcal{O}^*(1.89^n)$ et en temps $\mathcal{O}^*(2^n)$. On peut même construire des instances qui forcent simultanément les deux algorithmes à leur borne respective. Par exemple, considérons D un ensemble dominant avec capacité où un nombre constant de sommets W dominent ρn sommets, pour une constante $0 < \rho < 1$, et les sommets de $D \setminus W$ utilisent leur capacité pour dominer exactement deux sommets de $V \setminus D$. Dans cette instance, l'algorithme de Koivisto doit considérer des sommets qui utilisent leur capacité pour dominer ρn autres sommets. Les ensembles de la famille $\mathcal{F}$ ne sont donc pas de taille bornée. L'algorithme de Cygan *et al.* (2011) doit quant à lui tester tous les sous-ensembles U de taille $|W| + (n - |W|\rho n)/3$. Ainsi, pour ce genre d'instance, on ne peut pas espérer combiner et équilibrer les deux approches.

Comme contribution essentielle, nous étendons l'algorithme de Koivisto pour dépasser la contrainte que les ensembles de la famille $\mathcal{F}$ soient de taille bornée. Nous pourrons ainsi utiliser l'algorithme généralisé même si quelques ensembles sont de taille non bornée. Pour notre problème DOMINATION AVEC CAPACITÉS, cela signifie que l'on s'autorisera un certain nombre de sommets avec des capacités non bornées. Nous verrons que cette extension de l'algorithme a une conséquence très limitée sur le temps d'exécution. La raison est que les ensembles de taille non bornée seront traités au début de la *programmation dynamique*, avant de considérer les ensembles plus petits. Pour la résolution de DOMINATION AVEC CAPACITÉS, nous utilisons également l'approche de Cygan *et al.* (2011) pour optimiser la borne du temps d'exécution et obtenir un algorithme en $\mathcal{O}^*(1.8463^n)$. Comme nous l'avons déjà souligné, cette dernière notation cache des facteurs polynomiaux : il y a en effet un compromis entre le facteur polynomial et la base de l'exponentielle, avec des temps d'exécution entre $\mathcal{O}(n^9 \cdot 1.8844^n)$ et $\mathcal{O}(n^{40005} \cdot 1.8463^n)$.

3.2.2 Les deux ingrédients principaux

Avant de rappeler les algorithmes de Cygan *et al.* (2011) et de Koivisto (2009), nous introduisons une définition qui nous sera utile tout au long de la section.

Définition 3.1. Soit $G = (V, E)$ un graphe muni d'une fonction de capacités $c : V \to \mathbb{N}$. Soit $\mathcal{F}$ une famille de sous-ensembles de V telle que $X \subseteq V$ soit un élément de $\mathcal{F}$ si et seulement s'il existe un sommet $v \in X$ tel que $X \subseteq N[v]$ et $|X| \leq c(v) + 1$.

Un tel sommet v est désigné comme étant le *représentant*[1] de X. Nous le notons par $v = R(X)$.

Toute partition $S_1, S_2, \ldots, S_d$ de V où $S_i \in \mathcal{F}$ (avec $i \in \{1, \ldots, d\}$ et $\mathcal{F}$ défini comme précédemment) définit D, un ensemble dominant avec capacité de G, où $|D| = d$. L'ensemble dominant avec capacités D peut être retrouvé de la partition en prenant le représentant de chaque ensemble, c'est-à-dire $D = \cup_{i=1}^{d} R(S_i)$. Dans la suite, nous étudions DOMINATION AVEC CAPACITÉS comme le problème demandant de trouver une telle partition où d est minimale.

3.2.2.1 L'algorithme de Cygan *et al.*

Soit $G = (V, E)$ un graphe muni d'une fonction de capacités c. Cygan *et al.* (2011) donnent un algorithme en temps $O^*(1.89^n)$ pour DOMINATION AVEC CAPACITÉS. Essentiellement, cet algorithme utilise une réduction à un problème de couplage. Plus précisément, les auteurs considèrent le problème plus contraint ainsi défini : étant donné un ensemble U de représentants d'ensembles de

[1] Si plusieurs sommets sont candidats pour être le représentant d'un ensemble, nous choisissons de façon arbitraire un sommet parmi les candidats (par exemple, si les sommets de V sont numérotés $v_1, v_2, \ldots, v_n$, sans perte de généralité nous pouvons choisir le sommet candidat v_i ayant le plus petit indice i).

taille au moins 3, calculer un plus petit ensemble dominant avec capacité D tel que que $U \subseteq D$, et tel que $D \setminus U$ soit l'ensemble des représentants d'ensembles de taille au plus 2. En d'autres termes, les sommets de $D \setminus U$ doivent dominer au plus un sommet à l'extérieur de D. Cygan *et al.* (2011) donnent un algorithme polynomial pour ce problème (on rappelle que U fait partie de l'entrée). Nous appelons cet algorithme **ExtendSolution(**G, c, U**)**. En énumérant tous les sous-ensembles U de taille au plus $n/3$, et en appliquant **ExtendSolution(**G, c, U**)** pour chacun d'eux, le problème DOMINATION AVEC CAPACITÉS peut se résoudre en temps $O^*(\binom{n}{n/3}) = O^*(1.89^n)$. Par souci de complétude (et aussi parce que nous ferons appel à celui-ci), une description précise de l'algorithme **ExtendSolution** est donnée par l'algorithme 8.

Théorème 3.1 (Cygan *et al.* (2011))**.** *Étant donné un ensemble U de représentants d'ensembles de taille au moins 3, l'algorithme **ExtendSolution(**G, c, U**)** calcule un ensemble dominant avec capacités $D \subseteq V$, de plus petite taille possible, en temps $O(n^2 m)$ où n est le nombre de sommets de G et m son nombre d'arêtes. Par conséquent, le problème* DOMINATION AVEC CAPACITÉS *peut être résolu en temps $O^*(1.89^n)$.*

Algorithme 8 : ExtendSolution$\big(G = (V, E), c : V \to \mathbb{N}, U\big)$

Entrée : Un graphe $G = (V, E)$, une fonction de capacités c et un ensemble $U \subseteq V$ de représentants d'ensembles de taille au moins 3.

Sortie : Le nombre de sommets nécessaires en plus de ceux de U pour dominer le graphe, tel que ces sommets additionnels dominent au plus un voisin.

```
/* Construire le graphe G' = (V', E') suivant                    */
```
$V' \leftarrow \varnothing$; $E' \leftarrow \varnothing$

pour chaque $v \in V \setminus U$ **faire**
  Ajouter v à V'

pour chaque $u \in U$ **faire**
  Ajouter $c(u)$ copies de u, notées $u_1, u_2, \ldots, u_{c(u)}$ à V'

pour chaque *arête* $vu \in E$ *telle que* $v \in V \setminus U$ *et* $u \in U$ **faire**
  pour tous les *les* $i \in \{1, \ldots, c(u)\}$ **faire**
    Ajouter vu_i à E'

pour chaque *arête* $vv' \in E$ *telle que* $v, v' \in V \setminus U$ **faire**
  si $c(v) + c(v') > 0$ **alors**
    ajouter vv' à E'

Calculer un couplage maximum $\mathcal{M}$ de $G' = (V', E')$

Soit $\mathcal{M}' \subseteq \mathcal{M}$ l'ensemble des arêtes de $\mathcal{M}$ dont les deux extrémités appartiennent à $V \setminus U$

Soit $U' \subseteq V \setminus U$ l'ensemble des sommets de $V \setminus U$ n'étant pas une extrémité d'une arête de $\mathcal{M}$

retourner $|\mathcal{M}'| + |U'|$

3.2.2.2 L'algorithme de Koivisto : partition en ensembles

Koivisto (2009) étudie le problème qui consiste à trouver une partition d'un univers V en k ensembles disjoints $S_1, S_2, \ldots, S_k$ d'une famille $\mathcal{F}$ de sous-ensembles de V, sous la condition que les ensembles de $\mathcal{F}$ soient de cardinalité bornée. Grâce à un ordre linéaire (arbitraire) $<$ sur les éléments de V (ce qui implique un ordre lexicographique $\prec$ sur les sous-ensembles de V), Koivisto conçoit un algorithme dont il montre que le temps d'exécution est $\mathcal{O}^*(\alpha^n)$ avec $\alpha < 2$. Nous

commençons par décrire l'approche de Koivisto et, à la prochaine section, nous montrons comment étendre son approche si la famille $\mathcal{F}$ contient *quelques* ensembles de cardinalité *non bornée*.

Théorème 3.2 (Koivisto (2009)). *Étant donné un univers V à n éléments, un nombre k et une famille $\mathcal{F}$ de sous-ensembles de V, chacun étant de cardinalité au plus r, les partitions de V en k membres de $\mathcal{F}$ peuvent être dénombrées en temps $\mathcal{O}^*(|\mathcal{F}|2^{n\,\lambda_r})$ où $\lambda_r = \frac{2r-2}{\sqrt{(2r-1)^2-2\ln 2}}$.*

Pour établir ce résultat, l'idée de Koivisto est de définir un ordre linéaire arbitraire $<$ sur l'univers et de dénombrer les k-partitions lexicographiquement ordonnées $(S_1, S_2, \ldots, S_k)$ de V telles que $S_i < S_j$ pour tout $1 \leq i < j \leq k$. Koivisto (2009) montre que, sans perte de généralité, le nombre de k-partitions lexicographiquement ordonnées est égal au nombre de k-partitions.

L'algorithme est relativement simple et est basé sur de la *programmation dynamique*. Rappelons tout d'abord la récurrence pour compter les k-partitions ordonnées. Pour tout sous-ensemble $W \subseteq V$ et tout entier j, où $1 \leq j \leq k$, on pose $f_j(W)$ comme étant le nombre de partitions ordonnées de W en j ensembles de $\mathcal{F}$. On a ainsi les relations suivantes qui définissent l'algorithme :

$$f_1(W) = [W \in \mathcal{F}] \qquad \text{et} \qquad f_j(W) = \sum_{X \subseteq W} f_{j-1}(W \setminus X)[X \in \mathcal{F}] \quad \text{pour } j > 1. \tag{3.1}$$

La notation $[W \in \mathcal{F}]$, utilisée dans la récurrence (3.1), compte les occurrences de l'ensemble W dans la famille $\mathcal{F}$. Observons que le nombre de k-partitions d'un univers V est égal à $f_k(V)/k!$.

Le schéma de *programmation dynamique* de la récurrence (3.1) est ensuite modifié pour considérer les membres d'une partition dans un certain ordre. Comme indiqué dans (Koivisto, 2009), en ne regardant que des partitions lexicographiquement ordonnées, on obtient une réduction du nombre de sous-ensembles de V qui doivent être considérés par l'algorithme. En effet, on peut observer que l'ensemble S_j doit contenir le plus petit élément de V qui n'appartient pas à $S_1 \cup S_2 \cup \ldots \cup S_{j-1}$. Notons $\mathcal{R}_j$ la famille d'ensembles W qui correspond à l'union de tels j ensembles $S_1, S_2, \ldots, S_j$. Cette famille est récursivement définie par :

- $\mathcal{R}_1 = \{X \quad \text{t.q.} \quad X \in \mathcal{F}, \quad \min(V) \in X\}$;
- $\mathcal{R}_j = \{Y \cup X \quad \text{t.q.} \quad Y \in \mathcal{R}_{j-1}, \quad X \in \mathcal{F}, \quad Y \cap X = \varnothing, \quad \min V \setminus Y \in X\}$.

Pour étendre la récurrence (3.1) aux partitions lexicographiquement ordonnées, une fonction $g_j(W)$ est introduite. Elle associe à tout ensemble $W \subseteq V$ le nombre de partitions ordonnées $(S_1, S_2, \ldots, S_j)$ de W, en j ensembles de $\mathcal{F}$, telles que pour tout $i \in \{1, 2, \ldots, j\}$, on ait

$$\min V \setminus (S_1, S_2, \ldots, S_{i-1}) \in S_i.$$

Clairement, pour $W = V$ la condition précédente est satisfaite si et seulement si $(S_1, S_2, \ldots, S_j)$ est une partition lexicographiquement ordonnée de V. Dans la suite, nous considérons l'algorithme de *programmation dynamique* basé sur l'évaluation de $g_j(W)$ par la récurrence suivante (Koivisto, 2009) :

$$\begin{aligned} g_1(W) &= [W \in \mathcal{R}_1] = [\min V \in W \text{ et } W \in \mathcal{F}] \\ g_j(W) &= \sum_{X \subseteq W} g_{j-1}(W \setminus X)[X \in \mathcal{F}][\min V \setminus (W \setminus X) \in X] \qquad \text{pour } j > 1. \end{aligned}$$

On peut observer, tout comme Koivisto (2009), que le temps nécessaire à l'évaluation de $g_1(W)$ pour tout $W \in \mathcal{R}_1$, de $g_2(W)$ pour tout $W \in \mathcal{R}_2$, etc, est proportionnel à la quantité

$$(|\mathcal{R}_1| + |\mathcal{R}_2| + \cdots + |\mathcal{R}_k|) \cdot |\mathcal{F}|.$$

En effet, pour chaque j ($1 < j \leq k$), le calcul de g_j (et donc de $\mathcal{R}_j$) peut s'effectuer en considérant chaque ensemble $Y \in \mathcal{R}_{j-1}$ et chaque ensemble $X \in \mathcal{F}$, tels que $X \cap Y = \varnothing$ et $\min V \setminus Y \in X$.

Notons que si la cardinalité de chacun des ensembles de $\mathcal{F}$ est borné par une constante r, alors la taille de $\mathcal{F}$ est au plus n^r. Pour améliorer le théorème 3.2, il nous reste donc à établir une meilleure borne sur chaque $|\mathcal{R}_j|$. Contrairement au théorème 3.2, qui donne une expression algébrique sur le temps d'exécution, notre nouveau résultat nécessite quelques calculs. Le théorème 3.3 donne notre nouvelle borne sur le temps d'exécution et la table 3.1 à la page 114 le précise pour quelques valeurs de r.

Théorème 3.3 (Liedloff *et al.* (2014), théorème 3)**.** *Étant donnés un univers V à n éléments, un nombre k et une famille $\mathcal{F}$ de sous-ensembles de V, chacun étant de cardinalité au plus r, les partitions de V en k membres de $\mathcal{F}$ peuvent être dénombrées en temps $\mathcal{O}^*\big(|\mathcal{F}| \cdot \binom{(1-\lambda)n}{(r-1)\lambda n}\big)$ où λ est l'unique solution de $\frac{(1-\lambda r)^r}{(\lambda(r-1))^{r-1}} = 1-\lambda$ dans l'intervalle $[0; \frac{1}{2r-1}]$.*

Démonstration. Soit V un univers avec un ordre linéaire arbitraire sur ses éléments : $v_1, v_2, \ldots, v_n$. Nous notons n le nombre d'éléments de V. Soit j un entier de $\{1, 2, \ldots, k\}$. Rappelons que $\mathcal{R}_j$ est la famille composée des ensembles W égaux à l'union de j ensembles disjoints $S_1, S_2, \ldots, S_j$, chacun de taille au plus r.

Par définition de $\mathcal{R}_j$, pour tout ensemble $W \in \mathcal{R}_j$, on a $\{v_1, v_2, \ldots, v_j\} \subseteq W$. Donc la taille de $\mathcal{R}_j$ est bornée supérieurement par le nombre maximum d'ensembles W différents qu'il peut contenir. Ce nombre est borné par la plus grande valeur de $\binom{n-j}{w-j}$ pour $j \leq w \leq rj$. En effet, la taille de W (que nous notons w) est au moins j puisque W contient les j premiers sommets et au plus rj puisque W est l'union disjointe de j ensembles de taille au plus r. Comme W doit contenir les j premiers sommets, les $|W| - j$ autres sommets doivent être choisis parmi les $n - j$ sommets restants.

En notant ρn, avec $0 \leq \rho \leq 1$, la valeur de j et par nw', avec $0 \leq w' \leq (r-1)\rho$, la valeur de $w - j$, l'expression $\binom{n-j}{w-j}$ peut se réécrire $\binom{n(1-\rho)}{nw'}$. Cette dernière expression atteint son maximum pour $w' = \frac{1-\rho}{2}$ si $\frac{1-\rho}{2} \leq (r-1)\rho$, ou pour $w' = (r-1)\rho$ sinon. (Cela s'observe facilement à partir de la formule du binôme de Newton.) Notons que $\frac{1-\rho}{2} = (r-1)\rho$ pour $\rho = \frac{1}{2r-1}$.

Aussi, par l'approximation de Stirling, $\binom{\alpha n}{\beta n}$ est asymptotiquement borné par $B(\alpha, \beta)^n$ où $B : (\alpha, \beta) \mapsto \frac{\alpha^\alpha}{\beta^\beta (\alpha-\beta)^{\alpha-\beta}}$ pour tout $\alpha \geq \beta$.

Ainsi, dans la suite de cette preuve nous étudions les fonctions f_1 et f_2 (voir la figure 3.2 pour un tracé de ces deux fonctions) :

- $f_1 : \rho \mapsto B(1-\rho, (r-1)\rho)$ définie sur $[0, \frac{1}{2r-1}]$; et
- $f_2 : \rho \mapsto B(1-\rho, \frac{1-\rho}{2})$ définie sur $[\frac{1}{2r-1}, 1]$

Commençons par étudier f_2. Sa dérivée est $f_2' = -(1-\rho)^{1-\rho}(\frac{1-\rho}{2})^{\rho-1}\ln(2)$ qui est négative sur l'intervalle $[\frac{1}{2r-1}, 1]$. Donc le maximum de f_2 est obtenu pour $\rho = \frac{1}{2r-1}$. Puisque $f_2(\frac{1}{2r-1}) = f_1(\frac{1}{2r-1})$, il est suffisant de se restreindre notre analyse à la fonction f_1 sur l'intervalle $[0, \frac{1}{2r-1}]$.

La dérivée de f_1 est donnée par

$$f_1' = (1-\rho)^{1-\rho}((r-1)\rho)^{(1-r)\rho}(1-\rho r)^{\rho r-1}(-\ln(1-\rho) - r\ln((r-1)\rho) + \ln((r-1)\rho) + r\ln(1-\rho r))$$

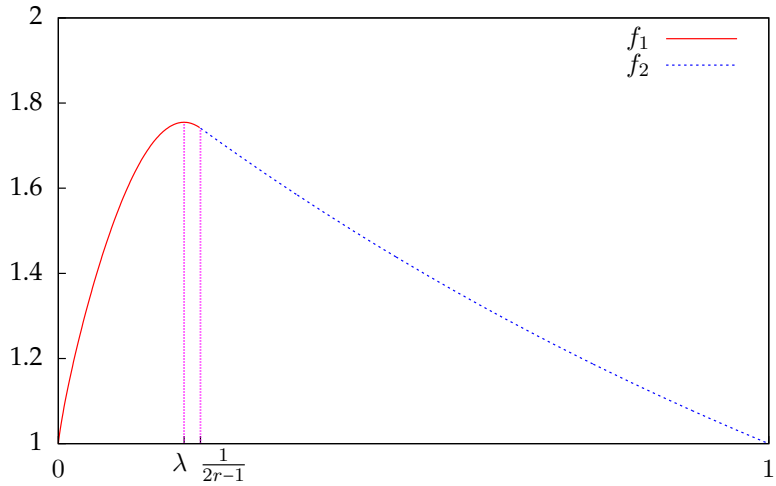


FIGURE 3.2 – Un tracé des deux fonctions f_1 et f_2, pour $r = 3$. Le maximum de f_1 est atteint pour la valeur λ.

et s'annule sur $]0, \frac{1}{2r-1}]$ si et seulement si ρ satisfait

$$(-\ln(1-\rho) - r\ln((r-1)\rho) + \ln((r-1)\rho) + r\ln(1-\rho r)) = 0.$$

En d'autres mots, f_1' s'annule à chaque fois que $\rho \in]0, \frac{1}{2r-1}]$ satisfait $\frac{(1-\rho r)^r}{(\rho(r-1))^{r-1}} = 1-\rho$ dans $]0, \frac{1}{2r-1}]$. Nous montrons maintenant qu'un tel zéro existe et qu'il est unique (ce zéro correspondra en fait au λ de la figure 3.2).

Considérons la fonction

$$\tilde{f}_1' : \rho \mapsto -\ln(1-\rho) - r\ln((r-1)\rho) + \ln((r-1)\rho) + r\ln(1-\rho r).$$

Sa dérivée est $\tilde{f}_1'' : \rho \mapsto \frac{1-r}{\rho(\rho-1)(\rho r-1)}$ et est négative sur $]0, \frac{1}{2r-1}]$ pour tout $r > 1$. Donc $\tilde{f}_1'$ est strictement décroissante sur l'intervalle $]0, \frac{1}{2r-1}]$ et admet un unique zéro puisque $\lim_{\substack{\rho\to 0 \\ \rho>0}} \tilde{f}_1'(\rho) = +\infty$ et $\tilde{f}_1'(\frac{1}{2r-1}) \leq 0$. □

Nous expliquons maintenant comment calculer une *bonne* approximation de λ (comme définie au théorème 3.3), puis nous donnons une borne asymptotique supérieure sur le temps d'exécution énoncé au théorème 3.3.

Nous avons montré que f_1' a un unique zéro sur l'intervalle $]0, \frac{1}{2r-1}]$. Il peut facilement être montré que f_1' est décroissante et monotone sur $]0, \frac{1}{2r-1}]$ avec f_1' positive sur $]0, \lambda]$ et négative sur $[\lambda, \frac{1}{2r-1}]$. Ici, λ est l'unique solution de $\frac{(1-\lambda r)^r}{(\lambda(r-1))^{r-1}} = 1-\lambda$ sur $]0, \frac{1}{2r-1}]$.

Soit ϵ un réel positif. Soit $\tilde{\lambda}$ « suffisamment proche de λ », c'est-à-dire tel que $f_1'(\tilde{\lambda}-\epsilon) \geq 0$ et $f_1'(\tilde{\lambda}+\epsilon) \leq 0$. Puisque le zéro de f_1' est unique, une telle valeur $\tilde{\lambda}$ peut être trouvée par une recherche dichotomique. Puisque f_1' est monotone, on obtient $\tilde{\lambda}-\epsilon < \lambda < \tilde{\lambda}+\epsilon$ et donc $|\tilde{\lambda}-\lambda| \leq \epsilon$. Soit $\Delta = \max\left(|f_1'(\tilde{\lambda}-\epsilon)|, |f_1'(\tilde{\lambda}+\epsilon)|\right)$. Par la classique *inégalité des accroissements finis*, $|f(\tilde{\lambda}) - f(\lambda)| \leq \Delta \cdot |\tilde{\lambda}-\lambda| \leq \Delta \cdot \epsilon$. Par conséquent, on a $f(\lambda) \leq f(\tilde{\lambda}) + \Delta \cdot \epsilon$. Finalement, on rappelle que $\mathcal{O}^*(f(\lambda)^n) = \binom{(1-\lambda)n}{(r-1)\lambda n}$.

Ainsi, il suffit de trouver un $\tilde{\lambda}$ suffisamment proche de λ, par une recherche dichotomique, afin d'établir une borne supérieure sur le temps d'exécution de l'algorithme de Koivisto. La table 3.1

r	Koivisto	**Koivisto***	$\tilde{\lambda}$
2	1.6181	**1.6181**	0.27629
3	1.7693	**1.7549**	0.17701
4	1.8271	**1.8192**	0.13051
10	1.9308	**1.9296**	0.05081
20	1.9654	**1.9651**	0.02520
50	1.9862	**1.9861**	0.01003
100	1.9931	**1.9931**	0.00501

TABLE 3.1 – Cette table donne, pour différentes valeurs de r, le temps d'exécution de l'algorithme de Koivisto (2009) d'après son analyse originale (colonne « Koivisto ») et d'après notre nouvelle analyse présentée au théorème 3.3 (colonne « Koivisto* »). Nous donnons aussi la valeur de $\tilde{\lambda}$ qui agit comme un *certificat* en prenant $\epsilon = 10^{-5}$ pour montrer la justesse des valeurs données par Koivisto*. Notons que pour le cas $r = 2$, l'analyse particulière présentée dans (Koivisto, 2009) est déjà optimale. Pour ce cas, l'existence d'une partition se réduit à la recherche d'un couplage parfait, qui peut être calculé en temps polynomial ; la version de dénombrement est un problème #P-complet (Valiant, 1979).

liste les valeurs de $\tilde{\lambda}$ pour quelques valeurs de r. Elle donne également le temps d'exécution obtenu par notre théorème 3.3 (appelé « Koivisto* ») comparé au temps d'exécution de l'analyse initiale de Koivisto (2009) (colonne « Koivisto »).

3.2.3 Un algorithme pour DOMINATION AVEC CAPACITÉS : cuisiner les ingrédients

Nous avons présenté tous les ingrédients nécessaires. Il ne reste plus qu'à les « cuisiner » pour obtenir un algorithme en temps $\mathcal{O}^*(1.8463^n)$ pour résoudre DOMINATION AVEC CAPACITÉS. Notre algorithme combine l'approche de Cygan *et al.* (2011) avec une version étendue de l'algorithme de Koivisto (2009). Précisément, nous allons montrer que l'approche de Koivisto peut être employée, même si *quelques* ensembles de la famille ne sont pas de cardinalité bornée. Pour cela, au lieu de prendre un ordre arbitraire sur les éléments de l'univers, nous mettrons certains éléments ayant un rôle particulier au début de cet ordre. Ces éléments sont les représentants des ensembles de tailles non bornées (c'est-à-dire non bornées par une constante). Comme nous le verrons, pour l'application au problème DOMINATION AVEC CAPACITÉS, ces éléments devront être *devinés*. Bien sûr, si ces éléments sont donnés comme partie de l'entrée, le temps d'exécution pourrait être amélioré. Une fois que ces représentants auront été devinés, l'algorithme n'aura à traiter que des ensembles de cardinalité bornée.

3.2.3.1 Description de l'algorithme

Soit $G = (V, E)$ un graphe et une fonction de capacités c sur ses sommets. Soit $\mathcal{F}$ la famille de sous-ensembles $X \subseteq V$ tels que $X \in \mathcal{F}$ si et seulement s'il existe un sommet $v \in X$ tel que $X \subseteq N[v]$ et $|X| \leq c(v) + 1$. Nous rappelons qu'un tel sommet $v = R(X)$ est appelé *représentant* de l'ensemble X. Remarquons qu'il est possible que deux ensembles X_1 et X_2 avec $X_1 = X_2$, $R(X_1) = u$, $R(X_2) = v$, $u \neq v$ satisfassent les propriétés requises pour appartenir à $\mathcal{F}$. Dans un tel cas, il est suffisant de ne garder arbitrairement que l'un de ces deux ensembles dans $\mathcal{F}$, avec son représentant correspondant. Ainsi, étant donné un ensemble $X \in \mathcal{F}$, son représentant est unique. Dans le reste de ce chapitre, nous appellerons *famille*, une famille $\mathcal{F}$ ainsi construite, sans préciser l'univers V et la fonction de capacités c s'il n'y a pas d'ambiguité.

Lemme 3.4. *Soit une famille $\mathcal{F}$. Si $S_1, S_2, \ldots, S_d$, où $S_i \in \mathcal{F}$ pour chaque $i \in \{1, \ldots, d\}$, est une partition de l'univers V, alors ses représentants sont deux à deux distincts.*

Démonstration. Puisque $S_1, S_2, \ldots, S_d$ est une partition de V et que pour chaque $i \in \{1, \ldots, d\}$, $R(S_i) \in S_i$, on obtient le lemme. □

Soit $\beta \in \mathbb{N}$ avec $3 < \beta \leq n$. Soit une famille $\mathcal{F}$ et soit $S_1, S_2, \ldots, S_d$ une partition de V. Nous enrichissons la notion de représentants avec les notions de *grands représentants*, *moyens représentants* et *petits représentants*, au rapport de cette partition. Nous disons qu'un sommet v est un :

- *grand représentant* s'il existe un ensemble S_i, $1 \leq i \leq d$, avec $v = R(S_i)$ et $|S_i| \geq \beta$;
- *moyen représentant* s'il existe un ensemble S_i, $1 \leq i \leq d$, avec $v = R(S_i)$ et $\beta > |S_i| \geq 3$;
- *petit représentant* s'il existe un ensemble S_i, $1 \leq i \leq d$, avec $v = R(S_i)$ et $|S_i| < 3$.

Par commodité, l'ensemble des grands (respectivement, des moyens et des petits) représentants est noté `(GR)` (respectivement, `(MR)` et `(PR)`).

Pour simplifier davantage la description de l'algorithme, étant donnés une collection disjointe $S_1, S_2, \ldots, S_d$ de sous-ensembles et un représentant v, nous notons $S(v)$ l'ensemble S_i dont v est le représentant, c'est-à-dire tel que $v = R(S_i) = R(S(v))$.

Comment fonctionne l'algorithme ? Une description formelle de l'algorithme est donnée par l'algorithme 9. Supposons que $D \subseteq V$ soit une solution de taille d de DOMINATION AVEC CAPACITÉS. Alors, il existe une partition de V en d ensembles $S_1, S_2, \ldots, S_d$ tels que chaque $v \in D$ soit le représentant de l'un des ensembles S_i avec $|S_i| \leq c(v) + 1$. Donc D peut être partitionné en `(GR)`, `(MR)` et `(PR)`.

Nous commençons d'abord à l'« étape 1 » (voir l'algorithme **minCDS**) par calculer toutes les solutions possibles (éventuellement pas de taille minimale) telles que $|\texttt{(GR)} \cup \texttt{(MR)}| \leq \gamma n$, pour un certain $\gamma \in [1/4, 1/3]$. Cela est réalisé en utilisant l'algorithme **ExtendSolution** de la section 3.2.2.1. Ensuite, dans toutes les solutions calculées à l'« étape 2 », nous pouvons supposer que $|\texttt{(GR)} \cup \texttt{(MR)}| \geq \gamma n$. Puisque tous les ensembles $S(v)$ avec $v \in \texttt{(GR)} \cup \texttt{(MR)}$ sont de cardinalité au moins 3, il s'ensuit que `(GR)` doit être de taille modérée comme le montre le prochain lemme. Cet ensemble `(GR)` de grands représentants est « deviné » par notre algorithme.

Lemme 3.5. *Supposons qu'il existe une solution $S_1, S_2, \ldots, S_d$ telle que $|\texttt{(GR)} \cup \texttt{(MR)}| \geq \gamma n$, pour $\gamma \in [0, 1]$. Alors la taille de `(GR)` est au plus $\frac{n - 3\gamma n}{\beta - 3}$.*

Démonstration. Chaque ensemble S_i tel que $R(S_i) \in \texttt{(GR)} \cup \texttt{(MR)}$ a pour cardinalité au moins 3 Donc seulement $n - 3\gamma n$ sommets peuvent être distribués sur les ensembles S_i ayant un grand représentant. Puisque les ensembles avec un grand représentant sont de taille au moins β (et contiennent déjà 3 sommets), nous obtenons que $|\texttt{(GR)}| \leq \frac{n - 3\gamma n}{\beta - 3}$. □

De plus, pour chaque ensemble S_i ayant un grand représentant, la taille de S_i n'est pas nécessairement bornée par une constante, mais une borne linéaire sur sa taille peut être établie :

Lemme 3.6. *Supposons qu'il existe une solution $S_1, S_2, \ldots, S_d$ telle que $|\texttt{(GR)} \cup \texttt{(MR)}| \geq \gamma n$, pour $\gamma \in [0, 1]$. Alors la taille de chaque S_i tel que $R(S_i) \in \texttt{(GR)}$ est au plus $n - 3\gamma n + 3$. De plus, la relation suivante est satisfaite : $|\cup_{v \in \texttt{(GR)}} S(v)| \leq \frac{n\beta - 3\gamma\beta n}{\beta - 3}$.*

Démonstration. Comme montré dans la preuve du lemme 3.5, seulement $n - 3\gamma n$ sommets peuvent être distribués sur les ensembles S_i ayant un grand représentant, en supposant que chaque S_j avec $R(S_j) \in \texttt{(GR)} \cup \texttt{(MR)}$ a pour cardinalité au moins 3. Donc chacun de ces S_i contient au plus $n - 3\gamma n + 3$ sommets. Puisque par le lemme 3.5, le nombre d'ensembles ayant un grand représentant est au plus $\frac{n-3\gamma n}{\beta-3}$, nous obtenons que $|\cup_{v \in \texttt{(GR)}} S(v)| \leq 3 \cdot \frac{n-3\gamma n}{\beta-3} + n - 3\gamma n$. □

À l'« étape 2.1 », notre algorithme considère ces ensembles S_i ayant un grand représentant. Ces ensembles ne peuvent apparaître que durant cette étape, puisque les grands représentants (devinés durant la boucle *pour*) sont placés au tout début de l'ordre $<$. Nous commençons donc le calcul des partitions de V en ensembles, en utilisant l'approche de *programmation dynamique* rappelée à la section 3.2.2.2. Ici, les tailles des ensembles de $\tilde{\mathcal{F}}$ ne sont pas bornées par une constante. Néanmoins, nous verrons à la section 3.2.3.2 qu'une *bonne* borne sur le temps d'exécution peut être établie. Finalement, à l'« étape 2.2 », il ne reste plus que des ensembles de taille bornée par la constante $\beta - 1$, et la *programmation dynamique* est poursuivie. La valeur i minimale nécessaire pour partitionner le graphe en i ensembles est calculée et on obtient la solution optimale en la comparant avec la meilleure solution trouvée à l'étape 1.

Remarque 8. *Il n'est pas difficile de modifier l'algorithme **minCDS** de sorte à ce qu'il retourne effectivement un ensemble dominant avec capacité minimum plutôt que seulement sa cardinalité.*

3.2.3.2 Analyse du temps d'exécution

Dans cette section, nous montrons que le temps d'exécution au pire des cas de l'algorithme **minCDS** est $\mathcal{O}^*(1.8573^n)$ (en utilisant $\gamma = 31/100$ et $\beta = 15$ comme dans l'algorithme). Avec des valeurs appropriées pour γ et β (qui sont utilisées comme des constantes par l'algorithme **minCDS**), ce pire des cas peut être ramené à $\mathcal{O}^*(1.8463^n)$. Notons toutefois quand dans cette dernière expression, un polynôme de grand degré est caché par la notation $\mathcal{O}^*$. Nous discuterons de ce polynôme à la section 3.2.3.2 (voir aussi la table 3.2).

Lemme 3.7. *Le temps d'exécution de l'étape 1 est borné par* $\mathcal{O}^*(n^3 m\binom{n}{\gamma n})$.

Démonstration. Le nombre total d'ensembles U est $\sum_{\ell=1}^{\gamma n}\binom{n}{\ell} \leq n\binom{n}{\gamma n}$ puisque $\gamma \leq 1/3$. Chaque appel à la procédure **ExtendSolution** coûte un temps $\mathcal{O}(n^2 m)$ par le théorème 3.1. Donc le temps total pour cette étape est borné par $\mathcal{O}(n^3 m\binom{n}{\gamma n})$. □

Par le même argument que précédemment, combiné avec les lemmes 3.5 et 3.6 on établit :

Lemme 3.8. *Le nombre d'ensembles* $\texttt{(GR)}$ *considérés par la boucle « PourChaque » la plus externe de l'étape 2 est au plus* $\mathcal{O}\left(n\binom{n}{\frac{n-3\gamma n}{\beta-3}}\right)$.

Lemme 3.9. *La taille de la famille* $\tilde{\mathcal{F}}$ *est bornée par* $\mathcal{O}(n\binom{n}{n-3\gamma n+3})$.

Lemme 3.10. *Soit* $\gamma \geq 2/7$. *Le temps d'exécution de l'étape 2.1 est borné par* $\mathcal{O}\left(n^4|\tilde{\mathcal{F}}| \cdot \binom{(1-\lambda)n}{(2\lambda+1-3\gamma)n}\right)$ *où* λ *est l' (éventuelle) unique racine réelle de* $27(\gamma-\lambda)^3 - (1-\lambda)(1-3\gamma+2\lambda)^2$ *dans* $[0, \frac{1-3\gamma}{\beta-3}]$ *si une telle solution existe, ou alors le temps d'exécution est borné par* $\mathcal{O}\left(n^4|\tilde{\mathcal{F}}| \cdot \binom{(\frac{\beta-4+3\gamma}{\beta-3})n}{(\frac{3\gamma+\beta-3\gamma\beta-1}{\beta-3})n}\right)$.

Algorithme 9 : minCD$\big(G=(V,E), c: V \to \mathbb{N}\big)$

Entrée : Un graphe $G=(V,E)$ et une fonction de capacités c.
Sortie : La taille minimum d'un ensemble dominant avec capacités de G.

```
/* γ est une constante à choisir dans [1/4, 1/3] et β est une constante telle
   que β > 3 (voir la section 3.2.3.2)                                        */

γ ← 31/100
β ← 15
MinSol ← +∞

/* -- étape 1 : basée sur l'approche de Cygan et al. --                       */
pour ℓ = 0 à γn faire
    pour chaque U ⊆ V de taille ℓ faire
        MinSol ← min{MinSol, ℓ + ExtendSolution(G, c, U)}

/* -- étape 2 : basée sur l'approche de Koivisto --                           */
pour ℓ = 0 à (n−3γn)/(β−3) faire
    pour chaque (GR) ⊆ V de taille ℓ faire
        Définir un ordre < en mettant d'abord les sommets de (GR) (dans un ordre arbitraire) puis les
        sommets de V ∖ (GR) (dans un ordre arbitraire)
        pour tous les i ∈ {0, 1, ..., n} faire R_i ← {∅}

        /* - étape 2.1: traitement des ensembles de taille ≥ β -              */
        z ← n − 3γn
        Soit F̃ = {X ⊆ V, |X| ≤ z + 3, ∃v ∈ (GR)  t.q.  X ⊆ N[v] et |X| ≤ c(v) + 1}
        pour i = 1 à ℓ faire
            pour chaque Y ∈ R_{i−1} et X ∈ F̃ t.q. Y ∩ X = ∅ et min V ∖ Y ∈ X faire
                Ajouter Y ∪ X à R_i

        /* - étape 2.2: traitement des ensembles de taille < β -              */
        Soit F = {X ⊆ V, |X| < β, ∃v ∈ V ∖ (GR) t.q. X ⊆ N[v] et |X| ≤ c(v) + 1}
        pour i = ℓ + 1 à n faire
            pour chaque Y ∈ R_{i−1} et X ∈ F t.q. Y ∩ X = ∅ et min V ∖ Y ∈ X faire
                Ajouter Y ∪ X à R_i

        Soit i l'indice minimum tel que V ∈ R_i
        MinSol ← min{MinSol, i}
retourner MinSol
```

Démonstration. Nous donnons d'abord une borne sur la taille de la famille $\mathcal{R}_i$, $1 \le i \le \ell$, où ℓ représente la taille de (GR). Pour chaque $W \in \mathcal{R}_i$, sa taille est au plus $3i+n-3\gamma n$ (rappelons que W est l'union d'au plus i ensembles de S de cardinalité au moins 3 ; en outre, au plus $n-3\gamma n$ sommets peuvent être distribués sur tous ces ensembles – voir le lemme 3.6 et sa preuve). Néanmoins, à cause de l'ordre $<$ et par construction de $W \in \mathcal{R}_i$, chaque $W \in \mathcal{R}_i$ doit contenir les i premiers éléments. Il y a donc au plus $\sum_{\kappa=0}^{2i+n-3\gamma n} \binom{n-i}{\kappa}$ ensembles W possibles dans $\mathcal{R}_i$.

On note que $2i + n - 3\gamma n \leq (n-i)/2$, pour tout $1 \leq i \leq \ell$, à chaque fois que $i \leq \frac{6\gamma n - n}{5}$. En posant $i = \rho n$ nous obtenons $\rho \leq \frac{6\gamma-1}{5}$, avec $\rho \in [0, \frac{1-3\gamma}{\beta-3}]$. Comme $\frac{1-3\gamma}{\beta-3} \leq \frac{6\gamma-1}{5}$ pour tout $\gamma \geq \frac{\beta+2}{6\beta-3}$, et puisque $\beta \geq 4$, nous supposons dans le reste de la preuve que $\gamma \geq \frac{2}{7}$. Sous cette hypothèse, le plus grand terme de la somme $\sum_{\kappa=0}^{2i+n-3\gamma n} \binom{n-i}{\kappa}$ est le dernier.

Comme énoncé au théorème 3.3, par l'approximation de Stirling, $\binom{\alpha n}{\beta n}$ est asymptotiquement borné par $B(\alpha,\beta)^n$ où $B : (\alpha,\beta) \mapsto \frac{\alpha^\alpha}{\beta^\beta(\alpha-\beta)^{\alpha-\beta}}$. Considérons à nouveau la fonction $f_1 : \rho \mapsto B(1-\rho, 2\rho+1-3\gamma)$ définie sur $[0, \frac{1-3\gamma}{\beta-3}]$. Sa dérivée est égale à zéro si et seulement si $3\ln(3) + 3\ln(\gamma-\rho) - \ln(1-\rho) - 2\ln(1-3\gamma+2\rho) = 0$, ou encore autrement dit, à chaque fois que l'on a $27(\gamma-\rho)^3 = (1-\rho)(1-3\gamma+2\rho)^2$. Par des calculs, on vérifie que cette équation a une unique solution, si elle existe, dans l'intervalle $[0, \frac{1-3\gamma}{\beta-3}]$. Sinon, si aucune solution n'existe, alors f_1 est croissante sur $[0, \frac{1-3\gamma}{\beta-3}]$ et son maximum est $f_1(\frac{1-3\gamma}{\beta-3})$. Estimons (certes un peu grossièrement) une borne supérieure sur la contribution polynomiale au temps d'exécution : (i) un facteur n est nécessaire à la boucle sur les i ; (ii) un facteur n^2 est nécessaire pour retrouver les ensembles dans les familles $\tilde{\mathcal{F}}$ and $\mathcal{R}$, pour s'assurer que $\min V \setminus Y$ appartient à X, pour vérifier que $Y \cap X = \emptyset$, et pour construire le nouvel ensemble $Y \cup X$; (iii) un facteur n supplémentaire peut être nécessaire car chaque ensemble $\mathcal{R}_i$ peut contenir des sous-ensembles de tailles différentes (toutes comprises entre i et $3i + n - 3\gamma n$). □

Lemme 3.11. *Soit $\gamma \in [1/4, 1/3]$. Le temps d'exécution de l'étape 2.2 est borné par $\mathcal{O}\big(n^4 \cdot n^\beta \cdot \binom{(1-\lambda)n}{(1-3\gamma+2\lambda)n}\big)$ où λ est l'unique racine réelle de $27(\gamma-\lambda)^3 - (1-\lambda)(1-3\gamma+2\lambda)^2$ sur $[0, \frac{6\gamma-1}{5}]$.*

Démonstration. La preuve est assez similaire à la précédente. À nouveau, chaque $W \in \mathcal{R}_i$, $\ell < i \leq n$, est de taille au plus $3i + n - 3\gamma n$. Puisqu'il est requis que chacun de ces ensembles W contienne les i premiers éléments, il s'ensuit que la taille de $\mathcal{R}_i$ est au plus $\binom{n-i}{2i+n-3\gamma n}$.

Considérons maintenant les fonctions $f_1 : \rho \mapsto B(1-\rho, 1-3\gamma+2\rho)$ définie sur $[0; \frac{6\gamma-1}{5}]$ et $f_2 : \rho \mapsto B(1-\rho, \frac{1-\rho}{2})$ définie sur $[\frac{6\gamma-1}{5}; 1]$. (Pour justifier cette découpe entre f_1 et f_2, observons que $(1-\rho)/2 \leq 1 - 3\gamma + 2\rho$ à chaque fois que $\rho \geq \frac{6\gamma-1}{5}$.) On peut facilement montrer que f_2 est décroissante sur $[\frac{6\gamma-1}{5}; 1]$ et donc se restreindre à l'étude de f_1. À nouveau, l'étude de sa dérivée nous permet d'affirmer que f_1 atteint son maximum sur $[0, \frac{6\gamma-1}{5}]$ pour λ qui est l'unique racine réelle de $(1-\lambda)(1-3\gamma+2\lambda)^2 - 27(\gamma-\lambda)^3$. Par les mêmes arguments que ceux utilisés dans la preuve du lemme 3.10, nous obtenons un facteur polynomial en n^4. □

En combinant les lemmes précédents, nous pouvons établir la borne suivante sur le temps d'exécution au pire des cas de notre algorithme :

Théorème 3.12. *Le temps d'exécution au pire des cas de l'algorithme **minCDS** est maximum pour :*

- *étape 1 : $\mathcal{O}\big(n\binom{n}{\gamma n}\big)$ (par le lemme 3.7) ;*
- *étape 2.1 : $\mathcal{O}\big(n^6\binom{n}{\frac{n-3\gamma n}{\beta-3}}\cdot\binom{n}{n-3\gamma n+3}\cdot\binom{(1-\lambda)n}{(2\lambda+1-3\gamma)n}\big)$ si une solution λ à $27(\gamma-\lambda)^3 = (1-\lambda)(1-3\gamma+2\lambda)^2$ existe sur $[0, \frac{1-3\gamma}{\beta-3}]$; sinon $\mathcal{O}^*\big(n^6\binom{n}{\frac{n-3\gamma n}{\beta-3}}\cdot\binom{n}{n-3\gamma n+3}\cdot\binom{(\frac{\beta-4+3\gamma}{\beta-3})n}{(\frac{3\gamma+\beta-3\gamma\beta-1}{\beta-3})n}\big)$ (par le lemme 3.8, le lemme 3.9 et le lemme 3.10) ;*
- *étape 2.2 : $\mathcal{O}\big(n^5\binom{n}{\frac{n-3\gamma n}{\beta-3}}\cdot\binom{(1-\lambda)n}{(1-3\gamma+2\lambda)n}\big)$ où λ est l'unique racine réelle de $(1-\lambda)(1-3\gamma+2\lambda)^2 - 27(\gamma-\lambda)^3$ sur $[0, \frac{6\gamma-1}{5}]$ (par le lemme 3.8 et le lemme 3.11).*

Temps d'exécution	degré du polynôme	γ	β
$O^*(1.8844^n)$	$n^5 \cdot n^4$	0.32914	4
$O^*(1.8798^n)$	$n^5 \cdot n^5$	0.32574	5
$O^*(1.8649^n)$	$n^5 \cdot n^{10}$	0.31520	10
$O^*(1.8573^n)$	$n^5 \cdot n^{15}$	0.31000	15
$O^*(1.8486^n)$	$n^5 \cdot n^{50}$	0.30424	50
$O^*(1.8463^n)$	$n^5 \cdot n^{40000}$	0.30275	40000

TABLE 3.2 – La table présente les temps d'exécution au pire des cas de l'algorithme **minCDS**, selon les valeurs de γ et β. Le degré du polynôme caché dans la notation $\mathcal{O}^*$ est donné par la seconde colonne.

Finalement, nous établissons le corollaire suivant :

Corollaire 3.13. *En utilisant $\gamma = 31/100$ et $\beta = 15$, l'algorithme* **minCDS** *s'exécute en temps $\mathcal{O}(n^{20}1.8573^n)$ et espace exponentiel.*

Un compromis entre les termes polynomiaux et exponentiels

Comme nous l'avons vu au théorème 3.12, le temps d'exécution de notre algorithme **minCDS** dépend essentiellement de deux paramètres : γ et β. Le paramètre β a une influence directe sur le degré du terme polynomial qui apparaît dans le temps d'exécution. Rappelons aussi que la taille de la famille $\mathcal{F}$ contribue également au temps d'exécution de l'algorithme de Koivisto (2009). Ainsi, en choisissant de façon adéquate les paramètres (par exemple, avec $\gamma = 0.30275$ et $\beta = 40000$), le théorème 3.12 montre que le temps d'exécution est $O^*(1.8463^n)$. Malheureusement, la notation $\mathcal{O}^*$ cache un *gros* terme polynomial de degré n^β. À la table 3.2 nous donnons des temps d'exécution que l'on peut atteindre avec notre algorithme pour plusieurs valeurs de γ et β.

3.3 Énumération des ensembles dominants minimaux

Depuis les années soixante, la littérature témoigne d'un intérêt croissant pour l'énumération d'objets dans les graphes. L'objectif est à la fois de répondre à des questions purement combinatoires mais aussi d'énumérer ces objets en vue d'une utilisation algorithmique pour résoudre d'autres problèmes. Pour spécifier des objets concrets ayant été étudiés, on peut citer les ensembles stables maximaux, les ensembles dominants minimaux (connexes), les chemins induits, les séparateurs minimaux, les cliques maximales potentielles, les coupes cycles minimaux, ou encore les transversaux dans les hypergraphes (Moon et Moser (1965); Fomin et Villanger (2010); Golovach *et al.* (2015); Fomin *et al.* (2008a, 2011b); Lonc et Truszczynski (2008)). Bien souvent, on souhaite une borne combinatoire sur le nombre maximum de ces objets énumérés. Nous pouvons par exemple citer le célèbre résultat de Moon et Moser (1965), qui établit une borne de $3^{n/3}$ sur le nombre maximum d'ensembles stables maximaux dans un graphe à n sommets.

Les techniques pour établir de telles bornes peuvent être *combinatoires* ou *algorithmiques*. C'est cette deuxième approche que nous allons employer : en construisant des algorithmes qui énumèrent certains objets et en analysant leur temps d'exécution au pire des cas, nous serons à la fois capables d'énumérer ces objets et d'établir des bornes combinatoires sur leur nombre maximum. Nous nous

sommes déjà intéressés à ce type de questions au chapitre précédent. En effet, au chapitre 2, nous avons présenté un algorithme pour énumérer des étiquetages $L(2,1)$. En particulier, la section 2.2.2 donne un algorithme qui énumère les étiquetages de largeur 5 dans les graphes cubiques en temps $\mathcal{O}(1.7990^n)$ (théorème 2.12) et a pour conséquence que le nombre de ces étiquetages est au plus $\mathcal{O}(1.7990^n)$ (corollaire 2.13). D'un autre côte, le théorème 2.14 montre que leur nombre est au moins $\Omega(2^{2n/7})$. Bien que ce soit la meilleure borne supérieure que nous ayons réussi à établir, celle-ci est très certainement sur-estimée. Nous l'avions établie grâce à un algorithme de *branchement*. Il est dès lors légitime de se demander si les algorithmes de *branchement* (et leurs analyses via l'étude de récurrences) sont les *bons* outils pour établir des bornes combinatoires serrées, c'est-à-dire, des bornes *atteintes* dans le pire des cas. Dans cette section, nous montrons qu'il est effectivement possible d'établir des bornes serrées via des algorithmes de *branchement*.

Dans la suite, nous allons nous intéresser au problème de l'énumération des ensembles dominants minimaux pour quelques classes de graphes. Étant donné un graphe $G = (V, E)$, un ensemble $D \subseteq V$ est un ensemble dominant minimal par l'inclusion si $N[D] = V$ et il n'existe pas de sous-ensemble $D' \subset D$ tel que $N[D'] = V$. Il est connu que les graphes ont $\mathcal{O}(1.7159^n)$ ensembles dominants minimaux qui peuvent être énumérés dans ce même temps (Fomin *et al.*, 2008b). Cette borne n'est probablement pas serrée et la meilleure borne inférieure connue est $\Omega(1.5705^n)$. Il est conjecturé que cette borne inférieure est effectivement aussi la borne supérieure. L'énumération de ces ensembles dominants est utile pour résoudre d'autres problèmes, comme par exemple NOMBRE DOMATIQUE (Fomin *et al.*, 2008b). Récemment, l'énumération des ensembles dominants minimaux a été étudiée pour plusieurs classes de graphes (essentiellement par Couturier *et al.* (2013b)). Dans cette section, nous montrons que pour les graphes splits et d'intervalles, leur nombre est au plus $3^{n/3} \approx 1.4423^n$, et que cette borne est la meilleure possible. Ces résultats répondent à une conjecture proposée par Couturier *et al.* (2012).

3.3.1 Définition du problème et résultats connus

Fomin *et al.* (2008b) ont montré que tout graphe a au plus $O(1.7159^n)$ ensembles dominants minimaux et qu'il existe des graphes avec au moins $O(1.5705^n)$ ensembles dominants minimaux. L'énumération des ensembles dominants minimaux a aussi été étudiée pour plusieurs classes de graphes et un résumé des connaissances actuelles (essentiellement de Couturier *et al.* (2012), Couturier *et al.* (2013b), Fomin *et al.* (2008b) et Krzywkowski (2013)) est donné à la table 3.3. Nous mentionnons également dans cette table nos résultats récents et dont certains (de la publication avec Couturier *et al.* (2015)) sont présentés dans la suite de ce chapitre. Pour établir ces bornes, nous concevons des algorithmes exponentiels pour énumérer les ensembles dominants puis nous analysons leur complexité au pire des cas. Pour être totalement complet, mentionnons que l'énumération des ensembles dominants minimaux a aussi été étudiée sous l'angle des algorithmes à délais polynomiaux et des algorithmes *output-polynomiaux* (voir les travaux de Kanté *et al.* (2011, 2012, 2013) et le résumé de Kanté et Nourine (2015)).

En particulier, nous démontrons dans la suite le conjecture suivante :

Conjecture (Couturier *et al.* (2013b))**.** *Le nombre maximum d'ensembles dominants minimaux dans un graphe d'intervalles propres et dans un graphe split est* $3^{n/3}$.

Plus précisément, nous montrons la conjecture pour les graphes splits et aussi pour les graphes d'intervalles qui forment une super-classe des graphes d'intervalles propres. Dans Couturier *et al.* (2015), nous améliorons également la borne supérieure de Couturier *et al.* (2012) sur le nombre maximum d'ensembles dominants minimaux pour les graphes cobipartis, de 1.5875^n

classes de graphes	bornes inf.	références	bornes sup.	références
général	$15^{n/6}$	Fomin *et al.* (2008b)	1.7159^n	Fomin *et al.* (2008b)
arbre	1.4167^n	Krzywkowski (2013)	1.4656^n	Krzywkowski (2013)
cordal	$3^{n/3}$	Couturier *et al.* (2013b)	1.6181^n	Couturier *et al.* (2013b)
cobiparti	1.3195^n	Couturier *et al.* (2012)	1.5875^n	Couturier *et al.* (2012)
			$n^2 + 2 \cdot 1.4511^n$	Couturier *et al.* (2015)
			1.3803^n	Golovach *et al.* (2015)
split*	$3^{n/3}$	Couturier *et al.* (2013b)	1.4656^n	Couturier *et al.* (2013b)
			$3^{n/3}$	Couturier *et al.* (2015)*
intervalles propres	$3^{n/3}$	Couturier *et al.* (2013b)	1.4656^n	Couturier *et al.* (2013b)
intervalles*	$3^{n/3}$	Couturier *et al.* (2013b)	$3^{n/3}$	Couturier *et al.* (2015)*
cographe	$15^{n/6}$	Couturier *et al.* (2013b)	$15^{n/6}$	Couturier *et al.* (2013b)
trivialement parfait	$3^{n/3}$	Couturier *et al.* (2013b)	$3^{n/3}$	Couturier *et al.* (2013b)
threshold	$\omega(G)$	Couturier *et al.* (2013b)	$\omega(G)$	Couturier *et al.* (2013b)
chain	$\lfloor n/2 \rfloor + m$	Couturier *et al.* (2013b)	$\lfloor n/2 \rfloor + m$	Couturier *et al.* (2013b)

TABLE 3.3 – Les bornes inférieures et supérieures connues pour le nombre maximum d'ensembles dominants minimaux. La borne inférieure $15^{n/6}$ pour le cas général est mentionnée dans (Fomin *et al.*, 2008b) et est due à D. Kratsch. Les deux résultats marqués avec un symbole $\star$ sont présentés dans ce chapitre. Notons que $3^{n/3} \approx 1.4423^n$ et $15^{n/6} \approx 1.5704^n$.

à $n^2 + 2 \cdot 1.4511^n$. La preuve est là encore basée sur un algorithme de *branchement* que nous ne détaillerons pas dans ce document. D'ailleurs, très récemment, Golovach *et al.* (2015) améliorent encore cette borne en montrant que tout graphe cobiparti admet au plus $\mathcal{O}(1.3803^n)$ ensembles dominants minimaux ; la meilleure borne inférieure est de $\Omega(1.3195^n)$ Couturier *et al.* (2012).

Notions préliminaires et caractérisation des ensembles dominants minimaux.

Rappelons quelques définitions. Soit $G = (V, E)$ un graphe. Étant donné un sommet v, son voisinage est noté $N(v)$ et son voisinage fermé est défini par $N[v] = N(v) \cup \{v\}$. Cette définition est étendue aux sous-ensembles de sommets de la façon suivante : pour $D \subseteq V$, l'ensemble $N[D]$ est défini par $\cup_{v \in D} N[v]$. L'ensemble D est *dominant* si $N[D] = V$. Étant donné un sommet v et un ensemble X, nous définissons $N_X(v) = N(v) \cap X$ et $d_X(v) = |N_X(v)|$. Pour un ensemble D, un sommet $x \in D$ et un sommet $v \in V$ (où éventuellement $x = v$), nous disons que v est un *voisin privé* de x si $v \notin N[D \setminus \{x\}]$.

Nous pouvons observer que si un ensemble dominant D contient un sommet $x \in D$ sans voisin privé, alors l'ensemble $D \setminus \{x\}$ est également dominant. Inversement, il est facile d'observer que si chaque sommet d'un ensemble dominant D a un voisin privé, alors D est un ensemble dominant minimal (par l'inclusion). On établit ainsi la propriété suivante qui nous sera utile tout au long du reste du chapitre :

Proposition 3.14 (notée $(\star)$ dans la suite)**.** *Un ensemble dominant D est minimal si et seulement si chaque sommet de D a un voisin privé.*

Quelques mots sur les classes de graphes.

Nous considérons deux classes de graphes : les graphes splits et les graphes d'intervalles. Nous

rappelons brièvement leur définition ici et nous renvoyons le lecteur aux ouvrages de Brandstädt *et al.* (1999) et de Golumbic (2004) pour davantage de propriétés concernant les classes de graphes.

Un graphe $G = (V, E)$ est un *graphe split* (respectivement, *cobiparti*) si son ensemble de sommets V peut être partitionné en deux ensembles C et I, où C est une clique et I est un ensemble stable (respectivement, C et I sont deux cliques). Un graphe $G = (V, E)$ est un *graphe d'intervalles* si les sommets de V peuvent être mis en bijection avec des intervalles de la droite réelle, tels que deux sommets soient adjacents si et seulement si les intervalles correspondants s'intersectent. Pour faciliter les notations, étant donné un sommet v, nous notons $l(v)$ (respectivement $r(v)$) la valeur de l'extrémité gauche (respectivement, l'extrémité droite) de l'intervalle de la droite réelle correspondant à v. Nous travaillerons avec des intervalles représentés par leur modèle normalisé : un modèle d'intervalles d'un graphe G est *normalisé* si $\cup_{v \in V}\{l(v), r(v)\} = \{1, 2, ..., 2n\}$.

3.3.2 Graphes splits

Dans cette section, nous montrons que les graphes splits admettent au plus $3^{n/3}$ ensembles dominants minimaux, qui peuvent être énumérés en temps $O^*(3^{n/3})$.

Pour énumérer ces ensembles dominants minimaux, nous proposons un algorithme de *branchement*. Il est composé d'une collection de règles (de branchement) qui sont décrites ci-après. L'algorithme est récursif et, à chaque appel, il cherche à appliquer chacune de ces règles successivement. Cela signifie que lorsqu'une règle est appliquée, aucune règle n'apparaissant avant celle-ci ne s'applique. Comme nous le verrons, cela pourra garantir une certaine structure au graphe lors de l'utilisation d'une règle.

Nous notons DS l'ensemble dominant minimal courant (qui est initialement vide). Soit v un sommet de la clique C ayant un nombre maximum de voisin dans l'ensemble stable I

Description de l'algorithme

Cas 1. Si $d_I(v) \geq 3$ alors nous branchons selon :

- « Ajouter v à DS ». Au moins 4 sommets (v et $N_I(v)$) sont supprimés.
- « Exclure v ». Dès qu'un voisin de v sera dominé, le sommet v le deviendra également ; on peut donc sans crainte éliminer v.

Cela donne un vecteur de branchement $(4, 1)$ dont la solution est $\tau(4, 1) < 1.3803$.

Cas 2. Si $d_I(v) = 2$ alors nous posons $N_I(v) = \{x, y\}$. Sans perte de généralité, supposons que $d(x) \leq d(y)$.

Cas 2.1. Si $d(x) = 1$ alors v est l'unique voisin de x et il y a deux possibilités pour dominer x :

- « Ajouter x à DS ». Par minimalité de DS, le sommet v ne peut pas appartenir à l'ensemble dominant, car sinon x n'aurait pas de voisin privé ; donc x et v peuvent être supprimés.
- « Exclure x ». Pour dominer x, nous devons ajouter v à DS. Le sommet y est supprimé car il est aussi dominé par v, et par minimalité de DS il ne peut lui-même pas appartenir à DS (rappelons que C est une clique et que $v \in C$).

Cela nous donne un vecteur de branchement $(2, 3)$ dont la solution est $\tau(2, 3) < 1.3248$.

Cas 2.2. Si $d(x) = 2$ alors on note v' l'autre voisin de x.

Cas 2.2.1. Si $d_I(v') = 1$ alors x est l'unique voisin de v' dans I, et il y a trois possibilités pour dominer x :

- « Ajouter v à DS ». Comme x est l'unique voisin de v' dans I, il n'y a pas d'autre ensemble dominant minimal contenant v'. Ainsi, au moins 4 sommets (v, $\{x, y\} \subseteq N_I(v)$ et v') sont supprimés.
- « Ajouter v' à DS ». Si v n'appartient pas à l'ensemble dominant, nous pouvons utiliser v' pour dominer x. Dans ce cas, nous supprimons x, v et v'. Notons que le cas « Ajouter v à DS » a déjà été considéré précédemment, donc on sait que v ne peut plus être ajouté à DS (et v est dominé par v').
- « Ajouter x à DS ». Si ni v ni v' n'appartiennent à l'ensemble dominant, alors nous devons placer x dans DS. Encore une fois, nous supprimons v, v' et x.

Nous obtenons $(4, 3, 3)$ comme vecteur de branchement, dont la solution est $\tau(4, 3, 3) < 1.3954$.

Cas 2.2.2 Si $d_I(v') \geq 2$ et $y \notin N(v')$ alors on observe que $d_I(v') = 2$ (puisque $d_I(v) = 2$ et $d_I(v)$ est maximum sur tous les sommets $v \in C$). Notons z l'autre voisin de v' dans I.

- « Ajouter v à DS et soit x son voisin privé ». Nous pouvons supprimer v, v' et $N_I(v)$, c'est-à-dire au moins 4 sommets.
- « Ajouter v à DS et soit y son voisin privé ». Puisque x n'est pas le voisin privé de v, il s'ensuit que v' appartient à DS. Ainsi, au moins 6 sommets sont supprimés (v, x, y, v', z et $N(y)$).
- « Exclure v et ajouter v' à DS ». Dans ce cas, x et z sont dominés par v' et peuvent être supprimés, tout comme v et v'.
- « Ajouter x à DS ». Puisque ni v, ni v' n'appartiennent à DS, x est ajouté à DS et $N(x)$ est supprimé.

On obtient une récurrence dont le vecteur de branchement n'est pas pire que le vecteur $(4, 6, 4, 3)$ qui donne $\tau(4, 6, 4, 3) < 1.4064$.

Cas 2.2.3. Si $d_I(v') \geq 2$ et $y \in N(v')$ alors $N[v'] = N[v]$.

- « Ajouter v à DS ». Par minimalité, v' ne peut jamais être ajouté à DS. Nous supprimons donc v, v', x et y.
- « Ajouter v' à DS ». Ce branchement est symétrique au précédent.
- « Ajouter x à DS ». Comme v et v' n'appartiennent pas à DS, x appartient à DS et $N[x]$ est supprimé.

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

Cas 2.3. Si $d(x) = 3$ alors on note v, v'_1 et v'_2 les voisins de x. Sans perte de généralité, supposons que $d(v'_1) \leq d(v'_2)$.

Cas 2.3.1. Si $y \notin N(v'_1) \cup N(v'_2)$ alors nous branchons de la façon suivante :

- « Ajouter v à DS et soit x son voisin privé ». Nous supprimons 5 sommets : v, $N_I(v)$ et $N(x)$.
- « Ajouter v à DS, soit y son voisin privé et $v'_1 \in DS$ ». Si $d_I(v'_1) = 1$ alors cette branche est éliminée puisqu'il n'est pas minimal d'ajouter à la fois v'_1 et v à DS. Nous supposons donc que $d_I(v'_1) \geq 2$. Dans ce cas, nous supprimons v, v'_1, x, y, $N(y)$ (puisque y est le voisin privé de v) et $N_I(v'_1)$; c'est-à-dire au moins 7 sommets.
- « Ajouter v à DS, soit y son voisin privé et $v'_2 \in DS$ ». À cause de la branche précédente, $v'_1 \notin DS$. À nouveau, nous pouvons supposer que $d_I(v'_2) \geq 2$ car sinon nous éliminons cette branche. Dans ce cas, nous supprimons v, v'_1, v'_2, x, y, $N(y)$, et $N_I(v'_2)$, c'est-à-dire au moins 8 sommets.
- « Exclure v ». Dans ce cas, nous ne supprimons que v.

Cela donne une récurrence dont le vecteur de branchement n'est pas pire que $(5, 7, 8, 1)$, qui donne $\tau(5, 7, 8, 1) < 1.4331$.

Cas 2.3.2. Si $y \in N(v'_1) \cup N(v'_2)$ alors on branche de la façon suivante :

- « Ajouter v à DS et soit x son voisin privé ». Nous supprimons 5 sommets : v, $N_I(v)$ et $N(x)$.
- « Ajouter v à DS, soit y son voisin privé et $v'_i \in DS$ ». Si $y \in N(v'_1) \cap N(v'_2)$ cette branche est éliminée puisque y ne peut être un voisin privé de v ; sinon, nous posons i tel que $y \notin N(v'_i)$. Si $d_I(v'_i) = 1$ alors cette branche est éliminée puisqu'il n'est pas minimal d'ajouter à la fois v'_1 et v à DS ; sinon on supprime v, v'_i, x, y, $N(y)$ (puisque y est le voisin privé de v) et $N_I(v'_i)$; c'est-à-dire au moins 7 sommets.
- « Exclure v ». Dans ce cas, nous ne supprimons que v.

Cela nous donne une récurrence dont le vecteur de branchement n'est pas pire que $(5, 7, 1)$, ou $(5, 1)$ dans le cas où $y \in N(v'_1) \cap N(v'_2)$; ce qui nous donne $\tau(5, 7, 1) = 1.3971$ ou $\tau(5, 1) < 1.3248$ respectivement.

Cas 2.4. Si $d(x) \geq 4$ alors nous effectuons le branchement suivant :

- « Ajouter v à DS et soit x son voisin privé ». On supprime v, x, y et $N(x)$, donc au moins 6 sommets.
- « Ajouter v à DS et soit y son voisin privé ». On supprime v, x, y et $N(y)$, donc au moins 6 sommets.
- « Exclure v ». On ne supprime que v.

Cela donne un vecteur de branchement $(6, 6, 1)$, où $\tau(6, 6, 1) < 1.3881$. Observons que les ensembles dominants minimaux où x et y ne sont que dominés par v sont générés à la fois par la première branche et par la deuxième. Ainsi, certains ensembles dominants minimaux peuvent être produits plus d'une fois par notre algorithme.

Cas 3. Puisque les cas précédents ne peuvent être appliqués, on en déduit que pour chaque sommet $v \in C$, on a $d_I(v) \leq 1$. Prenons un sommet $x \in I$ de degré maximum et notons $v_1, \ldots, v_{d(x)}$ ses voisins. On observe que dès que x est dominé, tous ses voisins peuvent être immédiatement supprimés. Cela suggère le branchement suivant :

- « Ajouter x à DS » et supprimer $N[x]$.
- « Exclure x et ajouter v_i à DS » pour chaque i, $1 \leq i \leq d(x)$. Nous avons $d(x)$ possibilités pour choisir un voisin de x et l'ajouter à DS. Dans chacun des cas, nous supprimons $N[x]$.

Cela donne le vecteur de branchement $(d(x) + 1, d(x) + 1, \ldots, d(x) + 1)$ de longueur $d(x) + 1$ et son pire des cas est obtenu pour $d(x) = 2$ avec $\tau(d(x) + 1, d(x) + 1, \ldots, d(x) + 1) \leq 3^{1/3} < 1.4423$.

Cas 4. Finalement, si le graphe ne contient plus aucune arête, alors les sommets restants dans I sont ajoutés à DS. Si $DS \cap C$ est non vide, alors les sommets restants de C sont supprimés et $DS \cup I$ est retourné ; sinon, pour chaque sommet v de C, l'ensemble $DS \cup I \cup \{v\}$ est retourné. Puisque $|C| \leq n \leq 3^{n/3}$, le nombre d'ensembles dominants du type $DS \cup I \cup \{v\}$ satisfait la borne.

La correction de l'algorithme est facile puisque les règles de branchement considèrent toutes les possibilités pour dominer un sommet, et pour ajouter ou exclure un sommet de l'ensemble dominant. Pour chacune des règles de branchement, le vecteur de branchement et son facteur de branchement correspondant ont été donnés. La valeur maximale sur tous ces facteurs de branchement nous permet d'établir le théorème 3.15. On conclut la preuve en observant que l'algorithme ne nécessite qu'un espace polynomial. On obtient ainsi la borne combinatoire sur le nombre d'ensembles dominants minimaux énumérés par l'analyse du temps d'exécution de notre algorithme.

Théorème 3.15. *Les graphes splits ont au plus $3^{n/3}$ ensembles dominants minimaux qui peuvent être énumérés en temps $O^*(3^{n/3})$ et espace polynomial.*

Notre borne supérieure est la meilleure possible comme l'atteste la preuve du théorème suivant :

Théorème 3.16 (Couturier *et al.* (2013b))**.** *Il existe des graphes splits avec au moins $3^{n/3}$ ensembles dominants minimaux.*

Démonstration. La construction proposée par Couturier *et al.* (2013b) est représentée à la figure 3.3. Il s'agit d'une collection de $n/3$ triangles. La clique C du graphe split est obtenue en prenant, dans chaque triangle, deux sommets parmi ses trois sommets. On observe que tout ensemble dominant minimal contient précisément l'un des sommets de chaque triangle. □

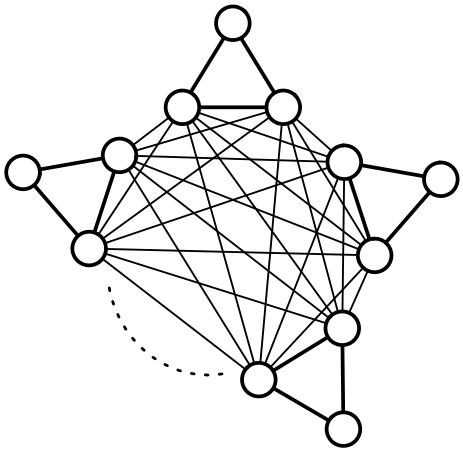


FIGURE 3.3 – Un graphe split ayant $3^{n/3}$ ensembles dominants minimaux.

3.3.3 Graphes d'intervalles

Les graphes d'intervalles sont des graphes pour lesquels chaque sommet peut être mis en bijection avec un intervalle de la droite réelle, de sorte que deux sommets soient adjacents si et seulement si les intervalles correspondants ont une intersection non vide. Si on peut trouver une représentation telle qu'il n'y ait pas d'intervalle strictement inclus dans un autre, alors le graphe est un graphe d'intervalles propres.

Couturier *et al.* (2013b) ont montré que les graphes d'intervalles propres ont au plus 1.4656^n ensembles dominants minimaux. Nous améliorons à présent cette borne et montrons que les graphes d'intervalles (non nécessairement propres) admettent au plus $3^{n/3} \approx 1.4423^n$ ensembles dominants minimaux. Ce résultat répond à une question ouverte de Couturier *et al.* (2012). Pour établir notre résultat, nous proposons un algorithme qui énumère tous les ensembles dominants minimaux en temps $O^*(3^{n/3})$. Puisqu'il existe des graphes d'intervalles (propres) ayant $3^{n/3}$ ensembles dominants minimaux, notre résultat est le meilleur possible.

Théorème 3.17 (Couturier *et al.* (2013b))**.** *Il existe des graphes d'intervalles avec au moins $3^{n/3}$ ensembles dominants minimaux.*

Démonstration. Pour démontrer le théorème, il est suffisant de construire un graphe ayant $3^{n/3}$ ensembles dominants minimaux. Considérons le graphe de la figure 3.4. Il est composé d'une collection disjointe de $\frac{n}{3}$ triangles et un modèle d'intervalles propres est proposé sur cette même figure. On observe que tout ensemble dominant minimal doit contenir précisément un sommet de chaque triangle. On établit ainsi que ce graphe admet $3^{n/3}$ ensembles dominants minimaux. □

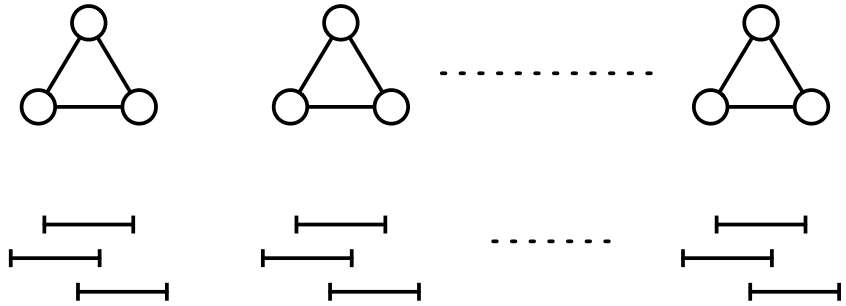


FIGURE 3.4 – Un graphe d'intervalles (propres) ayant $3^{n/3}$ ensembles dominants minimaux.

Une question que l'on se pose naturellement concernant cette borne inférieure est de savoir si elle est toujours valable lorsqu'on impose que le graphe d'intervalles propres soit connexe. L'ajout d'un sommet universel, souvent utilisé comme astuce, ne convient pas : le modèle n'est pas propre. Néanmoins, on peut considérer la construction de la figure 3.5 qui montre que la borne inférieure de $3^{n/3}$ ensembles dominants minimaux est encore vraie pour une collection d'intervalles propres correspondant à un graphe connexe. On observe facilement que tout ensemble dominant doit contenir un sommet (noté x_i) de chaque triangle $\{a_i, b_i, c_i\}$ $(1 \leq i \leq n/3)$ car le sommet a_i doit être dominé par l'un des trois sommets de $N[a_i]$. De plus, le sommet x_i possède le sommet a_i comme voisin privé. L'ensemble est donc bien minimal par la propriété $(\star)$. Nous établissons ainsi le théorème 3.18.

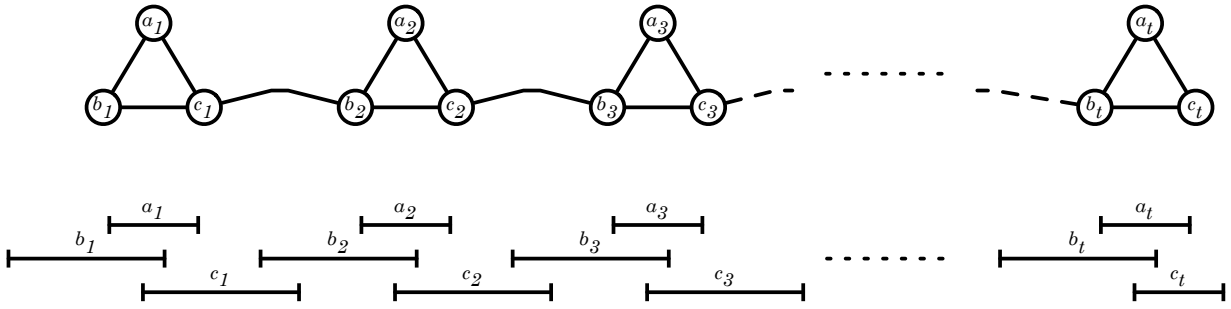


FIGURE 3.5 – Un graphe d'intervalles (propres) connexe ayant $3^{n/3}$ ensembles dominant minimaux, où $t = n/3$.

Théorème 3.18. *Il existe des graphes d'intervalles propres connexes avec au moins $3^{n/3}$ ensembles dominants minimaux.*

Supposons qu'un modèle d'intervalles normalisé $\mathcal{I}$ d'un graphe $G = (V, E)$ fasse partie de l'entrée ; c'est-à-dire tel que $\cup_{v \in V}\{l(v), r(v)\} = \{1, 2, ..., 2n\}$. L'algorithme **EnumDomSetInterval** est donné ci-après. Nous montrons dans la suite qu'il énumère effectivement tous les ensembles dominants minimaux d'un graphe d'intervalles G donné en entrée.

Algorithme 10 : EnumDomSetInterval$\big(G = (V, E), \mathcal{I}\big)$

Entrée : Un graphe d'intervalles G et son modèle normalisé $\mathcal{I}$.
Sortie : Tous les ensembles dominants minimaux de G.

Initialisation :
Soit y l'intervalle de $\mathcal{I}$ avec la plus petite extrémité droite
$\mathcal{L} \leftarrow \varnothing$
pour chaque $z \in N[y]$ **faire**
  Ajouter le triplet $\Big\{y, z, \{z\}\Big\}$ à $\mathcal{L}$
tant que $\mathcal{L}$ *n'est pas vide* **faire**
  Extraire un triplet $\{y, z, DS\}$ de $\mathcal{L}$
  Soit y' le sommet tel que $r(y') = \min_{x \in \mathcal{I}}\{r(x)$ t.q. $l(x) > r(z)\}$
  si *il n'y a pas de tel sommet* y' **alors**
    retourner « DS est un ensemble dominant minimal »
  sinon si $N[y'] \setminus N[y]$ *est vide* **alors**
    Oublier le triplet $\{y, z, DS\}$
  sinon
    pour chaque $z' \in (N[y'] \setminus N[y])$ **faire**
      Ajouter le triplet $\Big\{y', z', DS \cup \{z'\}\Big\}$ à $\mathcal{L}$

Pour simplifier les notations dans la suite de la preuve, nous ne distinguons plus un sommet de son intervalle correspondant dans le modèle $\mathcal{I}$. Ainsi, lorsque nous parlons de l'extrémité gauche ou droite d'un sommet, nous désignons l'extrémité gauche ou droite de l'intervalle correspondant.

Commençons par définir les notions de *graphes partiels* et de *bons triplets* :

Définition 3.2. Étant donné un sommet $z \in V$, nous définissons le *graphe partiel* noté $G_{\leq z}$, comme le sous-graphe induit $G[S]$ où S est l'ensemble des sommets dont l'extrémité gauche est plus petite ou égale à $r(z)$.

Définition 3.3. Pour deux sommets $y, z \in V$ et un sous-ensemble $DS \subseteq V$, nous disons que $\{y, z, DS\}$ est un *bon triplet* si :

1. z est le sommet de DS dont l'extrémité droite est la plus grande possible ;
2. DS est un ensemble dominant minimal du graphe partiel $G_{\leq z}$;
3. y est le voisin privé de z dans $G_{\leq z}$, par rapport à l'ensemble DS, avec la plus petite extrémité droite.

L'intuition est que, pour un bon triplet $\{y, z, DS\}$, le sommet y est utilisé par notre algorithme afin d'assurer que la propriété de minimalité (de l'ensemble dominant) soit bien préservée. Cela garantit que le sommet z est nécessaire pour dominer le graphe et que y est l'un de ses voisins privés. Puisque z pourrait avoir plusieurs voisins privés (y compris lui-même), nous retenons seulement celui avec la plus petite extrémité droite.

Les prochains lemmes assurent la correction de l'algorithme **EnumDomSetInterval**.

Lemme 3.19. *Soit y l'intervalle de $\mathcal{I}$ tel que $r(y)$ soit minimum. Dans tout ensemble dominant minimal DS, le sommet y est le voisin privé d'un certain sommet de DS.*

Démonstration. Soit $DS_y = N[y] \cap DS$. Puisque DS est un ensemble dominant, l'ensemble DS_y est non vide. Si $DS_y = \{z\}$, alors y est un voisin privé de z (avec éventuellement $z = y$). Supposons maintenant que $|DS_y| \geq 2$. Soit z, z' deux sommets appartenant à DS_y. Sans perte de généralité, supposons que $r(z') < r(z)$. Puisque y est l'intervalle de $\mathcal{I}$ ayant la plus petite extrémité droite, alors $N[z'] \subseteq N[z]$. Ceci contredit le fait que DS soit un ensemble dominant minimal, puisque l'ensemble $DS \setminus \{z'\}$ est aussi dominant. □

Lemme 3.20. *Soit $\{y, z, DS\}$ un bon triplet. Soit y' le sommet tel que $r(y') = \min_{x \in \mathcal{I}}\{r(x)$ t.q. $l(x) > r(z)\}$. Supposons qu'un tel sommet y' existe. Dans tout ensemble dominant minimal DS' tel que $DS \subseteq DS'$, le sommet y' est un voisin privé d'un certain sommet de DS'.*

Démonstration. Supposons que y' existe et ne soit pas le voisin privé d'un sommet de DS'. Alors l'ensemble $DS'_{y'} = N[y'] \cap DS'$ contient au moins deux sommets, appelés z'_1 et z'_2. Par définition de y' et de DS, ces deux sommets z'_1, z'_2 appartiennent à $DS' \setminus DS$. Sans perte de généralité, supposons que $r(z'_1) < r(z'_2)$. Comme y' est le sommet non dominé par DS ayant la plus petite extrémité droite, par le lemme 3.19 nous établissons que l'ensemble $DS' \setminus \{z'_1\}$ est également dominant. Ceci contredit le fait que DS' soit un ensemble dominant minimal et donc $|DS'_{y'}| = 1$. □

Le prochain lemme montre que certains triplets ne peuvent être étendus en des ensembles dominants minimaux du graphe en entier, et donc qu'ils peuvent être « oubliés ».

Lemme 3.21. *Soit $\{y, z, DS\}$ un bon triplet. Soit y' le sommet tel que $r(y') = \min_{x \in \mathcal{I}}\{r(x)$ t.q. $l(x) > r(z)\}$. 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 dominer à lui seul tout le graphe (y' est non dominé par DS). L'ensemble $N[y']$ est trivialement non vide (il contient au moins y') mais puisque l'ensemble $N[y'] \setminus N[y]$ est vide, chaque sommet de $N[y']$ est également un voisin de y. Ainsi, tout ensemble dominant DS' tel que$DS \subseteq DS'$ doit contenir z (qui appartient à DS) et un voisin commun à y' et à y, noté z' (c'est-à-dire $z' \in N[y']$). Puisque y est le voisin privé de z ayant la plus petite extrémité droite, on établit que $DS' \setminus \{z\}$ est aussi un ensemble dominant de $G_{\leq z'}$. Il est donc impossible d'étendre DS en un ensemble dominant minimal. □

L'idée générale de l'algorithme **EnumDomSetInterval** est de trouver (de façon itérative) un sommet non dominé y' qui pourrait être utilisé comme voisin privé d'un certain sommet z' de son voisinage. Ensuite, l'algorithme « essaye » tous les candidats z' possibles qui pourraient appartenir à des ensembles dominants minimaux. Le prochain lemme montre que tous les triplets ajoutés par l'algorithme **EnumDomSetInterval** à la liste $\mathcal{L}$ sont de bons triplets et correspondent à des ensembles dominants minimaux de graphes partiels.

Lemme 3.22. *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 $\mathcal{L}$. Regardons comment un tel triplet $\{y, z, DS\}$ a été ajouté à $\mathcal{L}$. Soit il a été obtenu par extension d'un $\{\tilde{y}, \tilde{z}, \widetilde{DS}\}$ dans une itération de la boucle « tant que », soit il a été ajouté à $\mathcal{L}$ durant l'étape d'initialisation (c'est-à-dire avant la boucle « tant que »).

Par le lemme 3.19, le sommet y avec la plus petite extrémité droite est nécessairement le voisin privé d'un certain sommet $z \in N[y]$. Donc chaque triplet $\{y, z, DS\}$, où $DS = \{z\}$, ajouté à l'étape d'initialisation est clairement un bon triplet.

Supposons maintenant que $\{y, z, DS\}$ ait été obtenu d'un certain triplet $\{\tilde{y}, \tilde{z}, \widetilde{DS}\}$ dans la boucle « tant que ». Par le lemme 3.20, le sommet y non dominé par $\widetilde{DS}$ et ayant la plus petite extrémité droite, est le voisin privé (ayant la plus petite extrémité droite) d'un certain sommet $z \in N[y]$. Puisque $\tilde{y}$ est le voisin privé de $\tilde{z}$, le sommet z est choisi parmi $N[y] \setminus N[\tilde{y}]$. Ainsi, le sommet z est le sommet de DS ayant la plus grande extrémité droite. Puisque $\widetilde{DS}$ est un ensemble dominant minimal de $G_{\leq \tilde{z}}$, nous établissons que $DS = \widetilde{DS} \cup \{z\}$ est un ensemble dominant minimal de $G_{\leq z}$.

Finalement, on observe que les triplets ajoutés à la liste $\mathcal{L}$ durant l'étape d'initialisation sont également des bons triplets ; la propriété est donc être établie par induction. □

Le prochain lemme montre que si des bons triplets sont produits par l'algorithme, tels que tous les sommets du graphe donné soient dominés, alors un ensemble dominant minimal du graphe donné en entrée a été construit (et est retourné par l'algorithme). Ainsi, tous les ensembles retournés par l'algorithme sont *des* ensembles dominants du graphe. Les lemmes 3.24 et 3.25 montrent que *tous* les ensembles dominants minimaux du graphe sont en fait retournés par l'algorithme.

Lemme 3.23. *Soit $\{y, z, DS\}$ un bon triplet de $\mathcal{L}$ tel qu'il n'y ait aucun sommet dont l'extrémité gauche 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 $\{y, z, DS\}$ n'admet pas de sommet y' tel que $r(y') = \min_{x \in \mathcal{I}}\{r(x)$ tel que $l(x) > r(z)\}$, alors tous les sommets sont dominés par DS. Par définition des bons triplets, DS est un ensemble dominant minimal du graphe. □

Lemme 3.24. *Soit DS un ensemble dominant minimal de G. Alors il existe deux sommets y et z tels que $\{y, z, DS\}$ soit un bon triplet.*

Démonstration. Considérons un ensemble dominant minimal DS. Soit z le sommet de DS ayant la plus grande extrémité droite. La propriété $\star$ appliquée à l'ensemble dominant minimal DS assure que z a au moins un voisin privé. Soit y son voisin privé (par rapport à DS) ayant la plus petite extrémité droite. Alors on observe que $\{y, z, DS\}$ est un bon triplet. □

Finalement, étant donné un graphe en entrée, nous montrons le résultat :

Lemme 3.25. *Soit $\{y, z, DS\}$ un bon triplet d'un graphe G donné. Alors, à une certaine étape de l'algorithme, le triplet $\{y, z, DS\}$ appartient à $\mathcal{L}$.*

Démonstration. Nous montrons par induction sur la taille de l'ensemble DS 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 d'un graphe G donné en entrée.

Clairement, si $DS = \{z\}$ alors le bon triplet $\{y, z, DS\}$ est produit durant l'étape d'initialisation, à cause de la première boucle « pour » de l'algorithme. Le sommet y ayant la plus petite extrémité droite est choisi et tous les sommets $z \in N[y]$ sont alors considérés pour ajouter le triplet $\{y, z, \{z\}\}$ à $\mathcal{L}$.

Supposons que $|DS| \geq 2$ et que, par induction, tous les bons triplets possibles $\{\tilde{y}, \tilde{z}, \widetilde{DS}\}$ aient été générés par l'algorithme (et ajoutés à la liste $\mathcal{L}$ à une certaine étape) 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. Puisque $\{y, z, DS\}$ est un bon triplet, l'ensemble DS est un ensemble dominant minimal de $G_{\leq z}$ et, par la propriété $(\star)$, z' a un voisin privé. Donc DS' est un ensemble dominant minimal de $G_{\leq z'}$. Soit y' le voisin privé de z' ayant la plus petite extrémité droite. Alors $\{y', z', DS \setminus \{z\})$ est un bon triplet et $r(z') < l(y)$.

Par l'hypothèse d'induction, à une certaine étape de l'algorithme, ce bon triplet $\{y', z', DS \setminus \{z\}\}$ appartient à $\mathcal{L}$. À cause de la boucle « tant que », un tel triplet est extrait de $\mathcal{L}$ puis étendu au triplet $\{y, z, DS\}$, lorsque y est choisi comme étant le sommet non dominé par DS' (c'est-à-dire $l(y) > r(z')$) avec la plus petite extrémité droite, et z l'un de ses voisins. □

Par les lemmes précédents, nous avons montré que tous les bons triplets sont produits par notre algorithme et ceux correspondant à des ensembles dominants minimaux du graphe donné en entrée sont retournés en sortie. Le prochain lemme implique que le temps d'exécution au pire des cas de l'algorithme est borné par $O^*(3^{n/3})$, puisque l'algorithme **EnumDomSetInterval** passe un temps polynomial par bon triplet et que leur nombre est au plus $\frac{3}{2} \cdot 3^{n/3}$. De plus, puisque les ensembles dominants minimaux du graphe donné en entrée correspondent aux triplets (qui ne peuvent plus être étendus) retournés par l'algorithme, nous montrons que tout graphe d'intervalles admet au plus $3^{n/3}$ ensembles dominants minimaux.

Lemme 3.26. *Tout graphe d'intervalles a au plus $\frac{3}{2} \cdot 3^{n/3}$ bons triplets.*

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

L'algorithme **EnumDomSetInterval** étend le triplet $\{y, z, DS\}$ en fixant d'abord un sommet y' (tel que $r(y') = \min_{x \in \mathcal{I}}\{r(x) \text{ t.q. } l(x) > r(z)\}$), puis en essayant d'ajouter à DS tous ses voisins z' (formellement, $z' \in N[y'] \setminus N[y]$). Par le lemme 3.20, y' est le voisin privé de z' et aucun autre voisin de y' (excepté z') ne peut ensuite être ajouté à DS.

Notons $L(k)$, où $k = |R|$, le nombre de telles séquences d'extensions possibles (y compris la séquence *vide* sans aucune extension). Ainsi, $L(k)$ représente le nombre de bons triplets qui peuvent être obtenus de $\{y, z, DS\}$ (incluant le triplet $\{y, z, DS\}$ lui-même).

Rappelons que z' doit être choisi dans $N[y'] \setminus N[y] = N[y'] \cap R$. Soit $d = |N[y'] \cap R|$. Le nombre maximal de bons triplets est donné par la récurrence $L(k) \leq 1 + d \cdot L(k - d)$ (c'est-à-dire le triplet courant plus tous ceux pouvant être obtenus par des extensions). Par des calculs standards (voir aussi la monographie de Fomin et Kratsch (2010)), nous établissons que

$$\begin{aligned} L(k) &\leq 1 + d + d^2 + d^3 + \cdots + d^{k/d} \\ &= \frac{1 - d^{k/d+1}}{1 - d} \\ &= \mathcal{O}^*(d^{k/d}). \end{aligned}$$

Autrement dit, comme la fonction $f(d) = d^{1/d}$ est maximale pour l'entier $d = 3$, on obtient $L(k) = \mathcal{O}(3^{k/3})$. Pour s'affranchir de la notation $\mathcal{O}(\cdot)$, on observe facilement que pour $d = 3$, $\frac{1-d^{k/d+1}}{1-d} \leq \frac{3}{2} \cdot 3^{k/3} - \frac{1}{2}$. Puisque $k \leq n$, nous obtenons que le nombre maximum de bons triplets dans un graphe d'intervalles est au plus $\frac{3}{2} \cdot 3^{n/3}$. □

Corollaire 3.27. *Tout graphe d'intervalles admet au plus $3^{n/3}$ ensembles dominants minimaux.*

Démonstration. La preuve du lemme 3.26 établit une borne sur le nombre de bons triplets. Il est suffisant d'observer que les ensembles dominants minimaux correspondent à certains triplets : ceux qui ne peuvent plus être entendus. Notons $T(k)$ le nombre maximum d'ensembles dominants minimaux retournés par l'algorithme **EnumDomSetInterval**. En utilisant des arguments similaires à ceux de la preuve du lemme 3.26, $T(k)$ satisfait la récurrence $T(k) \leq d \cdot T(k - d)$, donc $T(k) = d^{k/d}$ et donc $T(k) \leq 3^{n/3}$ (puisque la fonction $f(d) = d^{1/d}$ est maximum pour l'entier $d = 3$). □

Au théorème 3.18, il est montré qu'il existe des graphes d'intervalles (y compris *propres*) admettant au moins $3^{n/3}$ ensembles dominants minimaux. Par conséquent, le corollaire 3.27 et le théorème suivant sont optimaux.

Théorème 3.28. *L'algorithme* ***EnumDomSetInterval*** *énumère tous les ensembles dominants minimaux d'un graphe d'intervalles en temps* $O^*(3^{n/3})$.

Démonstration. Les lemmes 3.23 et 3.24 assurent que l'algorithme **EnumDomSetInterval** retourne effectivement tous les ensembles dominants minimaux. Par le lemme 3.26, nous savons que tout graphe d'intervalles admet au plus $O^*(3^{n/3})$ bons triplets, qui peuvent être énumérés par notre algorithme en temps $O^*(3^{n/3})$. □

Nous concluons cette section avec cette remarque :

Remarque 9. *L'algorithme* ***EnumDomSetInterval*** *tel que décrit par l'algorithme 10 peut nécessiter un espace exponentiel pour stocker la liste* $\mathcal{L}$*. Il est facile de modifier cet algorithme pour qu'il n'utilise qu'un espace polynomial. Pour cela, la liste* $\mathcal{L}$ *de bons triplets peut être implémentée par une pile (c'est-à-dire une structure de données* Last In First Out*). Ainsi, seulement un nombre polynomial de bons triplets est stocké dans* $\mathcal{L}$*.*

3.4 Conclusion

Nous avons vu que pour la résolution de DOMINATION AVEC CAPACITÉS, il ne semble pas du tout évident de combiner les deux approches existantes. En effet, comme nous l'avons expliqué sur l'exemple de la section 3.2.1, si on commence par deviner l'ensemble des représentants `(GR) ∪ (MR)`, puis qu'on utilise un couplage maximum pour résoudre le problème résiduel, il n'est pas possible de combiner ensuite avec l'approche *programmation dynamique* pour PARTITION EN SOUS-ENSEMBLES DE CARDINALITÉS BORNÉES sans dégrader fortement le temps d'exécution. Néanmoins, nous avons montré que les instances pour lesquelles, dans la solution optimale, l'ensemble `(GR) ∪ (MR)` contient un nombre significatif de sommets, disposent de suffisamment de structures pour développer une approche de *programmation dynamique* plus rapide. En particulier, nous avons étendu l'approche de Koivisto (2009) pour obtenir un algorithme en $\mathcal{O}^*(1.8463^n)$, plus rapide que celui de Cygan *et al.* (2011) pour le problème DOMINATION AVEC CAPACITÉS. De façon curieuse, notre algorithme offre un compromis entre ses facteurs polynomiaux et exponentiels. Selon deux paramètres choisis, son temps d'exécution varie entre $\mathcal{O}(n^9 \cdot 1.8844^n)$ et $\mathcal{O}(n^{40005} \cdot 1.8463^n)$.
Nous soulignons quelques questions intéressantes :

- Est-ce que notre algorithme pourrait être modifié pour dénombrer les ensembles dominants avec capacités ? Cette question est naturelle puisque l'algorithme de Koivisto (2009) peut dénombrer les partitions optimales.
- Est-ce que, avec quelques modifications, notre algorithme pourrait résoudre une version pondérée (c'est-à-dire où chaque sommet aurait un poids, voire même que le poids dépende de la capacité du sommet) ?
- Un tel compromis entre le terme exponentiel et le terme polynomial peut-il être obtenu pour d'autres problèmes ? En particulier, pourrait-on imaginer que pour tout problème, on puisse arbitrairement diminuer la base $c > 1$ de l'exponentielle au prix d'un grand degré d dans le temps d'exécution $\mathcal{O}(c^n \cdot n^d)$?

En deuxième partie de chapitre, nous nous sommes intéressés à l'énumération des ensembles dominants minimaux pour deux classes de graphes : les graphes splits et les graphes d'intervalles. Pour ces deux classes, nous avons démontré la borne optimale de $3^{n/3}$ ensembles dominants minimaux et avons donné des algorithmes en temps $\mathcal{O}^*(3^{n/3})$ pour les énumérer. Dans (Couturier *et al.*, 2015), nous montrons une borne de 1.4511^n pour les graphes cobipartis. Cette borne a été depuis réduite à 1.3803^n par Golovach *et al.* (2015). En combinant cette borne avec la borne inférieure de Couturier *et al.* (2012), nous savons donc que le nombre d'ensembles dominants minimaux dans ces graphes est α^n avec $1.3195^n \leq \alpha^n \leq 1.3803^n$. La borne supérieure est certainement encore sur-estimée et il serait intéressant d'établir une borne supérieure plus ajustée. Quelques indices nous laissent penser qu'il est difficile d'obtenir une borne optimale, car l'énumération des ensembles dominants minimaux dans les cobipartis généralise d'autres problèmes, comme l'énumération des transversaux dans les hypergraphes où là aussi la borne optimale reste encore à établir (voir par exemple les travaux de Lonc et Truszczynski (2008) qui donnent une borne supérieure non optimale). Pour les graphes cordaux aussi, une marge existe entre les bornes supérieure et inférieure connues. Si on note β^n leur nombre maximum d'ensembles dominants minimaux, Couturier *et al.* (2013b) ont montré que $1.4422^n \leq \beta^n \leq 1.6181^n$. Est-il possible d'améliorer ces bornes ?

D'autres classes de graphes pourraient être considérées pour établir des bornes sur le nombre d'ensembles dominants minimaux, et notamment la classe des *graphes de disques unitaires* (en anglais, *unit disk graphs*), qui est formée des graphes d'intersection de collection de disques de rayon unitaire. Ils généralisent, en dimension 2, les graphes d'intervalles propres et les graphes d'intervalles unitaires. Cette question a d'ailleurs encore été soulevée récemment au workshop *Enumeration Algorithms Using Structure*, au Lorentz Center (Leiden, Pays-Bas, août 2015).

Un autre problème proche est celui de l'énumération des ensembles dominants minimaux connexes. Golovach *et al.* (2015) ont initié leur étude sur quelques classes de graphes. Dans le cas général, la meilleur borne supérieure connue est en 2^n. Cette borne est triviale. Est-il possible de l'améliorer ? La connectivité du problème le rend certainement plus difficile. Déjà pour la version d'optimisation, qui demande de calculer un ensemble dominant connexe de taille minimale, ou de façon équivalente, un arbre couvrant ayant un nombre maximum de feuilles, le meilleur algorithme s'exécute en temps $\mathcal{O}(1.8619^n)$ contre $\mathcal{O}(1.4864^n)$ pour le problème DOMINATION (Abu-Khzam *et al.*, 2011; Iwata, 2011).

Finalement, pourrait-on développer une autre technique que le *branchement* pour établir (de façon « algorithmique ») des bornes combinatoires ? Par exemple, Nederlof *et al.* (2014) utilisent le principe d'*inclusion-exclusion* (certes, combiné également avec du *branchement*) pour concevoir un algorithme capable de dénombrer les ensembles dominants. Est-ce qu'une approche d'*inclusion-exclusion* ou une technique différente du *branchement* serait à même d'énumérer des objets combinatoires et de prouver de bonnes bornes supérieures sur leur nombre ?

Chapitre 4

Problèmes d'ordonnancement

Les problèmes d'ordonnancement ont été intensivement étudiés ces dernières décennies. À la fois des méthodes exactes et des algorithmes d'approximations ont été proposés, avec des techniques comme *séparation-et-évaluation*, *heuristiques*, *approximations avec garantie de performances*, ou encore des *algorithmes génétiques*. Très peu de travaux ont été menés pour concevoir des algorithmes exacts avec des garanties de performance au pire des cas meilleures que la recherche exhaustive. Dans ce chapitre, nous étudions deux approches pour l'obtention d'algorithmes modérément exponentiels. Tout d'abord, nous proposons des algorithmes de *programmation dynamique*. Cette technique classique est certainement la plus naturelle pour les problèmes d'ordonnancement. Les premiers algorithmes utilisant cette technique pour ces problèmes datent des années soixante (Held et Karp, 1962). Nous proposons quelques algorithmes de *programmation dynamique*, couplés avec le *produit de convolution rapide* pour la résolution de problèmes d'ordonnancement. Comme deuxième approche, nous abstrayons puis nous généralisons la technique *trier-et-chercher* introduite par Horowitz et Sahni (1974). Grâce à un algorithme résolvant cette généralisation et utilisant des structures de données un peu particulières, nous résolvons un certain nombre de problèmes d'ordonnancement pouvant se modéliser comme des *problèmes à une seule et à plusieurs contraintes*.

Sommaire

Les problèmes autour de l'ordonnancement de tâches sont très étudiés et leurs applications sont nombreuses dans l'industrie. Typiquement, ces problèmes demandent de déterminer une affectation optimale d'un ensemble de tâches (on parle également de *travaux*) sur des machines (on parle aussi de *ressources*) sur une période de temps. Nous le verrons par la suite, des contraintes supplémentaires peuvent définir le problème. Comme en témoigne l'abondante littérature, l'étude des problèmes d'ordonnancement est intense depuis les années 50 avec des résultats de complexité et le développement d'approches pour leur résolution. La plupart des problèmes étudiés sont NP-difficiles. D'un point de vue algorithmique, les travaux proposés dans la littérature présentent des algorithmes d'approximation ou des heuristiques, combinés avec des approches de type *séparation et évaluation* (en anglais, *branch-and-bound*), de *plans sécants* (en anglais, *branch-and-cut*), de *programmation linéaire entière* (avec des approches de type *branch-and-price*). Ces approches donnent des résultats qui peuvent se révéler tout à fait satisfaisants en pratique, mais ces techniques ne permettent pas une résolution exacte avec une garantie sur le temps d'exécution au pire des cas. Curieusement, assez peu de résultats sont connus dans le domaine des algorithmes modérément exponentiels.

Pour la résolution de problème d'ordonnancement, nous nous sommes intéressés à trois techniques : la *programmation dynamique*, *trier-et-chercher*, *brancher-et-réduire*. La première technique est classique et l'on pourrait dire qu'elle « a fait ses preuves » sur des problèmes de permutation. La dernière technique est prometteuse car elle ne demande (bien souvent) qu'un espace polynomial ; ce qui pourrait se révéler bien utile pour des implémentations efficaces. Par ailleurs, cette technique de *branchement* se retrouve dans les algorithmes de type *séparation et évaluation*. Malheureusement, jusqu'à présent nos progrès sont restés très limités avec cette technique, ce qui laisse ouvertes de nombreuses questions intéressantes sur son application. Finalement, c'est la technique *trier-et-chercher* que nous avons réussi à généraliser pour résoudre des problèmes à *plusieurs dimensions*, et grâce à laquelle nous obtenons des algorithmes applicables à divers problèmes.

Les résultats présentés dans ce chapitre sont extraits des publications suivantes :

> **(Lenté *et al.*, 2013)** LENTÉ, C., LIEDLOFF, M., SOUKHAL, A. et T'KINDT, V. (2013). On an extension of the Sort & Search method with application to scheduling theory. *Theor. Comput. Sci.*, 511:13–22
>
> **(Lenté *et al.*, 2015)** LENTÉ, C., LIEDLOFF, M., T'KINDT, V. et TODINCA, I. (2015). Algorithmes modérément exponentiels pour problèmes NP-difficiles. *In* OLLINGER, N., editor: *Informatique Mathématique une photographie en 2015*. CNRS Editions

4.1 Définition des problèmes et résultats connus

Les problèmes d'ordonnancement sont traditionnellement définis par la notation $\alpha|\beta|\gamma$ introduite par Graham *et al.* (1979) et reprise par Blazewicz *et al.* (1983). Le champ α décrit les machines, le champ β les contraintes additionnelles et le champ γ décrit le critère d'optimalité.

La notation $\alpha|\beta|\gamma$

Nous reprenons la description donnée par Blazewicz *et al.* (1983). Nous supposons que $\mathcal{J}$ est

un ensemble de n travaux (ou tâches) notés $J_1, J_2, \ldots, J_n$. On désigne par $\mathcal{M}$ l'ensemble de m machines : $M_1, M_2, \ldots, M_m$. Nous supposons que chaque machine ne peut traiter qu'au plus une tâche à un moment donné et qu'une tâche ne peut être affectée qu'à au plus une machine à un moment donné.

Donnons une description plus détaillée des trois champs α, β et γ :

- α : la nature de l'atelier. Le champ $\alpha = \alpha_1\alpha_2$ définit l'environnement des machines.

  Pour $\alpha_1 \in \{P, Q, R\}$, chaque travail J_j consiste en une unique opération qui doit être traitée sur l'une quelconque des machines. La durée opératoire (aussi appelée durée de traitement) de J_j sur M_i est donnée par p_{ij} ($1 \leq i \leq m$ et $1 \leq j \leq n$).

  - si $\alpha_1 = P$ alors l'atelier est composé de machines parallèles identiques ($p_{ij} = p_j$ où p_j est la durée nécessaire pour traiter J_j) ;
  - si $\alpha_1 = Q$ alors l'atelier est composé de machines parallèles uniformes ($p_{ij} = p_j/q_i$ pour une vitesse q_i de M_i donnée) ;
  - si $\alpha_1 = R$ alors l'atelier est composé de machines parallèles non reliées (p_{ij} est arbitraire).

  Pour $\alpha_1 \in \{O, F, J\}$, chaque travail J_j consiste en un ensemble O_{ij} de m_j opérations qui doivent chacune être opérées sur une machine μ_{ij} pendant une durée p_{ij} ($1 \leq i \leq m_j$ et $1 \leq j \leq n$).

  - si $\alpha_1 = O$ alors le cheminement est libre ($m_j = m$ et $\mu_{ij} = M_i$) ;
  - si $\alpha_1 = F$ alors le cheminement est unique ($m_j = m$, $\mu_{ij} = M_i$ et $O_{i-1,j}$ doit être terminée avant que O_{ij} ne puisse commencer) ;
  - si $\alpha_1 = J$ alors le cheminement est multiple (m_j et μ_{ij} sont arbitraires, $\mu_{i-1,j} \neq \mu_{ij}$ et $O_{i-1,j}$ doit être terminée avant que O_{ij} ne puisse commencer).

  Si α_2 est précisé, alors c'est un entier qui indique le nombre de machines (qui est constant) ; s'il n'est pas précisé, alors le nombre de machines fait partie de l'entrée.

- β : les contraintes additionnelles. Le champ β donne des caractéristiques additionnelles sur les travaux.

  - si β contient $p_i = p$ ou $p_{ij} = p$ cela signifie que la durée de traitement est identique pour tous les travaux. Par $p_i = 1$ ou $p_{ij} = 1$, on indique que cette durée est unitaire ;
  - si β contient r_i, alors pour chaque travail, une date de disponibilité est donnée et le travail ne peut être traité avant cette date ;
  - si β contient d_i, alors pour chaque travail, une date de fin souhaitée est donnée et le travail ne peut être traité après cette date ;
  - si β contient $size_i = k$, alors chaque travail doit au même moment être traité par k machines ;
  - si β contient « pmtn », alors les travaux peuvent à tout moment être préemptés et leur traitement terminer plus tard sur éventuellement une autre machine ;
  - si β contient « prec », alors un graphe orienté acyclique est donné et indique des relations de précédence entre les travaux ;
  - ...

- γ : le critère d'optimalité. Ce dernier champ γ indique le critère à minimiser. Chaque ordonnancement définit pour chaque travail J_j une date de fin effective C_j et, étant donnée une date de fin souhaitée d_j, il définit un retard algébrique $L_j = C_j - d_j$ $(1 \leq j \leq n)$, un retard absolu $T_i = \max\{0, L_i\}$ ou un indicateur de retard $U_i = 1_{\text{si } C_i > d_i}$. On peut aussi affecter une pondération w_j à chaque travail et obtenir ainsi diverses fonctions objectifs :

  - $C_{\max} = \max\{C_1, C_2, \ldots, C_n\}$, c'est-à-dire la date de fin du dernier travail traité dans l'ordonnancement (aussi appelé *makespan*) ;
  - $\sum C_j = C_1 + C_2 + \cdots + C_n$ qui est le temps de traitement total ;
  - $L_{\max} = \max\{L_1, L_2, \ldots, L_n\}$ qui est le maximum des retards algébriques ;
  - $T_{\max} = \max\{T_1, T_2, \ldots, T_n\}$ qui est le maximum des retards absolus ;
  - $\sum T_i = T_1 + T_2 + \cdots + T_n$ qui est la somme des retards absolus ;
  - $\sum w_i T_i = w_1 T_1 + w_2 T_2 + \cdots + w_n T_n$ qui représente la somme pondérée des retards absolus ;
  - $\sum U_i = U_1 + U_2 + \cdots + U_n$ qui représente le nombre de travaux en retard ;
  - $\sum w_i U_i = w_1 U_1 + w_2 U_2 + \cdots + w_n U_n$ qui représente le nombre pondéré de travaux en retard.

Exemple. *Quelques exemples permettent de mieux comprendre la notation. Pour $\alpha = 1$, il s'agit de problème à une machine. Pour des problèmes de machines parallèles, on utilisera $\alpha = P$ ou $\alpha = P_2$ si on souhaite préciser qu'il y a deux machines. Pour un problème de flow-shop à trois machines, on utilisera $\alpha = F_3$. Si l'on s'intéresse à un problème à deux machines parallèles, pour lequel chaque travail a une date de fin souhaitée et l'on souhaite minimiser la somme pondérée du nombre de travaux en retard, on désignera simplement ce problème par $P_2|d_i|\sum w_i U_i$.*

La taxinomie précédente montre la richesse des problèmes étudiés en ordonnancement. Il est impossible dans ce document de dresser un état de l'art exhaustif. C'est pourquoi nous restreignons notre étude aux résultats relatifs aux algorithmes modérément exponentiels pour ces problèmes. Nous réferons le lecteur intéressé aux ouvrages re référence de Brucker (2007) et Pinedo (2012).

Nous pouvons néanmoins souligner que diverses approches ont été intensivement étudiées dans la littérature pour la résolution des problèmes d'ordonnancement. Parmi ces approches, les algorithmes de *séparation et évaluation* (*branch-and-bound*) calculent une solution exacte. On trouve également des *heuristiques* pour obtenir des solutions de *bonne* qualité, des algorithmes *génétiques*, de *recuit simulé*, de *recherche tabou*, ou des approches hybrides combinant ces techniques. Par ailleurs, des algorithmes *d'approximation* sont parfois disponibles, avec des bornes garanties sur la qualité de la solution.

Marx (2011) soulignait que peu de résultats étaient établis au regard des algorithmes et de la complexité à paramètre fixé. Depuis, Chu *et al.* (2013) et Mnich et Wiese (2014) ont proposé des algorithmes FPT et des preuves de W[1]-difficultés pour divers problèmes et paramètres.

Du point de vue de l'algorithmique exponentielle, nous pouvons recenser les travaux de Held et Karp (1962), Lawler (1977), Woeginger (2001), Gromicho *et al.* (2012), Jansen *et al.* (2013), et Cygan *et al.* (2014). Nous résumons succinctement leurs principaux résultats :

- Held et Karp (1962) donnent un algorithme de *programmation dynamique* en $\mathcal{O}^*(2^n)$ pour le problème $1||\sum f_i$ (minimiser la somme des dates de fin de chacune des tâches pour un problème à une machine) ;

- Lawler (1977) propose un algorithme de *programmation dynamique* pseudo-polynomial en $\mathcal{O}(n^4\,P)$ et $\mathcal{O}(n^5\,p_{\max})$ pour $1||\sum w_jT_j$ (minimiser la somme pondérée des travaux en retard, relativement à une date de fin souhaitée) dans le cas où $p_j < p_i$ implique $w_j \geq w_i$, avec $P = \sum p_i$ et $p_{\max} = \max p_i$;

- Woeginger (2001) propose dans son état de l'art de résoudre le problème $1|\text{prec}|\sum w_iC_i$ (minimiser la somme pondérée des dates de fin des travaux, sous des contraintes de précédence) par *programmation dynamique* en temps $\mathcal{O}^*(2^n)$;

- Gromicho *et al.* (2012) donnent un algorithme de *programmation dynamique* qui s'exécute en temps $\mathcal{O}((p_{\max}^{2n}\,(m+1)^n)$ pour la résolution d'un problème de flow-shop à m machines ;

- Jansen *et al.* (2013) démontrent des bornes inférieures : pour divers problèmes (dont $P_2||C_{\max}$ et $P_2|d_i|\sum w_iC_i$), ils prouvent, sous l'hypothèse ETH, que ces problèmes ne peuvent être résolus plus rapidement que $2^{o(n)}\cdot||I||^{O(1)}$, où $||I||$ est la taille de l'instance. Par ailleurs, ils proposent des algorithmes de *programmation dynamique* en $\mathcal{O}^*(2^{O(n)})$ pour résoudre ces problèmes ;

- Cygan *et al.* (2014) présentent un algorithme en $\mathcal{O}^*(c^n)$, avec $c < 2$ une constante, pour le problème $1|\text{prec}|\sum C_i$. Ce résultat est obtenu grâce à des propriétés structurelles de la solution, permettant ainsi d'éviter certains ensembles dans le schéma de *programmation dynamique*.

4.2 Programmation dynamique

L'une des techniques classiques pour les problèmes où l'on doit déterminer une permutation optimale est la *programmation dynamique*. Dans son état de l'art, Woeginger (2001) décrit un schéma de *programmation dynamique* pour le problème $1|\text{prec}|\sum w_iC_i$ (en anglais, *total completion time scheduling under precedence constraints*) en temps $\mathcal{O}^*(2^n)$. Il laisse en exercice la résolution des problèmes $1|d_i|\sum w_iU_i$ (c'est-à-dire, où l'on souhaite minimiser le nombre pondéré de travaux en retard) et $1|d_i|\sum T_i$ (où l'on souhaite minimiser le retard total) en temps $\mathcal{O}^*(2^n)$. Comme exemple introductif de la *programmation dynamique*, Fomin et Kratsch (2010) présentent dans leur ouvrage un problème d'ordonnancement où, à chaque travail J_i, est associée une fonction de coût f_i (c'est-à-dire qu'il y a un coût $f_i(t)$ lorsque le travail J_i termine au temps t). On souhaite alors minimiser le coût total d'un ordonnancement qui est défini par $\sum_{i=1}^{n} f_i(C_i)$. Comme contrainte (très naturelle) supplémentaire, les auteurs demandent qu'à partir du temps 0 les travaux soient traités sans interruption et sans « temps mort » ;' autrement dit $C_{\max} = \sum p_i$. Comme adopté par Lenté (2012), nous notons une telle contrainte par *cpct* (pour *compactée*). Avec la notation usuelle, le problème peut donc se définir par $1|cpct|\sum f_i$.

4.2.1 Problèmes à une machine

Pour résoudre l'ensemble des problèmes que nous venons de décrire, le schéma de *programmation dynamique* utilisé s'inspire fortement de celui de Bellman (1962) et Held et Karp (1962) de pour le problème VOYAGEUR DE COMMERCE. D'ailleurs, dans leur article, Held et Karp (1962) démontrent déjà l'utilisation de la *programmation dynamique* pour la résolution de $1|cpct|\sum f_i$. On vérifie facilement que le temps d'exécution de cette *programmation dynamique* est en $\mathcal{O}^*(2^n)$, ce qui améliore grandement l'énumération exhaustive de toutes les permutations en temps $\mathcal{O}^*(n!)$. Donnons par

exemple un schéma de *programmation dynamique* pour le problème $1|prec, cpct|f_i$ (les schémas de *programmation dynamique* pour la résolution des autres problèmes mentionnés en sont très proches).

Supposons que $\mathcal{J}$ soit l'ensemble des travaux à ordonnancer sur une machine, de telle sorte que la contrainte de compactage *cpct* soit respectée, ainsi que les contraintes de précédence *prec*. Ces contraintes de précédence peuvent nous être données sous la forme d'un graphe orienté acyclique et, pour deux travaux J_i et J_j, nous écrirons $J_i \to J_j$ si le travail J_i doit être traité avant le travail J_j. Étant donné un ensemble $S \subseteq \mathcal{J}$ de travaux, on note $\text{NoSucc}(S)$ l'ensemble des travaux de S qui n'admettent pas de successeur dans S. De la même façon on définit par $\text{NoPred}(S)$ l'ensemble des travaux de S qui n'admettent pas de prédécesseur dans S. Par abus de notation, nous noterons i pour J_i. Nous notons $P(S) = \sum_{J_i \in S} p_i$ la somme des durées opératoires des travaux de S. On obtient le schéma suivant, où les ensembles S sont évalués par ordre de cardinalité croissante :

$$\begin{cases} Opt[\{j\}] = f_j(p_j) \text{ pour tout } j \in \text{NoPred}(\{1, 2, \ldots, n\}) \\ Opt[S] = \min_{j \in \text{NoSucc}(S)} \{Opt[S \setminus \{j\}] + f_j(P(S))\} \end{cases}$$

où $Opt[S]$ donne le coût optimal d'un ordonnancement des travaux de S respectant les contraintes. Finalement, le coût d'un ordonnancement optimal est donné par la valeur de $Opt[\{1, 2, \ldots, n\}]$. Une description en pseudo-code de l'algorithme basé sur ce schéma de *programmation dynamique* est donné par l'algorithme 11.

Algorithme 11 : OrdonnancementUneMachine$(\mathcal{J}, p_1, p_2, \ldots, p_n, f_1, f_2, \ldots, f_n)$

Entrée : Un ensemble de travaux $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ avec leurs durées opératoires $p_1, p_2, \ldots, p_n$ et n fonctions de coût $f_i : \mathbb{R} \to \mathbb{R}$ $(1 \leq i \leq n)$.
Sortie : Le coût optimal d'un ordonnancement des n travaux sur une machine.

```
pour chaque travail j ∈ NoPred(J) faire
    Opt[{j}] ← f_j(p_j)
pour ℓ ← 2 à n faire
    pour chaque S ⊆ J tel que |S| = ℓ faire
        Opt[S] ← min_{j∈NoSucc(S)} {Opt[S \ {j}] + f_j(P(S))}
retourner Opt[J]
```

4.2.2 Problèmes à plusieurs machines

Intéressons-nous aux problèmes où les travaux peuvent être ordonnancés sur plusieurs machines parallèles identiques. On suppose que la contrainte de compactage *cpct* s'applique à chacune des machines : la date de fin de chaque machine est égale à la somme des durées opératoires des travaux traités par la machine. Cette contrainte est assez naturelle mais l'absence de temps morts nous permet, encore une fois d'appliquer un schéma de *programmation dynamique*. Pour traiter un problème à plusieurs machines, l'idée que nous développons est de partitionner itérativement l'ensemble des machines, jusqu'à se réduire à des problèmes à une unique machine dont nous avons présenté la résolution à la section 4.2.1. Pour accélérer le calcul, nous utiliserons le *produit de convolution rapide* de Björklund *et al.* (2007) (d'une façon similaire à celle que nous avons développée dans (Binkele-Raible *et al.*, 2013) pour le calcul d'un arbre couvrant ayant un nombre maximum de feuilles).

Soit s un ordonnancement des travaux. Nous notons $f_{\max}(s)$ (ou simplement $f_{\max}$ lorsque la notation $\alpha|\beta|\gamma$ est utilisée) la valeur de $\max_{1\leq i\leq n} f_i(C_i(s))$, où $C_i(s)$ est la date de fin de traitement du travail i dans l'ordonnancement s. De même, nous notons $\sum f_i(s)$ (ou simplement $\sum f_i$) la valeur $\sum_{i=1}^{n} f_i(C_i(s))$. À la prochaine section, nous décrivons notre algorithme pour ordonner des travaux sur plusieurs machines, puis à la section 4.2.2.2 nous présentons le calcul du produit de convolution de deux fonctions. Avec ces deux ingrédients, nous établissons un algorithme en temps $\mathcal{O}^*(2^n M)$ pour la résolution du problème $P_m|cpct|\sum f_i$, où $M = \sum_{i=1}^{n} j_i$ et j_i représente la durée opératoire du travail i ($1 \leq i \leq n$). Cette approche peut également résoudre le problème $P_m|cpct|f_{\max}$ où l'on cherche à minimiser $\max_{i=1}^{n} f_i(C_i)$ et être adaptée pour résoudre d'autres problèmes à plusieurs machines.

4.2.2.1 Description de l'algorithme

Soit $\mathcal{J}$ un ensemble de n travaux à ordonnancer sur m machines parallèles identiques. L'algorithme 12 détermine un ordonnancement optimal qui résout $P_m|cpct|\sum f_i$, c'est-à-dire qui minimise $\sum_{i=1}^{n} f_i(C_i)$.

Algorithme 12 : OrdonnancementPlusieursMachines$(\mathcal{J}, p_1, p_2, \ldots, p_n, f_1, f_2, \ldots, f_n, m)$

Entrée : Un ensemble de travaux $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ avec leur durée opératoire $p_1, p_2, \ldots, p_n$ et n fonctions de coût $f_i : \mathbb{R} \to \mathbb{R}$ ($1 \leq i \leq n$). Le paramètre m indique le nombre de machines parallèles identiques.
Sortie : Le coût optimal d'un ordonnancement des n travaux sur une machine.

```
/* Un appel à OrdonnancementUneMachine construit la table Opt_1 tel
   que décrite dans l'algorithme 11                                  */
Opt_1 ← OrdonnancementUneMachine(J, p_1, p_2, ..., p_n, f_1, f_2, ..., f_n)
pour k ← 2 à m faire
    pour chaque S ⊆ J faire
        Opt_k[S] ← min_{X⊆S}{Opt_{⌊k/2⌋}[X] + Opt_{⌈k/2⌉}[S \ X]}
retourner Opt_m[J]
```

L'algorithme considère itérativement un ordonnancement sur un nombre k de machines, jusqu'à considérer $k = m$. Pour $k = 1$, une solution optimale est calculée par **OrdonnancementUneMachine** (voir l'algorithme 11). Nous supposons que **OrdonnancementUneMachine** nous retourne la table OPT (appelé OPT_1 dans l'algorithme **OrdonnancementPlusieursMachines**) qu'il construit au cours de son exécution. Ainsi, pour chaque sous-ensemble S de tâches, $\text{OPT}_1[S]$ contient un ordonnancement optimal des tâches de S sur une machine. Les tables $\text{OPT}_2, \text{OPT}_3, \ldots, \text{OPT}_m$ sont ensuite successivement construites. Pour calculer OPT_k ($k \in \{2, 3, \ldots, m\}$) pour chaque sous-ensemble de travaux $S \subseteq \mathcal{J}$, on observe qu'il existe un certain sous-ensemble $X \subseteq S$ des travaux de S qui sont exécutés sur les $\lfloor k/2 \rfloor$ premières machines et le reste des travaux (*i.e.* $S \setminus X$) sur les $\lceil k/2 \rceil$ dernières machines. On rappelle que ces machines sont toutes identiques, ce qui garantit la validité de l'approche.

Pour chaque k fixé, le nombre d'ensembles S et X considérés par l'algorithme est donné par

$$\sum_{s=0}^{n} \left(\binom{n}{s} \sum_{x=0}^{s} \binom{s}{x} \right) = 3^n.$$

De plus, pour chaque ensemble S et chaque ensemble X donnés, le temps d'exécution de l'algorithme **OrdonnancementPlusieursMachines** est polynomial puisque la recherche de l'entrée X

dans la table Opt_ℓ (avec $\ell < k$) peut être effectuée en un temps logarithmique en la taille de Opt_ℓ. On établit ainsi le théorème suivant :

Théorème 4.1. *Le problème $P_m|cpct|\sum f_i$ peut être résolu en temps $\mathcal{O}^*(3^n)$.*

Nous observons dans la suite que ce temps d'exécution peut être ramené à $\mathcal{O}^*(2^n M)$, où $M = \sum_{i=1}^{n} p_i$. Lorsque les durées opératoires des tâches sont bornées par un polynôme en n, ce temps d'exécution s'écrit encore $\mathcal{O}^*(2^n)$.

4.2.2.2 Le produit de convolution

Björklund *et al.* (2007) ont proposé un algorithme rapide pour calculer le produit de convolution de fonctions. Étant donné deux fonctions f et g, leur produit de convolution noté $f * g$ est défini pour tout $S \subseteq V$ par

$$(f * g)(S) = \sum_{T \subseteq S} f(T) g(S \setminus T). \tag{4.1}$$

Une approche directe pour le calcul de ce produit demanderait un temps en $\mathcal{O}^*(3^n)$. Par une transformée de Möbius et une inversion, l'expression (4.1) peut être évaluée en temps $\mathcal{O}^*(2^n \log M)$, où les fonctions f et g ont pour ensemble d'arrivée $\{-M, -M+1, \ldots, M\}$ (Björklund *et al.*, 2007) (voir aussi la section 1.4.3 à la page 29 pour davantage de détails).

Nous allons appliquer le produit de convolution rapide sur le demi-anneau *min-somme* :

$$(f * g)(S) = \min_{T \subseteq S} \{f(T) + g(S \setminus T)\}. \tag{4.2}$$

Björklund *et al.* (2007) montrent que leur approche pour l'anneau *somme-produit* peut être étendue au demi-anneau *min-somme*, en continuant de travailler avec l'anneau *somme-produit* mais en modifiant les fonctions f et g. Cela induit un coût supplémentaire qui résulte en une complexité en $\mathcal{O}^*(2^n M)$ pour l'évaluation de l'expression (4.2).

Il nous reste à expliquer l'utilisation du produit rapide de convolution (4.2) dans la récurrence utilisée par la *programmation dynamique* de l'algorithme 12 :

$$Opt_k[S] \leftarrow \min_{X \subseteq S} \{Opt_{\lfloor k/2 \rfloor}[X] + Opt_{\lceil k/2 \rceil}[S \setminus X]\} \quad \text{pour } k \in \{2, \ldots, m\}.$$

Autrement dit, le calcul de $Opt_k[S]$ nécessite les valeurs (déjà calculées) de $Opt_{k'}[X]$ pour $X \subseteq S$ et $k' < k$.

Pour tout k, avec $1 \leq k \leq m$, nous définissons les fonctions $f_k(X)$ où $X \subseteq \mathcal{J}$ par :

$$f_{k'}(X) = \begin{cases} Opt_1[X] & \text{pour } k' = 1, \text{ où la table } Opt_1 \text{ est obtenu par l'algorithme 11} \\ Opt_{k'}[X] & \text{pour } 2 \leq k' \leq m. \end{cases}$$

Ces m fonctions sont calculées successivement : f_1 (par application directe de l'algorithme 11 en temps $\mathcal{O}^*(2^n)$), puis les fonction $f_2, f_3, \ldots, f_m$ sont obtenues par le produit de convolution :

$$f_k(X) = (f_{\lfloor k/2 \rfloor} * f_{\lceil k/2 \rceil})(X) \quad \text{pour } 2 \leq k \leq m.$$

Ces fonctions peuvent être calculées (pour tous les ensembles X) par le produit de convolution sur le demi-anneau *min-somme* en temps total $\mathcal{O}^*(2^n M)$, où M n'excède pas $\sum_{i=1}^{n} p_i$ et les p_i ($1 \leq i \leq n$) représentent les durées opératoires des n travaux. Finalement, on observe que $Opt_k[S] = f_k(S)$.

Ceci nous permet d'établir le théorème suivant :

Théorème 4.2. *Le problème $P_m|cpct|\sum f_i$ peut être résolu en temps $\mathcal{O}^*(2^n M)$, où $M = \sum_{i=1}^{n} p_i$.*

La *programmation dynamique* est un outil classique pour ces problèmes de permutation. Le produit rapide de convolution nous autorise ici à améliorer son temps d'exécution. Il ne fait pas de doute que l'approche que nous avons présentée dans cette section s'applique également à d'autres problèmes d'ordonnancement. À la section 4.3, nous nous intéressons à une autre technique, plus méconnue : *trier-et-chercher*. Avant cela, regardons un autre type de problème d'ordonnancement : les problèmes de *flow-shop*.

4.2.3 Problème de flow-shop à trois machines

Dans cette section, nous considérons un problème de « flow-shop » (en français, nous dirions *atelier en série*). On se donne m machines $m_1, m_2, \ldots, m_m$ et chaque tâche doit être traitée sur la machine m_1, puis sur la machine m_2, puis sur la machine m_3, ... Chaque machine ne peut traiter qu'une tâche à la fois. L'objectif est de minimiser la durée opératoire totale de l'ensemble de ces tâches. Cette durée est donnée par la date de fin de traitement sur la dernière machine, notée $C_{\max}$. Un exemple est proposé à la figure 4.1. Pour un problème à deux machines, noté $F_2||C_{\max}$ (F pour *flow-shop*), Johnson (1954) a montré que le problème est polynomial. Dès que l'on dispose d'au moins trois machines, le problème est NP-complet (Garey *et al.*, 1976). Dans cette section, nous proposons un algorithme de *programmation dynamique* pour la résolution du problème $F_3||C_{\max}$ en temps $\mathcal{O}^*(3^n)$, où n est le nombre de tâches.

$F_3||C_{\max}$

Entrée. Un ensemble $\mathcal{J} = (J_1, J_2, \ldots, J_n)$ de n tâches, 3 machines m_1, m_2, m_3 organisées en atelier en série, la durée opératoire p_{m_i,J_k} de la tâche J_k sur la machine m_i ($1 \leq i \leq 3$, $1 \leq k \leq n$).

Question. Un ordonnancement des n tâches, où chaque tâche est traitée par la machine m_1, puis par m_2, puis par m_3, et qui minimise $C_{\max}$ qui est la date de fin de traitement de la dernière tâche affectée à m_3.

Pour les problèmes $F_2||C_{\max}$ et $F_3||C_{\max}$, il existe des ordonnancements optimaux pour lesquels la séquence de traitement des travaux est la même sur toutes les machines. En conséquence, on peut facilement obtenir un algorithme naïf en $\mathcal{O}^*(n!)$ pour résoudre le problème $F_3||C_{\max}$. Cette propriété n'est pas nécessairement vérifiée pour les problèmes $F_i||C_{\max}$ ($i \geq 4$), même s'il existe toujours un ordonnancement qui soit le même sur les deux premières machines ainsi que sur les deux dernières (voir l'ouvrage de Pinedo (2012)). Cela fait donc une différence importante entre le problème $F_3||C_{\max}$ et les problèmes $F_i||C_{\max}$ pour $i \geq 4$.

Gromicho *et al.* (2012) montrent que le problème $F_m||C_{\max}$ peut-être résolu en temps et espace $\mathcal{O}^*((p_{\max}^{2n}(m+1)^n)$ par une approche de *programmation dynamique*, où $p_{\max}$ est le maximum des p_{m_i,J_k} sur toutes les machines m_i et travaux J_k. Cela donne une complexité de $\mathcal{O}^*(4^n)$ pour le problème $F_3||C_{\max}$ dans le cas où toutes les tâches ont une durée unitaire. Dans (Shang *et al.*, 2015), nous présentons un algorithme en temps $\mathcal{O}^*(3^n)$ pour le problème du flow-shop $F_3||C_{\max}$. Cet

algorithme utilise la *programmation dynamique* pour le calcul d'un front de Pareto de l'ensemble des permutations. Dans la suite de cette section, nous présentons un algorithme différent mais également basé sur la *programmation dynamique*. L'idée est de considérer des *paires* d'ensembles de travaux. Ces paires correspondent à des *chemins critiques* sur les deux premières machines. Notre algorithme associe à ces paires une date de fin de traitement sur la troisième machine, dont la valeur est calculée par *relâchement*. Nous décrivons notre algorithme à la section 4.2.3.2 et nous commençons d'abord par introduire quelques notions à la section suivante.

4.2.3.1 Graphe orienté et chemin critique

Étant donnée une permutation $\sigma = (J_1, J_2, \ldots, J_n)$ de n travaux, on peut définir un graphe orienté G_σ. On définit ce graphe de la façon suivante : chaque paire (i, J_k) correspond à un sommet ayant pour poids p_{m_i,J_K}, c'est-à-dire au temps de traitement du travail J_k sur la machine m_i. Ensuite, pour chaque $1 \leq i < m$ et $1 \leq k < n$, un arc orienté est ajouté entre le sommet (i, J_k) et les sommets correspondant à $(i+1, J_k)$ et à $(i+1, J_{k+1})$. Pour chaque $1 \leq i < m$, un arc est ajouté entre le sommet (i, J_n) et le sommet $(i+1, J_n)$, et pour chaque $1 \leq k < n$, un arc est ajouté entre le sommet (m, J_k) et le sommet (m, J_{k+1}). Ce graphe ressemble à une grille avec des arcs orientés ; un exemple est donné à la figure 4.1.

Il est connu que la durée opératoire totale de la permutation σ, dont on rappelle qu'elle est donnée par la date de fin d'utilisation de la dernière machine, est égale au poids d'un chemin de plus grand poids entre le sommet $(1, J_1)$ et le sommet (m, J_n) (voir par exemple Pinedo (2012)).

Définition 4.1. Soit $\mathcal{J}$ un ensemble de n travaux, m machines et une permutation σ des travaux. Un chemin orienté de $(1, J_1)$ à (m, J_n) dans G_σ est un *chemin critique* s'il est de poids maximum sur tous les chemins de $(1, J_1)$ à (m, J_n).

La lecture d'un chemin critique peut aussi se faire directement sur le diagramme représentant la permutation des travaux sur les machines. Notamment, si $\sigma = (J_{i_1}, J_{i_2}, \ldots, J_{i_n})$ est une permutation optimale des n travaux, on notera un chemin critique par $\mathcal{P} = \big((1, J_{i_1}), \ldots, (1, J_{i_k}), (2, J_{i_k}), \ldots, (2, J_{i_\ell}), (3, J_{i_\ell}), \ldots\ldots, (m, J_{i_n})\big)$.

4.2.3.2 Description de l'algorithme

Plaçons-nous dans le contexte d'un flow-shop à 3 machines. Nous présentons sa résolution en temps et espace $\mathcal{O}^*(3^n)$, où n est le nombre de travaux. Nous commençons par quelques définitions préliminaires pour décrire notre algorithme et montrer sa correction.

Définition 4.2. Étant donnée une permutation (nous dirons aussi une *séquence*) σ, nous écrivons $x \prec_\sigma y$ si x apparaît avant y dans la permutation σ. Nous oublierons l'indice σ s'il n'y a pas d'ambiguité sur la permutation considérée.

Définition 4.3. Étant donnée deux séquences σ_1 et σ_2, nous notons $\sigma = \sigma_1 \cdot \sigma_2$ la séquence obtenue par la concaténation de σ_1 et σ_2. Pour un travail j, par abus de notation, nous écrirons $\sigma_1 \cdot j$ pour désigner la concaténation de σ_1 avec la séquence σ_2 réduite à l'unique élément j.

Une notion essentielle à notre algorithme est celle de *couples candidats* (voir la figure 4.2) :

Définition 4.4. Soit $X_1, X_2 \subseteq \mathcal{J}$. On dit que (X_1, X_2) est un *couple candidat* si :

- $X_1 \cap X_2 = \{j\}$, pour un certain travail $j \in \mathcal{J}$; et

travaux	J_1	J_2	J_3	J_4	J_5	J_6
p_{m_1,J_k}	2	3	2	3	3	2
p_{m_2,J_k}	1	3	1	4	2	2
p_{m_3,J_k}	3	4	2	3	2	1

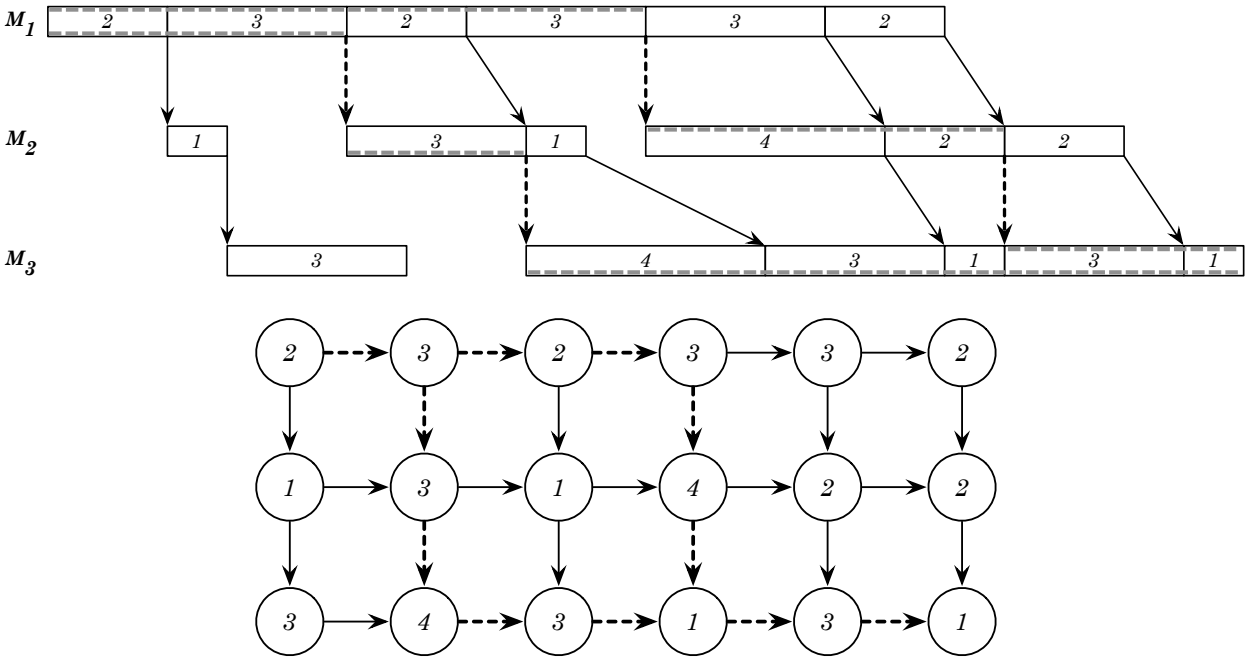


FIGURE 4.1 – Une table avec les durées opératoires de 6 travaux sur 3 machines, puis un exemple d'ordonnancement (avec la permutation $\sigma = (J_1, J_2, J_3, J_4, J_5, J_6)$ des travaux) en flow-shop de ces travaux. Sur l'exemple, la durée totale de traitement est $C_{\max} = 20$ (donnée par la date de fin de traitement sur la troisième machine). Pour cet ordonnancement, il existe deux chemins critiques représentés par des tirets. On retrouve les chemins critiques dans le graphe orienté correspondant à cette même permutation $\sigma = (J_1, J_2, J_3, J_4, J_5, J_6)$.

- il existe une séquence σ des travaux de $(X_1 \cup X_2)$ telle que :

  (i) il n'y ait pas de temps mort sur la machine M_1 ;

  (ii) $J_1 \prec j \prec J_2$, pour tout $J_1 \in (X_1 \setminus \{j\})$ et $J_2 \in (X_2 \setminus \{j\})$;

  (iii) le travail j commence sur la machine M_2 dès qu'il termine sur la machine M_1 ; et

  (iv) il n'y a pas de temps mort sur la machine M_2 entre les travaux de X_2.

Étant donnée une séquence σ, elle *respecte* le couple candidat (X_1, X_2) si elle satisfait les conditions (i) à (iv).

Définition 4.5. À tout couple candidat $t = (X_1, X_2)$, on associe une valeur $\mu^*(t)$ égale à la plus petite date de fin de traitement (donnée par la troisième machine) sur toutes les séquences σ respectant t :

$$\mu^*(t) = \min_{\sigma \text{ respectant } t} C_{\max}.$$

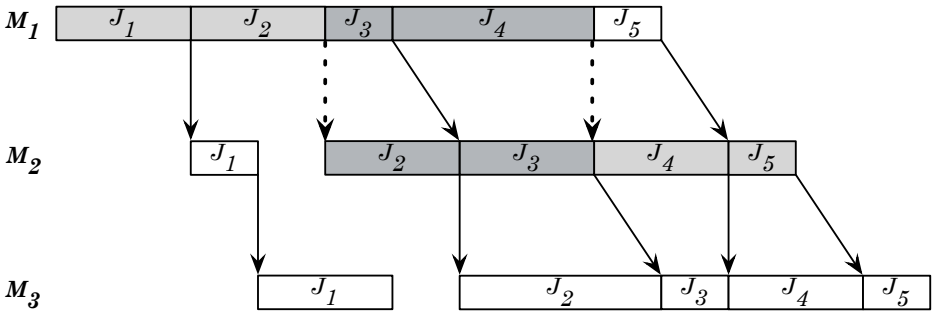


FIGURE 4.2 – Un exemple d'ordonnancement où $(X_1 = \{J_1, J_2\}, X_2 = \{J_2, J_3, J_4, J_5\})$ et $(X_1 = \{J_1, J_2, J_3, J_4\}, X_2 = \{J_4, J_5\})$ sont des couples candidats.

Remarque 10. *Si (X_1, X_2) est un couple candidat, alors il existe au moins une permutation σ des travaux de $X_1 \cup X_2$ qui respecte ce couple, ainsi qu'un chemin critique de longueur $\mu^*(X_1, X_2)$ dans G_σ (voir la figure 4.3).*

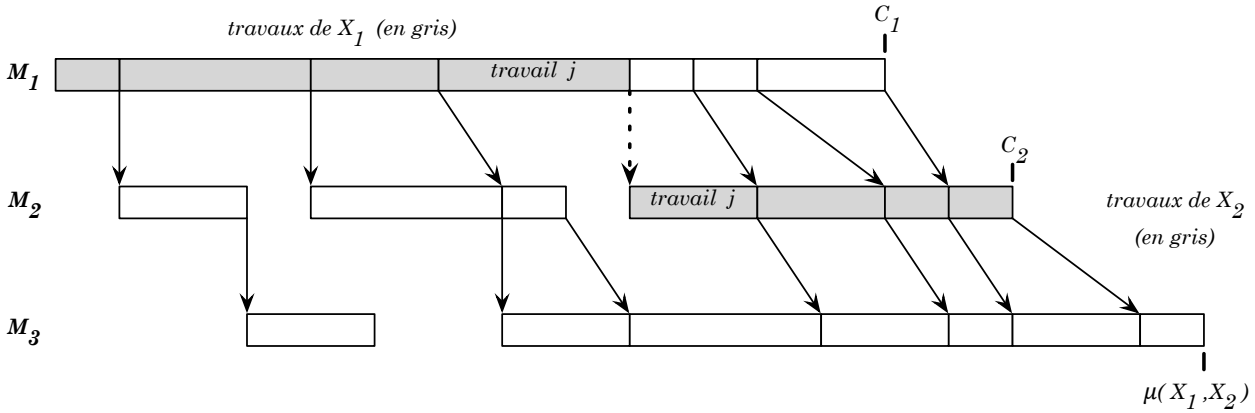


FIGURE 4.3 – Un exemple d'ordonnancement σ pour un couple (X_1, X_2) de tâches. Le travail j vérifie $X_1 \cap X_2 = \{j\}$.

Étant donné un couple candidat $t = (X_1, X_2)$, la plus petite date de fin de traitement des travaux (sur la machine 3) d'un ordonnancement σ respectant t, est donnée par $\mu^*(X_1, X_2) = C_{\max} = C_3(t)$. On observe que les dates de fin de traitement des deux autres machines, $C_1(t)$ et $C_2(t)$, peuvent être obtenues facilement.

Proposition 4.3. *Les valeurs de $C_1(t)$ et $C_2(t)$ sont données par :*

$$\begin{aligned} C_1(t) &= \sum_{i \in X_1 \cup X_2} p_{m_1,i}, \\ C_2(t) &= \sum_{i \in X_1} p_{m_1,i} + \sum_{i \in X_2} p_{m_2,i}. \end{aligned}$$

On note que les calculs de $C_1(t)$ et de $C_2(t)$ sont indépendants de toute séquence σ respectant $t = (X_1, X_2)$, puisqu'ils ne dépendent que de X_1 et de X_2.

Notre algorithme utilise une approche par *relâchement*. Pour tout couple candidat t, il calcule une borne supérieure de $\mu^*(t)$, notée $\mu(t)$, qu'il diminue itérativement jusqu'à en obtenir la valeur

optimale $\mu^*(t)$. Cette approche n'est pas nouvelle et on retrouve l'utilisation d'un *relâchement* dans des algorithmes de calcul de plus courts chemins tels que ceux de Bellman (1958) et Ford (1956), ou de Dijkstra (1959). On réfère le lecteur à des ouvrages comme celui de Cormen *et al.* (2009) pour une présentation détaillée.

Une autre caractéristique de notre algorithme est « d'étendre » les couples candidats jusqu'à avoir considéré toutes les tâches, c'est-à-dire jusqu'à parvenir à des couples candidats (X_1, X_2) pour lesquels $X_1 \cup X_2 = \mathcal{J}$.

Définition 4.6. Étant donnés deux couples candidats $t = (X_1, X_2)$ et $t' = (X_1', X_2')$ où $X_1 \subseteq X_1'$, on dit que t' est une *extension* de t (ou simplement que t' *étend* t, noté $t \to t'$) s'il existe j tel que $(X_1' \cup X_2') \setminus (X_1 \cup X_2) = \{j\}$, et si pour toute séquence σ respectant t, la séquence $\sigma \cdot j$ respecte t'.

Lorsqu'on souhaite préciser l'élément j dans la définition précédente, nous disons aussi que t' est une *extension* de t par j (ou que t' *étend* t par j), que nous noterons $t \underset{j}{\to} t'$. Le lemme suivant définit plus précisément la forme du couple t' :

Lemme 4.4. *Soient $t = (X_1, X_2)$ et $t' = (X_1', X_2')$ deux couples candidats tels que $t \to t'$. Soit j tel que $(X_1' \cup X_2') \setminus (X_1 \cup X_2) = \{j\}$. Alors l'une des conditions suivantes est satisfaite :*

- *si $C_1(t) + p_{m_1,j} \leq C_2(t)$ alors*

$$X_1' = X_1, \quad X_2' = X_2 \cup \{j\}, \quad C_1(t') = C_1(t) + p_{m_1,j}, \quad C_2(t') = C_2(t) + p_{m_2,j};$$

- *si $C_1(t) + p_{m_1,j} \geq C_2(t)$ alors*

$$X_1' = X_1 \cup X_2 \cup \{j\}, \quad X_2' = \{j\}, \quad C_1(t') = C_1(t) + p_{m_1,j}, \quad C_2(t') = C_1(t) + p_{m_1,j} + p_{m_2,j}.$$

Démonstration. Considérons les deux couples t et t'. Soit σ une séquence quelconque qui respecte t. Puisque t' étend t alors, par définition, la séquence $\sigma' = \sigma \cdot j$ respecte t', avec j l'unique élément de $(X_1' \cup X_2') \setminus (X_1 \cup X_2)$. Comme j apparaît à la fin de la séquence σ', nous considérons deux cas (voir aussi la figure 4.4, où les cas (1) et (2) correspondent au premier cas de la preuve et les cas (3) et (4) correspondent au second) :

- Si $C_1(t) + p_{m_1,j} \leq C_2(t)$ alors puisque j est ajouté à la fin de la séquence σ, le couple $\hat{t} = (X_1, X_2 \cup \{j\})$ est candidat et étend t. On vérifie facilement que les conditions de la définition 4.4 sont respectées par $\hat{t}$ et la séquence $\sigma \cdot j$. De plus, on a effectivement $C_1(\hat{t}) = C_1(t) + p_{m_1,j}$ et $C_2(\hat{t}) = C_2(t) + p_{m_2,j}$.

- Si $C_1(t) + p_{m_1,j} \geq C_2(t)$ alors puisque j est ajouté à la fin de la séquence σ, le couple $\hat{t} = (X_1 \cup X_2 \cup \{j\}, \{j\})$ est candidat et étend t. Là encore, les conditions de la définition 4.4 sont respectées par $\hat{t}$ et la séquence $\sigma \cdot j$. On a également $C_1(\hat{t}) = C_1(t) + p_{m_1,j}$ et $C_2(\hat{t}) = C_1(t) + p_{m_1,j} + p_{m_2,j}$.

Finalement, il n'existe pas d'autres couples candidats qui étendent t par l'ajout de j. En effet, puisque t' est un couple candidat, il n'y a ni temps mort sur la machine M_1 (condition (i)), ni temps mort sur la machine M_2 entre les travaux de X_2' (condition (iv)). Par conséquent, le couple t' ne peut prendre que l'une des deux formes présentées ci-dessus. □

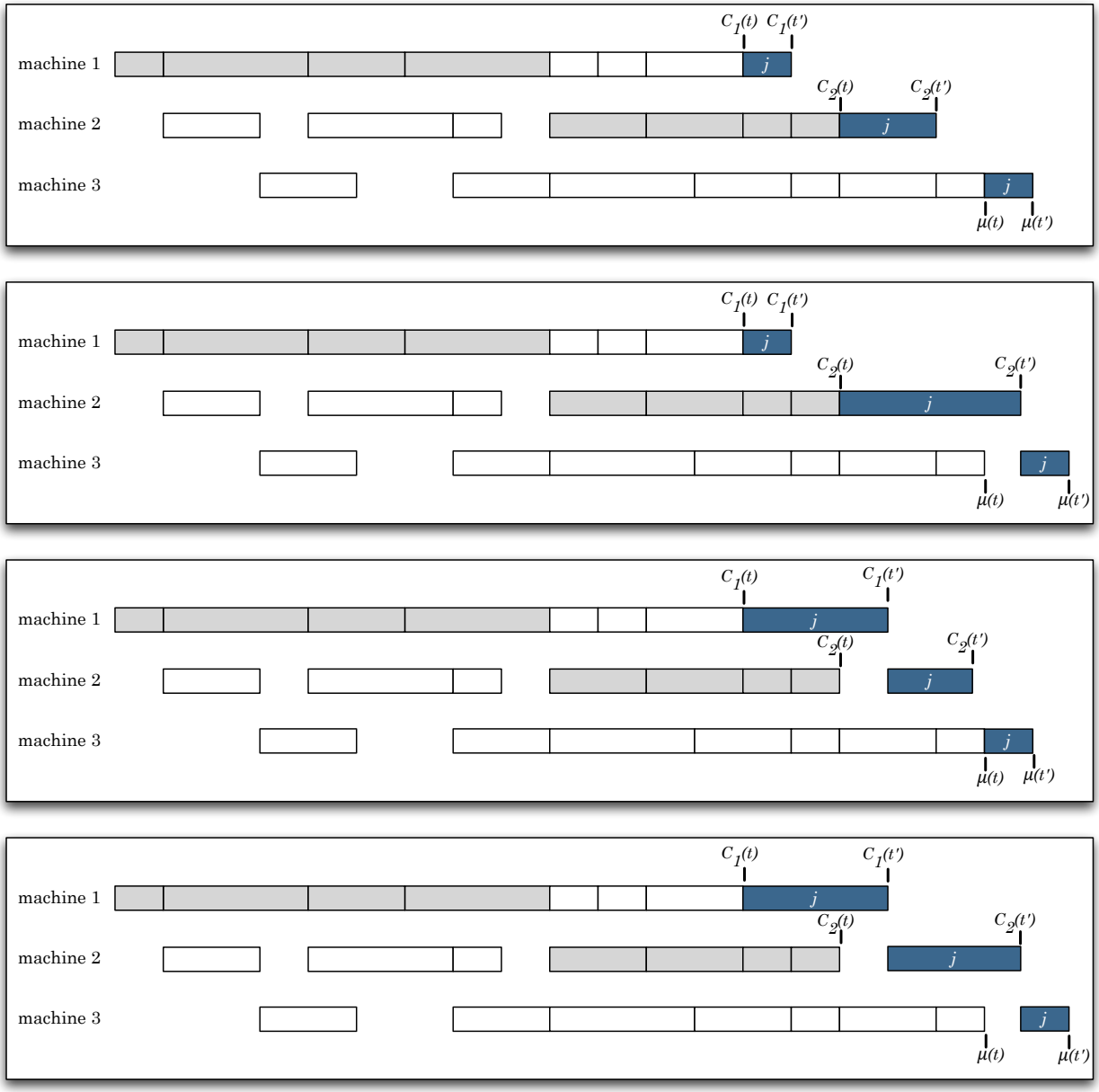


FIGURE 4.4 – Les 4 cas où l'on étend un couple candidat $t = (X_1, X_2)$ (représenté en grisé) en lui ajoutant une tâche j (représentée en bleu foncé) pour obtenir un couple t'.
Cas (1) : $C_1(t) + p_{m_1,i} < C_2(t)$ et $C_2(t) + p_{m_2,i} \leq \mu(t)$. Cas (2) : $C_1(t) + p_{m_1,i} < C_2(t)$ et $C_2(t) + p_{m_2,i} > \mu(t)$.
Cas (3) : $C_1(t) + p_{m_1,i} \geq C_2(t)$ et $C_2(t) + p_{m_2,i} \leq \mu(t)$. Cas (4) : $C_1(t) + p_{m_1,i} \geq C_2(t)$ et $C_2(t) + p_{m_2,i} > \mu(t)$.

Remarque 11. *Lorsque $C_1(t) + p_{m_1,j} = C_2(t)$ alors le lemme 4.4, nous indique que $t_1 = (X_1, X_2 \cup \{j\})$ et $t_2 = (X_1 \cup X_2 \cup \{j\}, \{j\})$ sont deux couples candidats tels que $t \underset{j}{\rightarrow} t_1$ et $t \underset{j}{\rightarrow} t_2$. On observe que dans ce cas $C_1(t_1) = C_1(t_2)$ et $C_2(t_1) = C_2(t_2)$. Nous verrons dans la suite que t_1 et t_2 sont alors* équivalents.

Les deux lemmes suivants établissent des relations sur $\mu^*(t')$ en fonction de $\mu(t)$ et de la tâche j telle que $t \underset{j}{\rightarrow} t'$ (voir aussi la figure 4.4) :

Lemme 4.5. *Soit t et t' deux couples candidats tels que $t \underset{j}{\rightarrow} t'$. Alors*

$$\mu^*(t') \leq \max(\mu^*(t), C_2(t')) + p_{m_3,j}.$$

Démonstration. Soit σ une séquence qui respecte t et telle que $\mu^*(t) = C_{\max}$. Si on ajoute j à la fin de cette séquence, alors on observe que la date de fin opératoire de la troisième machine est $\max(\mu^*(t), C_2(t')) + p_{m_3,j}$, où la valeur de $C_2(t')$ est donnée au lemme 4.4. □

Lemme 4.6. *Soit t' un couple candidat. Alors il existe un couple candidat t et un j tel que $t \underset{j}{\rightarrow} t'$ et*

$$\mu^*(t') = \max(\mu^*(t), C_2(t')) + p_{m_3,j}.$$

Démonstration. Par définition de $\mu^*(t')$, il existe une séquence σ respectant t' et pour laquelle $\mu^*(t') = C_{\max}$. Notons j le dernier travail de la séquence σ. Considérons t un couple candidat quelconque tel que $t \underset{j}{\rightarrow} t'$.

Par contradiction, supposons que $\mu^*(t') < \max(\mu^*(t), C_2(t')) + p_{m_3,j}$. Considérons d'abord le cas où $C_2(t') \geq \mu^*(t)$, alors on obtient une contradiction car le travail j ne peut commencer sur la troisième machine qu'une fois celui-ci traité sur la deuxième machine. Donc nécessairement $\mu^*(t') \geq C_2(t') + p_{m_3,j}$. Par le lemme précédent, on établit l'égalité. Dans le cas où $C_2(t') < \mu^*(t)$, cela signifie que la séquence σ dont on supprime le dernier élément j est une séquence qui respecte t' et pour laquelle la date de fin de traitement sur la troisième machine est inférieure à $\mu^*(t)$. Ce qui est une contradiction sur la définition de $\mu^*(t)$. □

Finalement, de façon intuitive, des couples candidats sont *équivalents* s'ils correspondent à un ordonnancement des mêmes tâches, avec des dates de fin opératoires identiques sur les deux premières machines. Formellement, nous avons la définition suivante :

Définition 4.7. Soient $t = (X_1, X_2)$ et $t' = (X_1', X_2')$ deux couples candidats tels que $C_1(t) = C_1(t')$, $C_2(t) = C_2(t')$ et $X_1 \cup X_2 = X_1' \cup X_2'$, alors ces deux couples sont dits *équivalents*. On note $t \simeq t'$. Étant donné un couple t, on note $\mathcal{E}(t) = \{t' \text{ t.q. } t' \simeq t\}$ l'ensemble des couples candidats équivalents à t.

Lemme 4.7. *Soient $t = (X_1, X_2)$ et $t' = (X_1', X_2')$ deux couples candidats équivalents. Soit $j \in (\mathcal{J} \setminus (X_1 \cup X_2)$. Soient $\tilde{t}$ et $\tilde{t}'$ tels que $t \underset{j}{\rightarrow} \tilde{t}$ et $t' \underset{j}{\rightarrow} \tilde{t}'$. Alors $\tilde{t} \simeq \tilde{t}'$.*
De plus, si $\mu(t') = \min_{t'' \in \mathcal{E}(t)} \mu(t'')$ alors $\mu^*(\tilde{t}') \leq \mu^*(\tilde{t})$.

Démonstration. Soient $t = (X_1, X_2)$ et $t' = (X_1', X_2')$ deux couples candidats équivalents et soit j un élément quelconque de $\mathcal{J} \setminus (X_1 \cup X_2)$. Considérons $\tilde{t} = (\tilde{X}_1, \tilde{X}_2)$ et $\tilde{t}' = (\tilde{X}_1', \tilde{X}_2')$, les extensions respectives de t et t' par j. Puisque $t \simeq t'$ alors par définition, on a $C_1(t) = C_1(t')$ et $C_2(t) = C_2(t')$.

De plus, par le lemme 4.4, nous pouvons distinguer deux cas :

- si $C_1(t)+p_{m_1,j} \leq C_2(t)$ alors on a $C_1(\tilde{t}) = C_1(t)+p_{m_1,j} = C_1(\tilde{t'})$ et $C_2(\tilde{t}) = C_2(t)+p_{m_2,j} = C_2(\tilde{t'})$.
- si $C_1(t)+p_{m_1,j} \geq C_2(t)$ alors on a $C_1(\tilde{t}) = C_1(t)+p_{m_1,j} = C_1(\tilde{t'})$ et $C_2(\tilde{t}) = C_1(t)+p_{m_1,j}+p_{m_2,j} = C_2(\tilde{t'})$.

Par conséquent, on a bien $\tilde{t} \simeq \tilde{t'}$. Supposons maintenant que $\mu(t') = \min_{t'' \in \mathcal{E}(t)} \mu(t'')$. Par définition, on a $\mu^*(t') = \mu(t')$. De plus, par le lemme 4.5, on a $\mu^*(\tilde{t'}) \leq \max(\mu^*(t'), C_2(\tilde{t'})) + p_{m_3,j}$. Comme $\mu^*(t') = \mu(t') \leq \mu(t)$ et que $C_2(\tilde{t'}) = C_2(\tilde{t})$, on obtient $\mu^*(\tilde{t'}) \leq \mu^*(\tilde{t})$. □

Une conséquence immédiate du lemme 4.7 est qu'il est suffisant d'étendre seulement le couple t de $\mathcal{E}(t)$ pour lequel la valeur $\mu(t)$ est minimale.

Considérons à présent l'algorithme 13. Nous montrons qu'étant donné une collection de n travaux, il calcule un ordonnancement optimal de ceux-ci pour le problème de flow-shop à 3 machines $F_3||C_{\max}$.

Algorithme 13 : ExactF3Cmax$(\mathcal{J})$

Entrée : Une collection de n travaux $\mathcal{J}$ et leur durée de traitement sur les 3 machines.
Sortie : La durée opératoire minimum d'un ordonnancement pour le problème de flow-shop à 3 machines $F_3||C_{\max}$.

pour chaque *couple* (X_1, X_2) *t.q.* $X_1, X_2 \subseteq \mathcal{J}$ *et* $|X_1 \cap X_2| = 1$ **faire**
  $\mu(t) \leftarrow \infty$
  Ajouter (X_1, X_2) à $\mathcal{L}$

$t \leftarrow (\varnothing, \varnothing)$
$\mu(t) \leftarrow 0$
Ajouter t à $\mathcal{L}$

tant que $\mathcal{L}$ *est non vide* **faire**
  Extraire un couple $t = (X_1, X_2)$ de $\mathcal{L}$ de valeur $\mu(t)$ minimum
  si $X_1 \cup X_2 = \mathcal{J}$ **alors**
    $\mathcal{R} \leftarrow \mathcal{R} \cup \{t\}$
  sinon
    pour chaque *travail* $J_i \in \mathcal{J} \setminus (X_1 \cup X_2)$ **faire**
      si $C_1(t) + p_{m_1,i} < C_2(t)$ **alors**
        $(X'_1, X'_2) \leftarrow (X_1, X_2 \cup \{i\})$
      sinon
        $(X'_1, X'_2) \leftarrow (X_1 \cup X_2 \cup \{i\}, \{i\})$
      Soit $t' = (X'_1, X'_2)$
      $d \leftarrow \max(\mu(t), C_2(t')) + p_{m_3,i}$
      si $d < \mu(t')$ **alors**
        $\mu(t') \leftarrow d$

retourner le couple $t \in \mathcal{R}$ ayant la plus petite valeur $\mu(t)$.

Lemme 4.8. *Lorsqu'un couple* $t = (X_1, X_2)$ *est extrait de la liste* $\mathcal{L}$ *de l'algorithme* ***AlgoExactF3Cmax****, on a* $\mu^*(X_1, X_2) = \mu(X_1, X_2)$.

Démonstration. On montre l'invariant suivant :

« chaque couple $t = (X_1, X_2)$ extrait de $\mathcal{L}$ respecte $\mu^*(X_1, X_2) = \mu(X_1, X_2)$ ».

initialisation. On observe facilement que pour le couple $(X_1, X_2) = (\varnothing, \varnothing)$, sa valeur $\mu(X_1, X_2)$ est initialisée à 0. Il est immédiatement extrait de $\mathcal{L}$ et on a bien $\mu(X_1, X_2) = \mu^*(X_1, X_2) = 0$.

conservation. Supposons par contradiction que $t = (X_1, X_2)$ soit le premier couple extrait de $\mathcal{L}$ pour lequel on ait $\mu^*(X_1, X_2) \neq \mu(X_1, X_2)$. Clairement, $X_1 \cup X_2 \neq \varnothing$ puisque le couple $(\varnothing, \varnothing)$ est le premier généré durant l'étape d'initialisation (et pour lequel on a effectivement $\mu(\varnothing, \varnothing) = \mu^*(\varnothing, \varnothing)$).

Un ordonnancement optimal σ du couple t peut être obtenu par une séquence d'extensions (avec ajout d'une tâche à chaque étape) : $t_0 \to t_1 \to \cdots \to t$ où $t_0 = (X_1^0, X_2^0) = (\varnothing, \varnothing)$ et $t = (X_1^t, X_2^t) = (X_1, X_2)$. Dans cette séquence, notons $t_\ell = (X_1^\ell, X_2^\ell)$ le premier couple qui n'a pas encore été extrait de $\mathcal{L}$. Le couple qui précède t_ℓ dans la séquence d'extensions est $t_{\ell-1}$. Notons $k \in \mathcal{J}$ tel que $t_{\ell-1} \underset{k}{\to} t_\ell$.

Nous affirmons que $\mu(t_\ell) = \mu^*(t_\ell)$ lorsque t est extrait de $\mathcal{L}$. Pour cela, observons que $t_{\ell-1}$ a été extrait avant le couple t et donc, comme t est le premier couple pour lequel $\mu^*(t) \neq \mu(t)$, nous avons $\mu(t_{\ell-1}) = \mu^*(t_{\ell-1})$ lorsque $t_{\ell-1}$ a été extrait. À ce moment, la tâche k a été considérée et la valeur de $\mu(t_\ell)$ a été relaxée et le lemme 4.6 assure que $\mu(t_\ell) = \mu^*(t_\ell)$.

Nous pouvons à présent établir une contradiction. Puisque t_ℓ apparait avant t dans la séquence, nous avons $\mu(t_\ell) \leq \mu(t)$. En particulier, nous avons $\mu(t_\ell) = \mu^*(t_\ell) \leq \mu^*(t) \leq \mu(t)$. Mais comme t a été extrait de $\mathcal{L}$ avant t_ℓ, nous en déduisons que $\mu(t) = \mu(t_\ell)$ et donc $\mu(t) = \mu^*(t)$, ce qui contredit le choix du couple t.

Finalement, on observe qu'une fois qu'un couple t est extrait de $\mathcal{L}$ et ajouté à $\mathcal{R}$, la valeur de $\mu(t) = \mu^*(t)$ est minimum et ne peut donc jamais être modifiée. □

Nous pouvons finalement établir le temps d'exécution de l'algorithme 13 :

Théorème 4.9. *L'algorithme **ExactF3Cmax** s'exécute en temps $\mathcal{O}^*(3^n)$ au pire des cas.*

Démonstration. Notons tout d'abord que le nombre de couples candidats est au plus $n\,3^n$. La liste $\mathcal{L}$ est une file de priorité qui peut être implémentée par un *tas de Fibonacci* (voir l'ouvrage de Cormen *et al.* (2009) pour davantage de détail sur les tas).

On observe que chaque couple est inséré exactement une fois (à l'initialisation). Il s'ensuit que chaque couple est également extrait exactement une fois. Pour chaque couple extrait t, au plus n opérations de diminution de la valeur de $\mu(t')$, avec $t \to t'$ sont nécessaires. Par conséquent, l'algorithme peut être implémenté pour s'exécuter en temps et espace $\mathcal{O}^*(3^n)$. □

4.3 Trier-et-chercher

Nous continuons de considérer des problèmes d'ordonnancement, à une unique machine, avec des machines parallèles, ou dans le cas d'un flow-shop. À la section précédente, nous avons construit des algorithmes basés sur des schémas de *programmation dynamique*. À présent, nous allons nous intéresser à une autre approche appelée *trier-et-chercher* (en anglais, *sort-and-search*). Cette technique est essentiellement basée sur une partition de l'instance puis une combinaison des solutions. À la prochaine section, nous donnons une description de la technique, qui a d'abord été introduite par Horowitz et Sahni (1974) pour résoudre le problème SAC-À-DOS (dont la définition est rappelée à la section suivante) en temps et espace $\mathcal{O}^*(2^{n/2})$, puis améliorée par Schroeppel et Shamir (1981)

pour établir un compromis entre le temps d'exécution et l'espace utilisé. Bien que peu utilisée, cette technique est également présentée dans l'état de l'art de Woeginger (2001) et dans l'ouvrage de Fomin et Kratsch (2010). Nous avons notamment démontré l'utilité de cette technique pour résoudre quelques cas du problème (σ, ϱ) DOMINATION dans (Fomin *et al.*, 2009b) (voir aussi (Liedloff, 2007)). Dans son cadre le plus général, ce problème généralise plusieurs problèmes classiques, comme DOMINATION et ENSEMBLE STABLE.

Dans la suite, nous présentons la technique puis nous montrons qu'il est possible de l'étendre. Nous donnons ensuite des applications de cette extension à des problèmes d'ordonnancement.

4.3.1 Présentation de la technique

L'idée de l'approche *trier-et-chercher* consiste à stocker davantage de données afin de réduire le temps de calcul. Une approche triviale pour résoudre le problème classique du SAC-À-DOS demande un temps en $\mathcal{O}(2^n)$, en énumérant tous les sous-ensembles des n objets. Horowitz et Sahni (1974) proposent de diminuer cette complexité à $\mathcal{O}(2^{n/2})$. Pour cela, ils partitionnent arbitrairement l'instance $\mathcal{I}$ en deux instances $\mathcal{I}_1$ et $\mathcal{I}_2$. Puis ils énumèrent toutes les solutions possibles pour $\mathcal{I}_1$ et font de même pour $\mathcal{I}_2$. Finalement, une solution optimale pour $\mathcal{I}$ est obtenue par la combinaison d'une solution de $\mathcal{I}_1$ avec une solution de $\mathcal{I}_2$.

> **SAC-À-DOS**
>
> ***Entrée.*** Un ensemble de n objets $O = \{o_1, o_2, \ldots, o_n\}$ ayant chacun une valeur $v(o_i)$ et un poids $w(o_i)$, $1 \leq i \leq n$. Une capacité maximale W du sac à dos.
>
> ***Question.*** Déterminer un sous-ensemble d'objets $O' \subseteq O$ tel que $\sum_{o \in O'} w(o) \leq W$ et $\sum_{o \in O'} v(o)$ soit maximum.

Pour être plus précis, nous décrivons l'approche *trier-et-chercher* sur le problème SAC-À-DOS. Supposons que $O = \{o_1, o_2, \ldots, o_n\}$ soit l'ensemble des n objets. Cet ensemble est partitionné en $O_1 = \{o_1, \ldots, o_{\lceil n/2 \rceil}\}$ et en $O_2 = \{o_{\lceil n/2 \rceil + 1}, \ldots, o_n\}$ contenant chacun (à une unité près, dont on ne tiendra pas rigueur dans la suite) $\frac{n}{2}$ objets. À chaque sous-ensemble $O_j \subseteq O_1$, est associé le couple $(w(O_j), v(O_j))$ où $w(O_j) = sum_{o \in O_j} w(o)$ et $v(O_j) = sum_{o \in O_j} v(o)$. Ces couples sont stockés dans une table T_1. On procède de la même façon avec tous les sous-ensembles $O_k \subseteq O_2$ et en stockant les couples $(w(O_k), v(O_k))$ dans une table T_2. Les tables T_1 et T_2 contiennent chacune $2^{n/2}$ couples.

Les couples de T_2 sont ensuite triés par valeur de $w(O_k)$ croissante. Pour plus de simplicité, on renumérote ces couples (c'est-à-dire que $(w(O_k), v(O_k))$ devient l'élément de rang k dans la table T_2, par rapport aux valeurs des premières composantes). Pour chaque couple $(w(O_k), v(O_k))$, nous conservons l'indice $\ell_k \leq k$ du couple $(w(O_{\ell_k}), v(O_{\ell_k}))$ pour lequel la valeur $v(O_{\ell_k})$ est la plus grande. Puisque $\ell_k \leq k$ et que la table a été triée, on a $w(O_{\ell_k}) \leq w(O_k)$. Le tri peut être réalisé au moyen de *tri fusion* en temps $\mathcal{O}^*(\frac{n}{2}2^{n/2})$. De même, une fois la table triée, le calcul des indices ℓ_k, pour chaque k, peut s'effectuer en une passe en temps $\mathcal{O}^*(2^{n/2})$.

Finalement, une étape de recherche est menée : pour chaque couple $(w(O_j), v(O_j))$ de T_1, nous recherchons dans la table T_2 un couple $(w(O_k), v(O_k))$ tel que $w(O_j) + w(O_k) \leq W$ et $v(O_j) + v(O_k)$ soit le plus grand possible. La recherche du couple $(w(O_k), v(O_k))$ pour lequel $w(O_k)$ soit le plus grand possible, avec $w(O_k) \leq W - O(j)$, se fait par dichotomie en temps $\mathcal{O}(n/2)$; et la valeur maximum nous est donnée par $v(O_j) + v(O_{\ell_k})$. En itérant cette recherche sur les $2^{n/2}$ couples de T_1, la solution optimale est trouvée en temps $\mathcal{O}^*(\frac{n}{2}2^{n/2}) = \mathcal{O}^*(2^{n/2})$.

O	a	b	c	d	e	f
w	4	2	1	3	2	5
v	3	4	2	5	1	3

$O_1 = \{a, b, c\}$ $O_2 = \{d, e, f\}$

T_1	$\varnothing$	$\{a\}$	$\{b\}$	$\{c\}$	$\{a,b\}$	$\{a,c\}$	$\{b,c\}$	$\{a,b,c\}$
w	0	4	2	1	6	5	3	7
v	0	3	4	2	7	5	6	9

T_2	$\varnothing$	$\{e\}$	$\{d\}$	$\{f\}$	$\{d,e\}$	$\{e,f\}$	$\{d,f\}$	$\{d,e,f\}$
w	0	2	3	5	5	7	8	10
v	0	1	5	3	6	4	8	9
ℓ_k	1	2	3	3	5	5	7	8

j	$\varnothing$	$\{a\}$	$\{b\}$	$\{c\}$	$\{a,b\}$	$\{a,c\}$	$\{b,c\}$	$\{a,b,c\}$
ℓ_k	$\{d,f\}$	$\{d,e\}$	$\{e,f\}$	$\{d,f\}$	$\{d\}$	$\{d\}$	$\{d,e\}$	$\{e\}$
$w(O'_j) + w(O'_k)$	8	9	9	9	9	8	8	9
$v(O'_j) + v(O'_{\ell_k})$	8	9	10	10	12	10	12	10

FIGURE 4.5 – Un exemple d'application de l'approche *trier-et-chercher* pour le problème SAC-À-DOS sur l'instance $O = \{a, b, c, d, e, f\}$ et $W = 9$. La première table décrit l'entrée du problème avec les valeurs et poids de chacun des 6 objets. Les tables T_1 et T_2 sont telles que construites par l'approche. La dernière table donne le résultat de l'opération de *recherche* : pour chaque colonne j de T_1 est indiquée la colonne ℓ_k de T_2 telle que $w(O'_j) + w(O'_{\ell_k}) \leq W$ et $v(O'_j) + v(O'_{\ell_k})$ est maximum. La solution optimale est de valeur 12 et peut être obtenue en mettant dans le sac-à-dos les objets $\{a, b, d\}$ ou $\{b, c, d, e\}$.

Finalement, nous observons que la technique demande également un espace mémoire exponentiel, de l'ordre de $\mathcal{O}^*(2^{n/2})$. Schroeppel et Shamir (1981) montrent qu'il est possible d'améliorer cet espace mémoire à $\mathcal{O}^*(2^{n/4})$, tout en préservant le temps d'exécution. Un exemple tiré de Lenté *et al.* (2015) est présenté en détail à la figure 4.5.

Nous pouvons distinguer deux propriétés devant être vérifiées par les problèmes qui pourraient être résolus efficacement par cette approche :

1. les solutions doivent être suffisamment structurées (par l'emploi d'un tri, par exemple) afin de réaliser la recherche dans un temps raisonnable ;

2. les solutions partielles doivent se combiner en temps polynomial pour construire une solution de l'instance initiale.

Dans la suite, nous commençons par abstraire la technique *trier-et-chercher* puis nous la généralisons. Essentiellement, nous montrons qu'il est possible de considérer des problèmes à plusieurs contraintes et de continuer d'utiliser l'approche, à condition de considérer des structures de données un peu plus complexes.

4.3.2 Abstraction et généralisation de la technique

La description de la méthode telle que proposée par Horowitz et Sahni (1974) peut en fait s'appliquer à une classe de problèmes que nous appelons *problèmes à une seule contrainte* (PSC). Une fois cette abstraction définie, nous verrons à la section 4.3.2.2 qu'il est possible d'étendre l'approche pour traiter des *problèmes à plusieurs contraintes* (PPC).

4.3.2.1 Algorithme pour les problèmes à une seule contrainte

Nous définissons un problème PSC (problème à une seule contrainte) de la façon suivante.

Soient $A = \left(\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_{n_A}\right)$ une table de n_A vecteurs de dimension d_A et $B = \left((b_1, b'_1), (b_2, b'_2), \ldots, (b_{n_B}, b'_{n_B})\right)$ une table de n_B couples. Soient f et g deux fonctions de $\mathbb{R}^{d_A+1}$ dans $\mathbb{R}$, croissantes par rapport à leur dernière variable. Le problème PSC, dans sa version de minimisation[1], est défini par :

$$\begin{array}{l} \text{Minimiser } f(\vec{a}_j, b_k) \\ \text{s.c.} \\ \quad g(\vec{a}_j, b'_k) \geq 0 \\ \quad \vec{a}_j \in A \\ \quad (b_k, b'_k) \in B. \end{array}$$

Nous décrivons la résolution de ce problème par l'approche *trier-et-chercher*. Supposons que les éléments de la table B soient triés par ordre croissant selon les valeurs b'_k. Notons m la plus petite valeur prise par la fonction f (tout en respectant les contraintes) et m_j la plus petite valeur de f pour un vecteur $\vec{a}_j$ donné ($1 \leq j \leq n_A$). La valeur m est donc la solution optimale du PSC et on a :

$$\begin{aligned} m &= \min_{1 \leq j \leq n_A} \{m_j\} \\ m_j &= \min_{1 \leq k \leq n_B} \{f(\vec{a}_j, b_k) \ : \ g(\vec{a}_j, b'_k) \geq 0\}. \end{aligned}$$

En notant $G_j^{-1} = \{k \in [\![1, n_B]\!] \ : \ g(\vec{a}_j, b'_k) \geq 0\}$, nous pouvons encore exprimer m_j par :

$$m_j = \min_{k \in G_j^{-1}} \{f(\vec{a}_j, b_k)\}.$$

Nous utilisons maintenant les propriétés de la fonction g. Comme celle-ci est croissante par rapport à sa dernière variable, nous pouvons observer que soit l'ensemble G_j^{-1} est vide, soit il est égal à un intervalle d'entiers de la forme $[\![k_j, n_B]\!]$, pour un certain k_j.

Le cas où G_j^{-1} est vide est facile à détecter ; cela se produit si et seulement si $g(\vec{a}_j, b'_{n_B} < 0$. Si G_j^{-1} n'est pas vide, alors l'indice k_j est égal à $\min\{k \ : \ g(\vec{a}_j, b'_k) \geq 0\}$. La valeur de k_j peut ainsi être déterminée par une recherche dichotomique sur la table B qui, rappelons-le, est triée par ordre croissant de valeurs b'_k.

De plus, comme la fonction f est également croissante par rapport à sa dernière variable, nous établissons que

$$m_j = f(\vec{a}_j, \min\{b_\ell \ : \ \ell \in G_j^{-1}\}).$$

Pour tout k, $1 \leq k \leq n_B$, si nous notons $b_k^{\min} = \min_{k \leq \ell \leq n_B}\{b_\ell\}$, alors m_j est encore défini par :

$$m_j = f(\vec{a}_j, b_{k_j}^{\min})$$

où k_j est l'indice défini précédemment.

Les valeurs de $b_k^{\min}$ peuvent être calculées indépendamment de j par un parcours de la table B, de l'indice n_B à l'indice 1. Toutes ces considérations justifient que la méthode *trier-et-chercher* de l'algorithme 14 résout un PSC de façon optimale.

[1] La description et la résolution que nous faisons dans la suite s'applique également à une version où l'on cherche à maximiser la fonction objectif et à des fonctions décroissantes.

Algorithme 14 : TrierChercher-PSC $(A, B, d_1, \ldots, d_q)$

Entrée : Deux tables A et B. Deux fonctions f et g.
Sortie : Une solution optimale au PSC, c'est-à-dire les indices j^* et k^* tels que $\vec{a}_{j^*} \in A$ et (b_{k^*}, b'_{k^*}) soient une solution optimale au PSC.

2 $m \leftarrow \infty$
4 $(j^*, k^*) \leftarrow (0, 0)$
6 Trier et indexer la table B par ordre croissant des valeurs b'_k : la k-ème colonne correspond au couple (b_k, b'_k) ayant la k-ème plus petite valeur b'_k
8 À chaque colonne k de B, ajouter une troisième valeur définie par $b_k^{min} = b_k$ si $k = n_B$, et par $b_k^{min} = min(b_{k+1}^{min}, b_k)$ sinon
10 **pour tous les** $j \in [\![1, n_A]\!]$ **tels que** $g(\vec{a}_j, b'_{n_B}) \geq 0$ **faire**
12 Calculer $k_j = min\{k \in [\![1, n_B]\!] \mid g(\vec{a}_j, b'_k) \geq 0\}$
14 $m_{local} \leftarrow f(\vec{a}_j, b_{k_j}^{min})$
16 **si** $m_{local} < m$ **alors**
18 $m \leftarrow m_{local}$
20 $(j^*, k^*) \leftarrow (j, k_j)$
22 **retourner** m et (j^*, k^*)

Complexité au pire des cas

Il nous reste à démontrer que la complexité en temps et mémoire de l'algorithme est bornée par $\mathcal{O}^*(n_A + n_B)$. On observe que l'étape de tri requiert un temps $\mathcal{O}(n_B \log(n_B))$. L'étape 8 peut être implémentée un temps $\mathcal{O}(n_B)$ par un simple parcours. La boucle *pour* demande un temps $\mathcal{O}(n_A \log(n_B))$ grâce à une recherche dichotomique pour calculer k_j. Par conséquent, la complexité totale est en $\mathcal{O}((n_A + n_B) \log(n_B))$, sous réserve que les fonctions f et g puissent être calculées dans un temps *négligeable* par rapport à cette complexité. Notons de plus, qu'à cette complexité peut éventuellement s'ajouter le coût pour générer les tables A et B, qui occupent un espace $\mathcal{O}(n_A + n_B)$

Nous établissons ainsi le théorème suivant :

Théorème 4.10 (Lenté *et al.* (2013)). *Étant donné deux tables A et B de tailles respectives n_A et n_B, ainsi que deux fonctions f, g, croissantes par rapport à leur dernière variable, le problème* PSC *peut être résolu en temps $\mathcal{O}((n_A + n_B) \log(n_B))$.*

Exemple. *Si nous revenons au problème* Sac-à-Dos, *avec $O = \{o_1, o_2, \ldots, o_n\}$ l'ensemble d'objets de valeurs $v(o_i)$ et de poids $w(o_i)$, $1 \leq i \leq n$, et un poids maximum W, nous pouvons évidement modéliser ce problème par un* PSC.

Pour chaque sous-ensemble O_j de $O_1 = \{o_1, \ldots, o_{\lceil n/2 \rceil}\}$, nous pouvons construire le vecteur $\vec{a}_j = (v(O_j), w(O_j))$ de A où $v(O_j) = \sum_{o \in O_j} v(o)$ et $w(O_j) = \sum_{o \in O_j} w(o)$. Nous désignons par $v(\vec{a}_j)$ la première composante du vecteur $\vec{a}_j$ et par $w(\vec{a}_j)$ la seconde composante de celui-ci. De même, pour chaque sous-ensemble O_k de $O_2 = \{o_1, \ldots, o_{\lceil n/2 \rceil}\}$, nous construisons le couple $(b_k, b'_k) = (v(O_k), w(O_k))$ de B.

Soient les fonctions $f(\vec{a}_j, b_k)$ et $g(\vec{a}_j, b'_k)$ définies pour tout vecteur de $\vec{a}_j \in A$ et couple $(b_k, b'_k) \in B$ par $f(\vec{a}_j, b_k) = v(\vec{a}_j) + b_k$ et $g(\vec{a}_j, b'_k) = W - w(\vec{a}_j) - b'_k$.

On observe que la résolution de Sac-à-Dos *revient à résoudre le problème* PSC, *dans sa version de maximisation, avec la fonction f croissante par rapport à sa deuxième variable et la fonction g décroissante*

en sa deuxième variable. L'approche que nous venons de décrire autorise la résolution de ce problème en temps $\mathcal{O}^*(2^{n/2})$.

4.3.2.2 Algorithme pour les problèmes à plusieurs contraintes

La méthode *trier-et-chercher* décrite à la section précédente pour résoudre des problèmes PSC peut être étendue naturellement à des problèmes à plusieurs contraintes, que nous appelons PPC dans la suite. D'un certaine façon, si le problème SAC-À-DOS est un problème PSC, alors le problème SAC-À-DOS à plusieurs dimensions est un problème PPC. Commençons par définir ces problèmes PPC.

Soient $A = \left(\vec{a}_1, \vec{a}_2, \ldots, \vec{a}_{n_A}\right)$ une table de n_A vecteurs de dimension d_A et $B = \left(\vec{b}_1, \vec{b}_2, \ldots, \vec{b}_{n_B}\right)$ une table de n_B vecteurs $\vec{b}_k = (b_k^0, b_k^1, \ldots, b_k^{d_B})$ de dimension $d_B + 1$. Soient f et g_ℓ $(1 \leq \ell \leq d_B)$ $d_B + 1$ fonctions de $\mathbb{R}^{d_A+1}$ dans $\mathbb{R}$, croissantes par rapport à leur dernière variable. Nous définissons un problème PPC par :

$$
\begin{array}{l}
\text{Minimiser } f(\vec{a}_j, b_k^0) \\
\text{s.c.} \\
\quad g_\ell(\vec{a}_j, b_k^\ell) \geq 0 \qquad 1 \leq \ell \leq d_B \\
\quad \vec{a}_j \in A \\
\quad \vec{b}_k \in B.
\end{array}
$$

Pour résoudre ce genre de problèmes avec la méthode *trier-et-chercher*, le point clé consiste à déterminer des valeurs minimales à l'interieur d'*hyperrectangles*, c'est-à-dire dans des rectangles de dimension d, au lieu d'intervalles comme ce fut le cas pour la résolution des PSC. Pour cela, nous utilisons une structure de données appropriée, utilisée également en algorithmique géométrique : les *arbres d'intervalles* (en anglais, *range trees*). Ces arbres d'intervalles permettent en particulier de répondre à des *requêtes d'intervalles rectangulaires*. On référera le lecteur à l'ouvrage de Berg *et al.* (2008) (chapitre 5) où une présentation détaillée de cette structure de données est proposée.

Quelques mots sur les arbres d'intervalles

Soit P un ensemble de n points $x^j \in \mathbb{R}^d$ et soit Q une *requête d'intervalles rectangulaire* définie par $[y_1; y_1'] \times [y_2; y_2'] \times \cdots \times [y_d; y_d']$. Calculer la requête Q demande de déterminer les points x^j qui appartiennent à l'hyperrectangle défini par Q. Pour calculer cette requête *rapidement*, nous utilisons des *arbres d'intervalles* que nous pouvons définir de la façon suivante (nous reprenons la présentation succinte que nous avions donnée dans (Lenté *et al.*, 2015)).

Définition 4.8. Soit P un ensemble de n points de dimension d. Notons $P = \left\{x^j = (x_1^j, x_2^j, \ldots, x_d^j), j \in \{1, 2, \ldots, n\}\right\}$. L'arbre d'intervalles $\mathcal{T}^d(P)$ est construit récursivement : c'est un arbre binaire équilibré dont les clés, stockées aux feuilles de l'arbre $\mathcal{T}^d(P)$, sont les premières coordonnées des points x^j.

Pour chaque nœud $v \in \mathcal{T}^d(P)$, on définit l'ensemble $P(v)$ des points stockés aux feuilles du sous-arbre enraciné en v.

Si $d > 1$, alors à chaque nœud interne $v \in \mathcal{T}^d(P)$, on associe un arbre d'intervalles $\mathcal{T}^{d-1}(P'(v))$, où $P'(v)$ s'obtient des points de $P(v)$ restreints à leurs $d-1$ dernières coordonnées.

$$
\begin{aligned}
x^1 &= [6\,;35\,;62]\\
x^2 &= [10\,;24\,;45]\\
x^3 &= [1\,;40\,;51]\\
x^4 &= [7\,;25\,;60]\\
x^5 &= [8\,;21\,;80]\\
x^6 &= [3\,;46\,;55]\\
x^7 &= [4\,;29\,;63]\\
x^8 &= [14\,;37\,;50]
\end{aligned}
$$

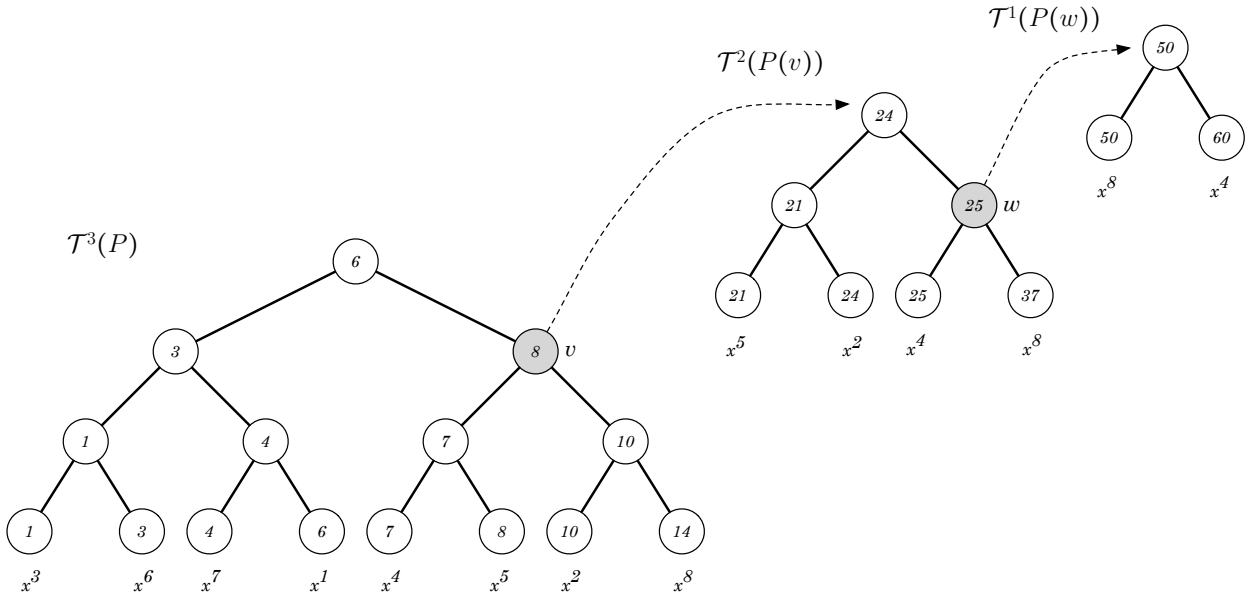


FIGURE 4.6 – Un exemple d'arbre d'intervalles pour l'ensemble de points $P = \{x^1, x^2, \ldots, x^8\}$ avec le détail de l'arbre associé au nœud v puis au nœud w. L'ensemble $P(v)$ correspond aux points $\{x^2, x^4, x^5, x^8\}$ et l'ensemble $P(w)$ correspond aux points $\{x^4, x^8\}$.

Notons qu'à chaque feuille de ces arbres construits récursivement, nous associons un point de P (représenté sur ses d coordonnées). Ceci est nécessaire pour résoudre la requête Q. Un exemple d'arbre d'intervalles est donné à la figure 4.6.

Pour résoudre une requête d'intervalles rectangulaire $Q = [y_1; y_1'] \times [y_2; y_2'] \times \cdots \times [y_d; y_d']$ sur un ensemble de points P, on commence par identifier dans $\mathcal{T}^d(P)$ les nœuds v dont les feuilles de leur sous-arbre enraciné appartiennent à l'intervalle pour leur première coordonnée ; c'est-à-dire dont les points x^j correspondant aux feuilles vérifient $x_1^j \in [y_1; y_1']$. Ensuite, pour chacun de ces nœuds v, il faut déterminer récursivement les nœuds $w \in \mathcal{T}^{d-1}(P_2(v))$ dont les feuilles du sous-arbre enraciné correspondent à des points x^j tels que $x_2^j \in [y_2; y_2']$. Ce processus est itéré jusqu'à l'exploration d'arbres d'intervalles $\mathcal{T}^1$. Les feuilles de ces derniers arbres permettent alors d'obtenir la réponse à la requête Q. L'exemple de la figure 4.6 est complété à la figure 4.7 pour traiter une requête.

Encore une fois, nous précisons que la création et la résolution de requêtes sont décrites précisément dans l'ouvrage de Berg *et al.* (2008) (chapitre 5). Nous n'en avons donné ici qu'une description informelle. Nous terminons ce paragraphe avec un théorème sur la complexité de ces deux opérations.

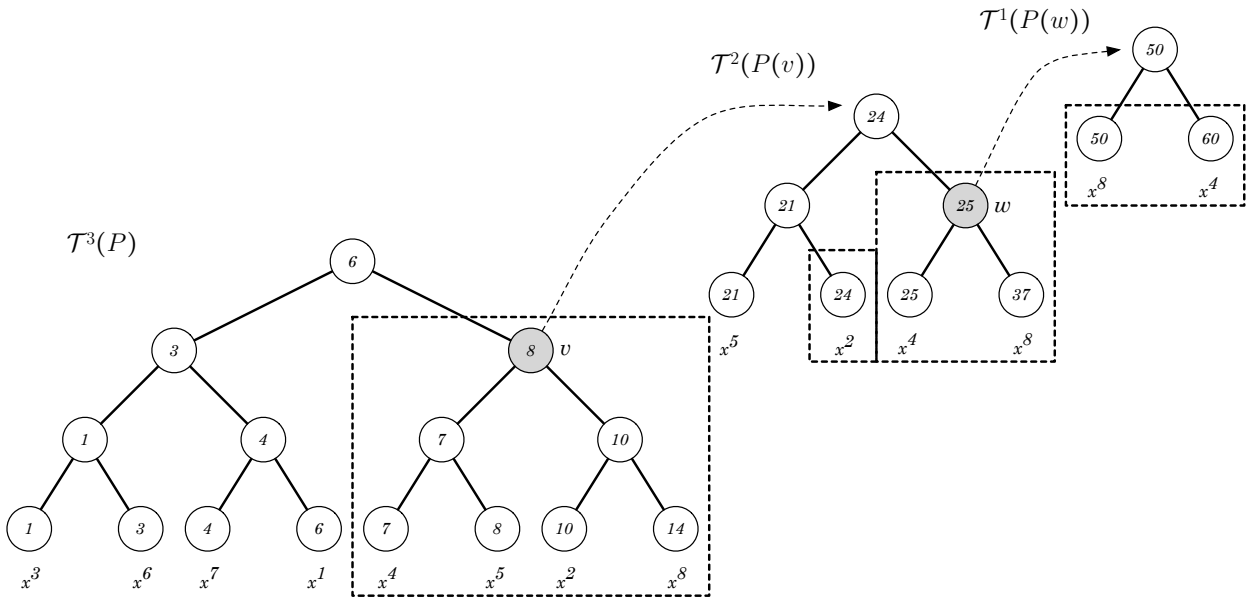


FIGURE 4.7 – L'exemple de la figure 4.6 sur lequel est résolue la requête $Q = [7;15] \times [22;37] \times [40,60]$. Les nœuds de $\mathcal{T}^3(P)$ dont les feuilles appartiennent à $[7;15]$ sont déterminés (il s'agit de l'unique nœud v sur la figure). Puis, dans $\mathcal{T}^2(P(v))$, les nœuds dont les feuilles appartiennent à $[22;37]$ sont déterminés. On obtient la feuille x^2 et le nœud w. Pour la feuille x^2, nous vérifions que sa troisième coordonnée appartient à $[40,60]$. On procède finalement sur $\mathcal{T}^1(P(w))$ pour déterminer les feuilles qui appartiennent à $[40,60]$. Les points x^2, x^8, x^4 sont finalement retournés.

Théorème 4.11 (voir par exemple Berg *et al.* (2008))**.** *Soit P un ensemble de n points dans un espace de dimension $d \geq 2$. Un arbre d'intervalles pour P nécessite un espace $\mathcal{O}(n \log^{d-1}(n))$ et peut être construit en temps $\mathcal{O}(n \log^{d-1}(n))$. Les points de P qui répondent à une requête d'intervalles rectangulaire peuvent être déterminés en temps*[2] *$\mathcal{O}(\log^d(n) + k)$, où k est le nombre de points retournés. De plus, dénombrer les points satisfaisant une requête peut être effectué en temps $\mathcal{O}(\log^d(n))$.*

Résolution d'un problème PPC

Nous allons utiliser des arbres d'intervalles pour résoudre les problèmes PPC avec l'approche *trier-et-chercher*, en lieu et place des tables que nous avions employées pour la résolution des problèmes PSC.

Essayons de généraliser l'approche présentée à la section 4.3.2.1. Étant donné un vecteur $\vec{a}_j$, nous lui associons d_B valeurs β_j^ℓ $(1 \leq \ell \leq d_B)$ définies par

$$\beta_j^\ell = \min\left\{b_k^\ell \ : \ 1 \leq k \leq n_B \quad \text{et} \quad g_\ell(\vec{a}_j, b_k^\ell) \geq 0\right\}.$$

Notons, s'il existe,

$$\beta_j = \min_{\vec{b}_k \in B}\left\{b_k^0 \ : \ (b_k^1, b_k^2, \ldots, b_k^{d_B}) \geq (\beta_j^1, \beta_j^2, \ldots, \beta_j^{d_B})\right\}.$$

Ainsi pour un vecteur $\vec{a}_j$ de A donné, la plus petite valeur de f pour lequel il existe un vecteur $\vec{b}_k$ de B satisfaisant les contraintes, est donné par $m_j = f(\vec{a}_j, \beta_j)$. Puisque les fonctions

[2] Ce temps peut être amélioré par un facteur logarithmique en utilisant une technique appelée *fractional cascading* (Berg *et al.*, 2008).

g_ℓ sont croissantes selon leur dernière variable, les β_j^ℓ peuvent facilement être calculés de façon indépendante, par dichotomie, en temps $\mathcal{O}(\log(n_B))$. Le calcul de β_j est moins facile.

Pour tout $\beta \in \{b_k^0 \ : \ k \in [\![1, n_B]\!]\}$, nous définissons

$$\mathcal{B}_\beta = \left\{\vec{b}_k \in B \ : \ b_k^0 \leq \beta \quad \text{et} \quad (b_k^1, b_k^2, \ldots, b_k^{d_B}) \geq (\beta_j^1, \beta_j^2, \ldots, \beta_j^{d_B})\right\}.$$

On a clairement

$$\mathcal{B}_\beta \neq \varnothing \Longleftrightarrow \beta_j \leq \beta$$

et plus précisément

$$\beta_j = \min\left\{\beta \ : \ \mathcal{B}_\beta \neq \varnothing\right\}.$$

Comme nous allons le voir, l'existence d'un tel β_j demande de calculer des requêtes rectangulaires sur les ensembles $\mathcal{B}_\beta$.

Nous construisons l'arbre d'intervalles $\mathcal{T}^{d_B+1}(B)$ à partir des vecteurs de la table B en temps $\mathcal{O}(n_B \log(n_B))$. Nous calculons également les d_B valeurs β_j^ℓ, par recherche dichotomique, en temps $\mathcal{O}(n_B \log(n_B))$. Nous pouvons décider si un ensemble $\mathcal{B}_\beta$ est vide en temps $\mathcal{O}(\log^{d_B+1}(n_B))$, en dénombrant les points satisfaisant la requête $\mathcal{Q}(\beta) : [-\infty; \beta] \times [\beta_j^1; +\infty] \times [\beta_j^2; +\infty] \times \cdots \times [\beta_j^{d_B}; +\infty]$.

Comme les $\mathcal{B}_\beta$ forment une chaîne d'ensembles (pour l'inclusion), la valeur de β_j peut être obtenue en triant initialement tous les b_k^0 puis en effectuant une recherche dichotomique : pour chaque valeur β testée, il suffit de résoudre la requête $\mathcal{Q}(\beta)$. La plus petite valeur β pour laquelle $\mathcal{B}_\beta \neq \varnothing$ (c'est-à-dire, pour laquelle la requête retourne au moins un vecteur de B) est la valeur β_j recherchée. Au total, la recherche complète peut être réalisée en temps $\mathcal{O}(\log^{d_B+2}(n_B))$ au pire des cas. (On rappelle que les vecteurs de B sont de taille $d_B + 1$.)

Nous donnons à l'algorithme 15 une description en pseudo-code de la méthode *trier-et-chercher* pour PPC, qui généralise celle de Horowitz et Sahni (1974) pour la résolution d'un PSC. Là encore, l'approche et les arguments exposés précédemment peuvent facilement s'adapter pour la résolution de PPC avec des fonctions f et g_ℓ décroissantes par rapport à leur dernière variable ou des problèmes de maximisation.

Complexité au pire des cas

La création de l'arbre d'intervalles nécessite un temps $\mathcal{O}(n_B \log^{d_B}(n_B))$. Le calcul des $b_{\max}^\ell$ demande un temps $\mathcal{O}(d_B\, n_B \log(n_B))$. Le calcul des β_j^ℓ peuvent s'implémenter en temps $\mathcal{O}(d_B \log(n_B))$ par une recherche dichotomique. La ligne 16 peut être implémentée par une recherche dichotomique dans l'ensemble des b_k^0 et, pour chaque valeur β, de calculer une requête $\mathcal{Q}(\beta)$, le tout en temps $\mathcal{O}(\log^{d_B+2}(n_B))$. Nous établissons ainsi le théorème suivant :

Théorème 4.12 (Lenté *et al.* (2013)). *Étant données deux tables A et B contenant respectivement n_A vecteurs de dimension d_A et n_B vecteurs de dimension d_B, ainsi que des fonctions f, g_ℓ ($1 \leq \ell \leq d_B$), croissantes par rapport à leur dernière variable, le problème* PPC *peut être résolu en temps $\mathcal{O}((1 + d_B)n_B \log(n_B) + n_A \log^{d_B+2}(n_B))$ et espace mémoire $\mathcal{O}(n_B \log^{d_B-1}(n_B))$.*

4.3.3 Application à des problèmes d'ordonnancement

Dans (Lenté *et al.*, 2011), la méthode *trier-et-chercher* nous a permis de résoudre en temps et espace $\mathcal{O}^*(2^{n/2}) = \mathcal{O}(1.4143^n)$ trois problèmes d'ordonnancement : $P_2||C_{\max}$, $1|d_i|\sum w_i U_i$ et $F_2||C_{\max}^k$

Algorithme 15 : TrierChercher-PPC $(A, B, d_1, \ldots, d_q)$

Entrée : Deux tables A et B contenant respectivement n_A vecteurs de dimension d_A et n_B vecteurs de dimension d_B. Des fonctions f et g_ℓ, $1 \leq \ell \leq d_B$.
Sortie : La valeur optimale de la fonction objectif du PPC.

Créer l'arbre d'intervalle $\mathcal{T}^{d_B+1}(B)$
pour tous les $\ell \in [\![1, d_B]\!]$ **faire**
  $b^\ell_{\max} \leftarrow \max\{b^\ell_k \ : \ 1 \leq k \leq n_B\}$
$m \leftarrow \infty$
pour tous les $j \in [\![1, n_A]\!]$ **tels que** $\forall \ell \in [\![1, d_B]\!],\ g_\ell(\vec{a}_j, b^\ell_{\max}) \geq 0$ **faire**
  pour tous les $\ell \in [\![1, d_B]\!]$ **faire**
    $\beta^\ell_j \leftarrow \min\left\{b^\ell_k \ : \ 1 \leq k \leq n_B \quad \text{et} \quad g_\ell(\vec{a}_j, b^\ell_k) \geq 0\right\}$
  $\beta_j \leftarrow \min\{\beta \ : \ \mathcal{B}_\beta \neq \varnothing\}$
  $m_j \leftarrow f(\vec{a}_j, \beta_j)$
  si $m_j < m$ **alors**
    $m \leftarrow m_j$
retourner m

(pour ce dernier problème, on cherche à minimiser la date de fin du travail en position k dans un problème de flow-shop à deux machines). Des propriétés structurelles de ces problèmes permettent de les exprimer comme des PSC. Dans la suite, nous considérons deux problèmes d'ordonnancement qui peuvent s'exprimer comme des PPC : le problème $P||C_{\max}$ et le problème $P|d_i|\sum w_i U_i$ pour lesquels nous appliquons également l'approche *trier-et-chercher*.

4.3.3.1 Le problème $P||C_{\max}$

Le problème $P||C_{\max}$ peut se définir ainsi : nous avons n tâches à ordonnancer sur m machines parallèles identiques. Chaque tâche doit être traitée par une et une seule machine. Pour chaque tâche i, on connaît sa durée opératoire p_i. On note C_j la date de fin de la dernière tâche traitée par la machine j. L'objectif consiste à déterminer un ordonnancement des n tâches de sorte à minimiser le *makespan*, c'est-à-dire minimiser $C_{\max} = \max_{1 \leq j \leq m} C_j$. Ce problème est connu comme étant NP-difficile. Dans (Lenté *et al.*, 2014), nous proposions un algorithme en $\mathcal{O}(2.0801^n)$ pour le résoudre dans le cas de $m = 3$ machines. Comme nous allons le voir, cette complexité peut être ramenée à $\mathcal{O}(1.7321^n)$ par une réduction à un problème PPC. Pour faire cette réduction, nous avons besoin de quelques résultats préliminaires que nous commençons par décrire. Nous verrons ensuite la mise en oeuvre de la méthode *trier-et-chercher*.

Résultats préliminaires

Nous considérons un ordonnancement s comme un ensemble de m séquences, chacune correspondant à l'une des m machines. Une séquence donnée est un ensemble de tâches traitées consécutivement (c'est-à-dire sans temps mort) sur la machine correspondante. Nous notons $P_\ell(s)$ la somme des durées opératoires des tâches traitées par la machine ℓ dans l'ordonnancement s (avec $1 \leq \ell \leq m$). Soit $P(s)$ la somme des durées opératoires de toutes les tâches ordonnancées dans s. On définit

$$\delta_\ell(s) = P_m(s) - P_\ell(s)$$

comme la différence entre la charge de travail de la dernière machine et celle de la machine ℓ. La propriété suivante établit des relations entre ces valeurs.

Propriété 4.13. *Les deux relations suivantes sont respectées :*

$$P(s) = \sum_{\ell=1}^{m} P_\ell(s),$$
$$\sum_{\ell=1}^{m-1} \delta_\ell(s) = m\,P_m(s) - P(s).$$

Puisque toutes les machines sont identiques, sans perte de généralité nous supposons (à une renumérotation près des machines) que le *makespan* est donné par la dernière machine. Nous notons O_m l'ensemble des ordonnancements tels que la valeur du *makespan* soit obtenu de la machine m.

Supposons que s et σ soient deux ordonnancements partiels disjoints, c'est-à-dire qu'aucune tâche n'apparaît à la fois dans s et dans σ. Alors le *makespan* de la concaténation de ces deux ordonnancements, notée $s \cdot \sigma$, est défini par

$$C_{\max}(s \cdot \sigma) = \max_{1 \le \ell \le m} \big(P_\ell(s) + P_\ell(\sigma)\big).$$

Propriété 4.14. *Le* makespan *de l'ordonnancement $s \cdot \sigma$ est donné par les travaux ordonnancés sur la machine m, si et seulement si la condition suivante est vérifiée :*

$$\forall \ell \in [\![1, m-1]\!], \quad \delta_\ell(s) + \delta_\ell(\sigma) \ge 0.$$

De plus, si cette condition est vérifiée, on a $C_{\max}(s \cdot \sigma) = P_m(s) + P_m(\sigma)$, que l'on peut encore réécrire comme

$$C_{\max}(s \cdot \sigma) = \frac{1}{m}\Big(P(s) + P(\sigma) + \sum_{\ell=1}^{m-1} \big(\delta_\ell(s) + \delta_\ell(\sigma)\big)\Big). \tag{4.3}$$

Cette dernière égalité se vérifie très aisément en utilisant les définitions de $\delta_\ell(s)$ et $\delta_\ell(\sigma)$.

Nous pouvons maintenant exprimer une propriété pour compléter optimalement un ordonnancement partiel donné.

Propriété 4.15. *Pour tout ordonnancement partiel s, l'ordonnancement $s \cdot \sigma$ est optimal parmi les ordonnancements de O_m commençant par s si et seulement si σ est une solution optimale du problème suivant :*

$$\begin{aligned} &\textit{Minimiser} \sum_{\ell=1}^{m-1} \delta_\ell(\sigma) \\ &\textit{s.c.} \\ &\quad \forall \ell \in [\![1, m-1]\!], \quad \delta_\ell(s) + \delta_\ell(\sigma) \ge 0. \end{aligned}$$

Démonstration. L'équation 4.3 donne la valeur du *makespan* de la concatenation des ordonnancements s et σ. Observons que s et σ forment une partition de l'ensemble des tâches, puisque $s \cdot \sigma$ est un ordonnancement de O_m. Ainsi, dans l'équation 4.3, le terme $P(s) + P(\sigma)$ est une constante égale à la somme de toutes les durées opératoires. De plus comme s est fixé, les valeurs de $\delta_\ell(s)$ sont également fixées et connues pour tout $1 \le \ell \le m-1$. Le seul terme variable à minimiser est donc $\sum_{\ell=1}^{m-1} \delta_\ell(\sigma)$. Finalement, les $m-1$ contraintes assurent que le *makespan* est effectivement donné par la machine m. □

Finalement, nous en déduisons la propriété suivante :

Propriété 4.16. *Un ordonnancement $s \cdot \sigma \in O_m$ est optimal si et seulement si s et σ sont une solution optimale du problème :*

$$\begin{aligned} &\textit{Minimiser} \sum_{\ell=1}^{m-1} \delta_\ell(s) + \delta_\ell(\sigma) \\ &\textit{s.c.} \\ &\quad \forall \ell \in [\![1, m-1]\!], \quad \delta_\ell(s) + \delta_\ell(\sigma) \geq 0. \end{aligned}$$

Nous pouvons à présent définir le PPC à résoudre. Nous le décrivons dans le prochain paragraphe où nous mettons en œuvre de la méthode *trier-et-chercher* pour résoudre le problème $P||C_{\max}$.

Mise en oeuvre de la méthode trier-et-chercher

Soit $\mathcal{J}$ un ensemble de n tâches. Nous pouvons partitionner $\mathcal{J}$ en m sous-ensembles E_ℓ ($1 \leq \ell \leq m$). Nous notons $\epsilon = (E_1, E_2, \ldots, E_m)$ une telle partition. Par abus de langage, nous identifions une tâche et son indice. Notons $I_1 = \{1, \ldots, \lfloor \frac{n}{2} \rfloor\}$ le sous-ensemble constitué des $\lfloor \frac{n}{2} \rfloor$ premières tâches de $\mathcal{J}$ et $I_2 = \{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$ le sous-ensemble constitué des $\lceil \frac{n}{2} \rceil$ dernières tâches de $\mathcal{J}$.

À chaque partition en m sous-ensembles $\epsilon_1^j = (E_{1,1}^j, E_{1,2}^j, \ldots, E_{1,m}^j)$ de I_1 (où $1 \leq j \leq m^{|I_1|}$) nous associons un ordonnancement s_1^j qui contient la séquence des tâches de $E_{1,\ell}^j$ sur la machine ℓ, avec $1 \leq \ell \leq m$. De façon similaire, à chaque m-partition ϵ_2^k de I_2, nous associons un ordonnancement s_2^k. Finalement, à chaque couple $(\epsilon_1^j, \epsilon_2^k)$, nous associons la valeur $F^{j\,k}$ définie par

$$F^{j\,k} = \frac{1}{m}\left(P + \sum_{\ell=1}^{m-1} \delta_\ell(s_1^j) + \sum_{\ell=1}^{m-1} \delta_\ell(s_2^k)\right)$$

où $P = \sum_{i=1}^{n} p_i$ est la somme des durées opératoires des n tâches. On observe que par l'équation 4.3, le *makespan* de l'ordonnancement $s_1^j \cdot s_2^k$ est donné par $F^{j\,k}$. La figure 4.8 présente une décomposition en deux ordonnancements pour un problème à 3 machines.

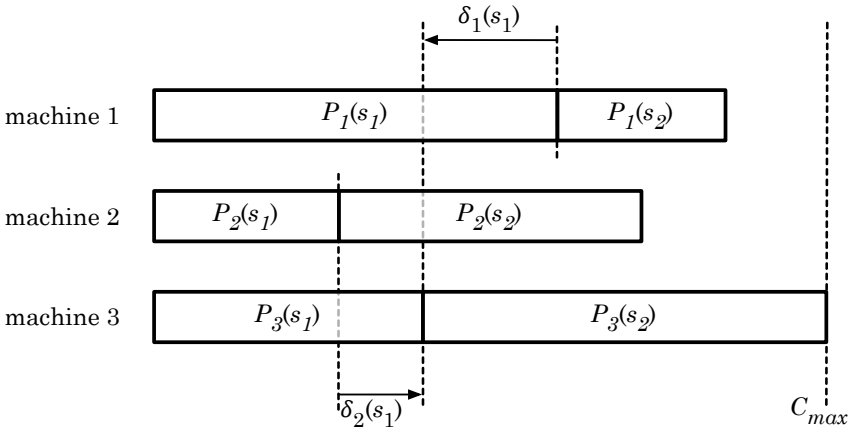


FIGURE 4.8 – Un exemple de décomposition pour un problème à 3 machines.

Nous pouvons maintenant exprimer le problème $P||C_{\max}$ comme un problème PPC.

Propriété 4.17. *On a*

$$\min_{s\in O_m} C_{\max}(s) = \min_{j,k} \left\{ F^{j\,k} \ : \ \forall \ell \in [\![1, m-1]\!], \delta_\ell(s_1^j) + \delta_\ell(s_2^k) \geq 0 \right\}.$$

Nous pouvons ainsi instancier le problème PPC puis le résoudre avec la méthode *trier-et-chercher*. Pour $1 \leq j \leq m^{\lfloor \frac{n}{2} \rfloor}$ et $1 \leq k \leq m^{\lceil \frac{n}{2} \rceil}$, on définit :

$$\begin{cases} \vec{a}_j & = & \left(\delta_1(s_1^j),\ \delta_2(s_1^j),\ \cdots,\ \delta_{m-1}(s_1^j)\right) \\ \vec{b}_k & = & \left(\sum_{\ell=1}^{m-1} \delta_\ell(s_2^k),\ \delta_1(s_2^k),\ \delta_2(s_2^k),\ \cdots,\ \delta_{m-1}(s_2^k)\right) \\ f(\vec{a}_j, b_k^0) & = & \frac{1}{m}\left(P + \sum_{\ell=1}^{m-1} \delta_\ell(s_1^j) + \sum_{\ell=1}^{m-1} \delta_\ell(s_2^k)\right) \\ g_\ell(\vec{a}_j, b_k^\ell) & = & \delta_\ell(s_1^j) + \delta_\ell(s_2^k) \quad \text{pour } 1 \leq \ell \leq m-1 \end{cases}$$

On vérifie facilement que les fonctions f et g_ℓ sont croissantes par rapport à leur dernière variable, ce qui nous permet d'appliquer l'approche de la section 4.3.2.2.

La complexité en temps est en $\mathcal{O}((1+d_B)\, n_B \log(n_B) + n_A \log^{d_B+2}(n_B))$ avec pour ce problème, $n_A = n_B = m^{n/2}$ et $d_B = m-1$. Ce qui nous donne le théorème :

Théorème 4.18 (Lenté *et al.* (2013))**.** *Le problème $P||C_{\max}$ peut être résolu en temps $\mathcal{O}^*(m^{n/2}\,(\frac{n}{2})^{m+1}\,\log^{m+1}(m))$ et espace $\mathcal{O}^*(m^{n/2}\,(\frac{n}{2})^{m-2}\,\log^{m-2}(m))$. Dès que le nombre $m \geq 2$ de machines est fixé, ces complexités deviennent $\mathcal{O}^*(m^{n/2})$. En particulier, le problème $P_3||C_{\max}$ peut être résolu en temps $\mathcal{O}(1.7321^n)$.*

La table 4.1 donne quelques temps d'exécution selon le nombre de machines.

nombre m de machines	2	3	4	5	6	7	8	9
base de l'exponentielle	1.4143	1.7321	2	2.2361	2.4495	2.6458	2.8285	3

TABLE 4.1 – Base de l'exponentielle du temps d'exécution de l'approche *trier-et-chercher* pour la résolution du problème $P_m||C_{\max}$ à m machines parallèles identiques.

4.3.3.2 Le problème $P|d_i|\sum w_iU_i$

Pour ce problème, on se donne à nouveau n tâches et m machines parallèles identiques. Pour chaque tâche i, nous connaissons sa durée opératoire p_i, une date de fin souhaitée d_i ainsi qu'une pénalité de retard w_i. Chaque tâche doit être affectée à l'une des machines de sorte à minimiser le nombre pondéré de travaux en retard.

$P|d_i|\sum w_iU_i$

Entrée. Un ensemble de n tâches et de m machines parallèles identiques ; pour chaque tâche, sa durée opératoire p_i, sa date de fin souhaitée d_i et une pénalité de retard w_i.

> ***Question.*** Déterminer un ordonnancement des tâches de sorte à minimiser $\sum_{1\leq i\leq n} w_i\, U_i$, où U_i est égal à 1 si la tâche i termine après la date d_i, et $U_i = 0$ sinon.

Ce problème est connu pour être NP-difficile (Brucker, 2007) et dans (Lenté *et al.*, 2011) nous avons proposé un algorithme en temps et espace $\mathcal{O}(2.0801^n)$ pour le résoudre dans le cas de $m = 2$ machines. Nous montrons dans la suite que cette complexité peut être améliorée en formulant le problème comme un PPC.

Résultats préliminaires

Nous commençons par donner quelques propriétés. Certaines de ces propriétés se restreignent au problème à une seule machine. Ces propriétés nous sont tout de même utiles car dès que les travaux auront été affectés à l'une des m machines, alors il nous restera m problèmes à une seule machine à résoudre.

Notons aussi que nous utilisons grandement la particularité de la fonction objectif. Celle-ci ne tient pas compte de la *quantité* retard de chaque tâche : soit une tâche est en retard (c'est-à-dire, $C_i > d_i$) et quelque soit son retard, cela implique un coût de w_i dans l'évaluation de la fonction objectif, soit le travail est en avance ($C_i \leq d_i$).

Dans la suite nous posons $L_{\max} = \max_{1\leq i\leq n}(C_i - d_i)$. Notons que le problème consistant à minimiser $L_{\max}$ pour une unique machine peut être résolu en temps polynomial grâce à une règle communément appelée « EDD » (*earliest due date first*). Pour cela, Jackson (1955) a montré qu'il suffit d'ordonner les tâches selon leur date de fin souhaitée d_i croissante.

Propriété 4.19. *Supposons que s soit un ordonnancement d'un ensemble de travaux affectés à l'une des machines. Nous avons les propriétés suivantes :*

1. $L_{\max}(s) \leq 0 \iff \sum_{i=1}^{n} w_i U_i = 0$.
2. *Soit $L_{\max}(s|t)$ le retard maximum dans la séquence s si elle commence exactement au temps t. Alors $L_{\max}(s) = L_{\max}(s|0)$ et $L_{\max}(s|t) = t + L_{\max}(s)$.*
3. *Il existe un ordonnancement optimal s^* pour lequel tous les travaux en avance sont ordonnancés avant ceux en retard. De plus, les travaux en avance sont ordonnancés selon la règle EDD.*
4. *Supposons qu'un ordonnancement s puisse être décomposé en deux séquences s_1 et s_2 telles que $s = s_1 \cdot s_2$. Alors $L_{\max}(s_1 \cdot s_2) = \max\{L_{\max}(s_1), C_{\max}(s_1) + L_{\max}(s_2)\}$.*
5. *Si $s = s_1 \cdot s_2$ et tous les travaux de s_1 sont en avance, alors on a $L_{\max}(s_1 \cdot s_2) \leq 0 \iff L_{\max}(s_2) \leq -C_{\max}(s_1)$.*

Démonstration. Le premier résultat provient directement de la définition de $L_{\max}$ et $\sum w_i U_i$. De même, le second résultat peut être déduit de la définition de $L_{\max}$. La troisième propriété peut être vérifiée en utilisant un argument d'échange sur les tâches : si s^* est un ordonnancement optimal ne satisfaisant pas la propriété, alors on peut toujours placer les travaux en retard à la fin de l'ordonnancement s^*, puisque cela n'augmente pas la valeur de la fonction objectif $\sum w_i U_i$. De plus, les travaux en avance peuvent être réarrangés selon la règle EDD. Ces travaux resteront en avance et leur valeur $C_i - d_i$ restera inférieure ou égale à 0. La quatrième propriété se déduit de la seconde et du fait que la séquence s_2 commence au temps $C_{\max}(s_1)$. Ce résultat nous permet d'en déduire facilement la cinquième propriété. □

Considérons le problème à m machines pour lequel nous établissons la propriété essentielle suivante :

Propriété 4.20. *Il existe un ordonnancement optimal pour lequel tous les travaux en retard sont affectés à la première machine et tous les travaux en avance, affectés à la première machine, sont ordonnancés avant ceux en retard. Les travaux en avance sur chaque machine sont ordonnés selon la règle EDD.*

Démonstration. Il est aisé de voir que les travaux en retard peuvent être regroupés sur l'une des machines, après les travaux en avance de cette machine. Cette propriété est aussi impliquée par le troisième point de la propriété 4.19. □

Notons que la propriété 4.20 et le troisième point de la propriété 4.19 ont été utilisés pour construire un algorithme pseudo-polynomial pour le problème à une machine $1|d_i|\sum w_iU_i$ par Lawler et Moore (1969).

Mise en oeuvre de la méthode trier-et-chercher

Supposons que les travaux soient numérotés selon leur date de fin souhaitée croissante. Soit $\mathcal{J}$ l'ensemble des travaux. Soit $I_1 = \{1, \dots, \lfloor \frac{n}{2} \rfloor\}$ l'ensemble des $\lfloor \frac{n}{2} \rfloor$ premiers travaux de $\mathcal{J}$ et soit $I_1 = \{\lfloor \frac{n}{2} \rfloor + 1, \dots, n\}$ les $\lceil \frac{n}{2} \rceil$ derniers travaux.

Pour toute $(m+1)$-partition $\epsilon_1^j = (E_1^{1,j}, E_1^{2,j}, \dots, E_1^{m+1,j})$ de I_1 (où $1 \le j \le (m+1)^{|I_1|}$) nous lui associons un ordonnancement s_1^j composé de $m+1$ séquences $(s_1^{1,j}, s_1^{2,j}, \dots, s_1^{m+1,j})$. Chaque séquence $s_1^{\ell,j}$ contient les tâches de l'ensemble $E_1^{\ell,j}$ ordonnées selon la règle EDD ($1 \le \ell \le m+1$). Les séquences $s_1^{1,j}$ à $s_1^{m,j}$ contiennent les travaux qui sont traités en avance sur les machines 1 à m. La séquence $s_1^{m+1,j}$ contient les travaux qui sont traités en retard, et donc affectés à la première machine. Nous procédons de façon similaire avec la $(m+1)$-partition $\epsilon_2^k = (E_2^{1,k}, E_2^{2,k}, \dots, E_2^{m+1,k})$ de I_2 (où $1 \le k \le (m+1)^{|I_2|}$). À chaque couple $(\epsilon_1^j, \epsilon_2^k)$, nous associons l'ordonnancement $o^{j\,k}$ obtenu de la concatenation des séquences $s_1^{1,j}$, $s_2^{1,k}$, $s_1^{m+1,j}$ et $s_2^{m+1,k}$ sur la machine 1, et de la concaténation des séquences $s_1^{\ell,j}$ et $s_2^{\ell,k}$, sur la machine ℓ ($2 \le \ell \le m$) (voir par exemple la figure 4.9). Pour tout couple $(\epsilon_1^j, \epsilon_2^k)$, nous définissons $F^{j\,k} = W(s_1^{m+1,j}) + W(s_2^{m+1,k})$ où $W(\sigma) = \sum_{i \in \sigma} w_i$ est la somme des poids w_i des tâches de σ.

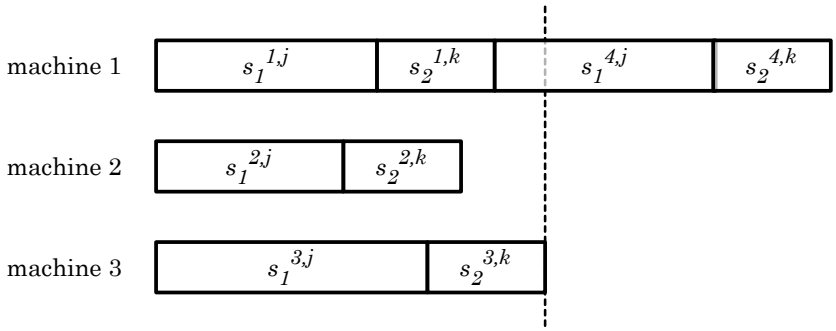


FIGURE 4.9 – Un exemple de décomposition pour le problème $P_3|d_i|\sum w_iU_i$ à 3 machines.

Soit $\mathcal{G}$ l'ensemble des couples défini par $\mathcal{G} = \left\{(j,k) \in [\![1, (m+1)^{|I_1|}]\!] \times [\![1, (m+1)^{|I_2|}]\!] \ : \ \forall \ell \in [\![1, m]\!], L_{\max}(s_1^{\ell,j} \cdot s_2^{\ell,k}) \le 0\right\}$. L'ensemble $\mathcal{G}$ représente l'ensemble des couples (j,k) pour lesquels la

concatenation des séquences $s_1^{\ell,j}$ et $s_2^{\ell,k}$ ne contient pas de travaux en retard pour tout $\ell \in [\![1,m]\!]$. Nous avons les deux propriétés suivantes :

Propriété 4.21. *Pour tout couple $(j,k) \in \mathcal{G}$, $F^{j\,k} \geq \sum w_i U_i(o^{j\,k})$ avec $U_i(o^{j\,k}) = 1$ si pour le travail i on a $C_i > d_i$ dans l'ordonnancement $o^{j\,k}$, et égal à 0 sinon.*

Démonstration. Cette propriété est correcte puisque les tâches des séquences $s_1^{\ell,j}$ et $s_2^{\ell,k}$ sont en avance, pour tout $\ell \in [\![1,m]\!]$. Notons qu'on n'a pas nécessairement l'égalité car, dans $o^{j\,k}$, il se peut que des travaux de s_1^{m+1} ou s_2^{m+1} soient en avance, bien qu'ils aient été affectés à des séquences de travaux *a priori* en retard. □

Propriété 4.22. *On a $\min_{(j,k)\in\mathcal{G}} F^{j\,k} = \min_\sigma \sum w_i U_i(\sigma)$.*

Démonstration. La propriété 4.21 nous permet d'établir que $\min_{(j,k)\in\mathcal{G}} F^{j\,k} \geq \min_\sigma \sum w_i U_i(\sigma)$. Pour montrer l'égalité, considérons σ^* un ordonnancement optimal satisfaisant le troisième point de la propriété 4.19. Soit $\sigma_1^{\ell^*}$ (respectivement $\sigma_2^{\ell^*}$) une sous-séquence des travaux en avance dans σ^* appartenant à I_1 (respectivement à I_2) sur la machine ℓ, avec $1 \leq \ell \leq m$. Puisque les travaux sont numérotés selon la règle EDD, il existe un couple (j^*,k^*) tel que $\sigma_1^{\ell^*} = s_1^{\ell,j^*}$ et $\sigma_2^{\ell^*} = s_2^{\ell,j^*}$. De plus, comme il n'y a pas de travaux en retard dans la séquence $\sigma_1^{\ell^*} \cdot \sigma_2^{\ell^*}$, le couple (j^*,k^*) appartient à $\mathcal{G}$. Les travaux n'appartenant ni à $\sigma_1^{l^*}$ ni à $\sigma_2^{l^*}$ sont par définition en retard, et donc $F^{j^*\,k^*} = \sum w_i U_i(\sigma^*)$. □

En se servant de la décomposition de l'ordonnancement (voir la figure 4.9) et des diverses propriétés démontrées, il s'ensuit que pour résoudre le problème $P|d_i|\sum w_i U_i$, il est suffisant de calculer $\min_{(j,k)\in\mathcal{G}} F^{j\,k}$. Cela peut se réécrire comme le problème PPC suivant :

$$\min\Big\{W(s_1^{m+1,j}) + W(s_2^{m+1,k}) \;:\; (j,k) \in [\![1,(m+1)^{|I_1|}]\!] \times [\![1,(m+1)^{|I_2|}]\!] \quad \text{et} \\ \forall \ell \in [\![1,m]\!], L_{\max}(s_1^{\ell,j}) \leq 0 \text{ et } L_{\max}(s_2^{\ell,k}) \leq -C_{\max}(s_1^{\ell,j})\Big\}.$$

Nous pouvons donc formuler notre problème $P|d_i|\sum w_i U_i$ comme un PPC et appliquer la méthode *trier-et-chercher* de la section 4.3.2.2. Pour $1 \leq j \leq m^{\lfloor \frac{n}{2} \rfloor}$ et $1 \leq k \leq m^{\lceil \frac{n}{2} \rceil}$, l'instance de ce PPC est donnée par :

$$\begin{cases} \vec{a}_j & = & \big(W(s_1^{m+1,j}), L_{\max}(s_1^{1,j}), C_{\max}(s_1^{1,j}), \ldots, L_{\max}(s_1^{m,j}), C_{\max}(s_1^{m,j})\big) \\ \vec{b}_k & = & \big(W(s_2^{m+1,k}), L_{\max}(s_2^{1,k}), \ldots, L_{\max}(s_2^{m,k})\big) \\ f(\vec{a}_j, b_k^0) & = & W(s_1^{m+1,j}) + W(s_2^{m+1,k}) \\ g_\ell(\vec{a}_j, b_k^\ell) & = & \begin{cases} 1 & \text{si } L_{max}(s_1^{\ell,j}) \leq 0 \text{ et } L_{\max}(s_2^{\ell,k}) \leq -C_{\max}(s_1^{\ell,j}) \\ -1 & \text{sinon} \end{cases} \quad \text{pour } 1 \leq \ell \leq m. \end{cases}$$

Notons que les fonctions g_ℓ sont décroissantes par rapport à leur dernière variable. Cela nécessite donc de légèrement ajuster notre description de la méthode *trier-et-chercher* de la section 4.3.2.2 en changeant la définition des ensembles $\mathcal{B}_\beta$ en $\mathcal{B}_\beta = \big\{\vec{b}_k \in B \;:\; \vec{b}_k \leq (\beta, \beta_j^1, \ldots, \beta_j^{d_B})\big\}$.

Concernant la complexité au pire des cas, l'approche de la section 4.3.2.2 donne un temps d'exécution en $\mathcal{O}((1+d_B)\,n_B\log(n_B)+n_A\log^{d_B+2}(n_B))$ avec $n_A = n_B = (m+1)^{n/2}$ et $d_B = m$. Ce qui nous donne le théorème :

Théorème 4.23 (Lenté *et al.* (2013))**.** *Le problème $P|d_i|\sum w_iU_i$ peut être résolu en temps $\mathcal{O}^*((m+1)^{n/2}\,(\frac{n}{2})^{m+2}\,\log^{m+2}(m+1))$ et espace $\mathcal{O}^*((m+1)^{n/2}\,(\frac{n}{2})^{m-1}\,\log^{m-1}(m))$. Dès que le nombre $m \geq 2$ de machines est fixé, ces complexités deviennent $\mathcal{O}^*((m+1)^{n/2})$. En particulier, le problème $P_2|d_i|\sum w_iU_i$ peut être résolu en temps $\mathcal{O}(1.7321^n)$.*

Notons qu'un algorithme trivial pour le problème $P_2|d_i|\sum w_iU_i$ demanderait un temps $\mathcal{O}^*(3^n)$. Pour chaque tâche, il suffirait de décider si elle est en avance en étant affectée sur la première machine, ou si elle est en avance en étant affectée sur la seconde machine, ou si elle est en retard (et donc arbitrairement ordonnancée sur la première machine, après les tâches en avance). L'algorithme en $\mathcal{O}(1.7321^n)$ obtenu par le théorème précédent améliore donc sensiblement cette complexité.

4.4 Un résumé des résultats connus

La table 4.2 présente un résumé des résultats connus et ceux présentés dans ce manuscrit. Les techniques utilisées se résument essentiellement à la *programmation dynamique* et à l'approche *trier-et-chercher*. Le cadre général de la *programmation dynamique* « à la » Held-Karp ou de *trier-et-chercher* pour résoudre des problèmes PSC et PPC permet sans aucun doute de résoudre bien d'autres problèmes d'ordonnancement, non listés ici.

problème	recherche exhaustive	meilleur algorithme	technique	référence
$1\|\text{prec}\|\sum w_iC_i$	$\mathcal{O}^*(n!)$	$\mathcal{O}^*(2^n)$	*programmation dynamique*	Woeginger (2001)
$1\|d_i\|\sum w_iU_i$	$\mathcal{O}^*(n!)$	$\mathcal{O}^*(2^n)$		Woeginger (2001)
$1\|d_i\|\sum T_i$	$\mathcal{O}^*(n!)$	$\mathcal{O}^*(2^n)$		Woeginger (2001)
$1\|cpct\|\sum f_i$	$\mathcal{O}^*(n!)$	$\mathcal{O}^*(2^n)$		Held et Karp (1962)
$P_m\|cpct\|\sum f_i$	$\mathcal{O}^*(m^n)$	$\mathcal{O}^*(3^n)$		section 4.2.2
$P_m\|cpct\|f_{\max}$	$\mathcal{O}^*(m^n)$	$\mathcal{O}^*(3^n)$		section 4.2.2
$F_3\|\|C_{\max}$	$\mathcal{O}^*(n!)$	$\mathcal{O}^*(3^n)$		section 4.2.3, Shang *et al.* (2015)
$1\|d_i\|\sum w_iU_i$	$\mathcal{O}^*(n!)$	$\mathcal{O}^*(2^{n/2})$	*trier-et-chercher*	section 4.3.2.1, Lenté *et al.* (2013)
$F_2\|\|C^k_{\max}$	$\mathcal{O}^*(2^n)$	$\mathcal{O}^*(2^{n/2})$		section 4.3.2.1, Lenté *et al.* (2013)
$P\|\|C_{\max}$	$\mathcal{O}^*(m^n)$	$\mathcal{O}^*(m^{n/2})$		section 4.3.3.1, Lenté *et al.* (2013)
$P_2\|\|C_{\max}$	$\mathcal{O}^*(2^n)$	$\mathcal{O}^*(2^{n/2})$		section 4.3.3.1, Lenté *et al.* (2013)
$P_3\|\|C_{\max}$	$\mathcal{O}^*(3^n)$	$\mathcal{O}(1.7321^n)$		section 4.3.3.1, Lenté *et al.* (2013)
$P\|d_i\|\sum w_iU_i$	$\mathcal{O}^*((m+1)^n)$	$\mathcal{O}^*((m+1)^{n/2})$		section 4.3.3.2, Lenté *et al.* (2013)
$P_2\|d_i\|\sum w_iU_i$	$\mathcal{O}^*(2^n)$	$\mathcal{O}(1.7321^n)$		section 4.3.3.2, Lenté *et al.* (2013)

TABLE 4.2 – Un résumé des complexités établies dans la littérature pour divers problèmes d'ordonnancement.

4.5 Conclusion

Les problèmes d'ordonnancement se révèlent particulièrement difficiles à résoudre, y compris sous l'angle des algorithmes modérément exponentiels. Lorsque nous avons attaqué ces problèmes, notre souhait était d'établir des algorithmes utilisables en pratique avec une garantie de performance au pire des cas. Pour cela, une technique naturelle est *brancher-et-réduire* dont résulte des algorithmes nécessitant (bien souvent) seulement un espace polynomial. Nous ne sommes pas encore parvenus à intégrer cette technique. Néanmoins, nous avons réussi à « pousser » d'autres techniques, comme la *programmation dynamique* (éventuellement combinée avec le *produit de convolution rapide*) ou *trier-et-chercher* afin de résoudre un certain nombre de problèmes d'ordonnancement. Cela nous a permis d'appréhender ces quelques techniques et de mieux comprendre les obstacles à la mise en œuvre. Typiquement, pour ces problèmes où l'on cherche une séquence optimale, y compris pour des problèmes de graphes, *brancher-et-réduire* semble peu utile. Il reste tout de même pertinent de déterminer des problèmes d'ordonnancement pour lesquels cette technique pourrait s'appliquer et donner des algorithmes avec des implémentations performantes.

Nous avons développé une approche par *programmation dynamique* pour le problème de flow-shop à trois machines. Plusieurs questions se posent. Tout d'abord, est-il possible d'adapter l'algorithme pour résoudre le problème à quatre machines ? Rappelons que dans ce cas, la séquence des travaux n'est pas nécessairement la même sur toutes les machines, contrairement au problème à trois machines. D'autre part, est-il possible de réduire le nombre de couples candidats à examiner afin de réduire la complexité de l'algorithme ? Aussi, est-il possible d'étendre les couples considérés au fur et à mesure de leur découverte (un peu comme si on explorait une arène par un parcours en profondeur) afin d'avoir un algorithme en espace polynomial (éventuellement en augmentant un peu la complexité en temps) ? Pourrait-on d'ailleurs concevoir des algorithmes avec un compromis entre leur temps d'exécution et l'espace nécessaire ? Finalement, que se passe-t-il si on regarde des variantes des problèmes que nous avons considérés, où par exemple, un délai de transfert entre chaque machine est imposé à chaque tâche ?

Concernant l'approche *trier-et-chercher*, nous avons observé à quel point il est important de structurer les données afin de rechercher des points dans un hyperrectangle. Les arbres d'intervalles (*range trees* en anglais) nous ont été particulièrement utiles. Serait-il possible d'améliorer le temps d'exécution de *trier-et-chercher*, en combinant cette technique avec *produit de convolution rapide* ? Pour la généralisation, peut-on diminuer l'espace nécessaire comme l'avaient fait Schroeppel et Shamir (1981) pour passer de $\mathcal{O}^*(2^{n/2})$ à $\mathcal{O}^*(2^{n/4})$ pour la résolution du Sac-à-Dos ? Nous avons vu quelques applications des PPC. Est-ce que notre approche pour le problème $P_m||C_{\max}$ fonctionnerait encore si les machines parallèles n'étaient pas identiques ?

Finalement, d'autres cadres existent pour modéliser des problèmes, tels que les programmes linéaires ou les CSP (*constraint satisfaction programming*). Des algorithmes modérément exponentiels ont été établis pour résoudre les CSP (Angelsmark, 2005; Gaspers et Sorkin, 2012). Là encore, une perspective serait d'étudier les liens entre les problèmes d'ordonnancement et les CSP.

Est-il également possible d'établir des résultats de complexité, sous certaines hypothèses (notamment sous les hypothèses ETH et SETH), pour montrer que des problèmes d'ordonnancement ne peuvent pas être résolus en dessous d'une certaine complexité ?

Conclusion

Les trois chapitres précédents ont mis en lumière trois axes de recherche que j'affectionne. Le premier est l'étude du problème ÉTIQUETAGE $L(2,1)$ qui m'a servi de fil conducteur tout au long de ces années depuis ma thèse. Au chapitre 2, nous avons présenté le résultat de nos recherches sur ce problème d'étiquetage, avec l'utilisation de nombreuses techniques (*branchement*, *mesurer-pour-conquérir*, *programmation dynamique*, *diviser-pour-régner*) pour traiter à la fois de problèmes de décision, d'optimisation et d'énumération. Le chapitre 3 est tout à fait dans la continuité de mes travaux de thèse. Nous y avons étudié deux problèmes autour de la domination : d'une part, le problème DOMINATION AVEC CAPACITÉS et d'autre part, l'énumération d'ensembles dominants minimaux. Pour le problème DOMINATION AVEC CAPACITÉS, qui généralise le problème DOMINATION, nous avons développé un algorithme qui n'utilise pas la technique *branchement* et dont la complexité est $\mathcal{O}(c^n\, n^d)$ où $c < 2$, avec un compromis entre le terme exponentiel c^n et le terme polynomial n^d. En ce qui concerne l'énumération des ensembles dominants minimaux, le temps d'exécution de notre algorithme est inévitablement exponentiel puisque le nombre d'objets à produire est de cet ordre de grandeur. Dans le dernier chapitre, nous avons abordé une problématique qui m'est nouvelle : celle des problèmes d'ordonnancement. S'attaquer à de tels problèmes a été essentiellement possible grâce à l'application d'une technique peu utilisée et que nous avons étendue : *trier-et-chercher*. Nous avons démontré toute l'utilité de cette technique pour la résolution d'une famille de problèmes, les *problèmes à plusieurs contraintes*. Comme conséquence, nous avons pu résoudre des problèmes d'ordonnancement pour lesquels aucune méthode plus rapide que l'approche naïve n'était connue.

À la fois notre algorithme en $\mathcal{O}(2.6488^n)$ pour la résolution de ÉTIQUETAGE $L(2,1)$, notre algorithme en $\mathcal{O}(n^{20}1.8573^n)$ pour DOMINATION AVEC CAPACITÉS, nos algorithmes en $O^*(3^{n/3})$ pour l'énumération des ensembles dominants minimaux dans les graphes splits et d'intervalles viennent améliorer les résultats précédents et sont les meilleurs actuellement connus. Pour plusieurs problèmes d'ordonnancement, nous avons proposé des algorithmes sensiblement plus rapides que les approches par recherche exhaustive. Cela ouvre la voie à de futures recherches. L'exploration de tous ces problèmes a permis d'en apprendre davantage sur les techniques de conception et d'analyse des algorithmes modérément exponentiels. Tout au long de ce manuscrit nous avons présenté et développé diverses techniques ; leurs applications à des problèmes de natures différentes aident à mieux cerner le cadre dans lequel il est préférable de se tourner vers telle technique plutôt qu'une autre. Des pistes plus spécifiques en vue d'améliorer la résolution de chaque problème, ainsi que quelques questions qui mériteraient d'être approfondies, ont été détaillée à la fin de chacun des chapitres.

Bibliographie

ABU-KHZAM, F. N., MOUAWAD, A. E. et LIEDLOFF, M. (2011). An exact algorithm for connected red-blue dominating set. *J. Discrete Algorithms*, 9(3):252–262. (cité pages 106 et 133)

ALON, N. et WORMALD, N. (2010). High Degree Graphs Contain Large-Star Factors. *In* KATONA, G., SCHRIJVER, A., SZŐNYI, T. et SÁGI, G., editors : *Fete of Combinatorics and Computer Science*, volume 20 of *Bolyai Society Mathematical Studies*, pages 9–21. Springer Berlin Heidelberg. (cité page 101)

ANGELSMARK, O. (2005). *Constructing Algorithms for Constraint Satisfaction and Related Problems : Methods and Applications*. Ph.D. Thesis, Linköping University, Suède. (cité pages 74 et 170)

ARORA, S. et BARAK, B. (2009). *Computational Complexity : A Modern Approach*. Cambridge University Press, New York, NY, USA. (cité page 8)

BAX, E. et FRANKLIN, J. (1996). A finite-difference sieve to count paths and cycles by length. *Information Processing Letters*, 60(4):171 – 176. (cité page 2)

BAX, E. T. (1993). Inclusion and exclusion algorithm for the Hamiltonian path problem. *Information Processing Letters*, 47(4):203 – 207. (cité page 2)

BAX, E. T. (1994). Algorithms to count paths and cycles. *Information Processing Letters*, 52(5):249 – 252. (cité page 2)

BEIGEL, R. (1999). Finding Maximum Independent Sets in Sparse and General Graphs. *In* TARJAN, R. E. et WARNOW, T., editors : *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, 17-19 January 1999, Baltimore, Maryland.*, pages 856–857. ACM/SIAM. (cité page 2)

BEIGEL, R. et EPPSTEIN, D. (2005). 3-coloring in time $O(1.3289^n)$. *J. Algorithms*, 54(2):168–204. (cité pages 2 et 74)

BELLMAN, R. (1958). On a Routing Problem. *Quarterly of Applied Mathematics*, 16:87–90. (cité page 148)

BELLMAN, R. (1962). Dynamic Programming Treatment of the Travelling Salesman Problem. *J. ACM*, 9(1):61–63. (cité page 140)

BERG, M. d., CHEONG, O., KREVELD, M. v. et OVERMARS, M. (2008). *Computational Geometry : Algorithms and Applications*. Springer-Verlag, 3rd edition. (cité pages 157, 158 et 159)

BERMAN, P. et FUJITO, T. (1999). On Approximation Properties of the Independent Set Problem for Low Degree Graphs. *Theory of Computing Systems*, 32(2):115–132. (cité page 19)

BINKELE-RAIBLE, D., FERNAU, H., GASPERS, S. et LIEDLOFF, M. (2013). Exact and Parameterized Algorithms for Max Internal Spanning Tree. *Algorithmica*, 65(1):95–128. (cité page 141)

BJÖRKLUND, A. (2014). Determinant Sums for Undirected Hamiltonicity. *SIAM J. Comput.*, 43(1):280–299. (cité page 2)

BJÖRKLUND, A., HUSFELDT, T., KASKI, P. et KOIVISTO, M. (2007). Fourier meets Möbius : fast subset convolution. *In* JOHNSON, D. S. et FEIGE, U., editors : *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pages 67–74. ACM. (cité pages 2, 29, 30, 141 et 143)

BJÖRKLUND, A., HUSFELDT, T. et KOIVISTO, M. (2009). Set Partitioning via Inclusion-Exclusion. *SIAM J. Comput.*, 39(2):546–563. (cité pages 2, 29 et 102)

BLAZEWICZ, J., LENSTRA, J. et KAN, A. (1983). Scheduling subject to resource constraints : classification and complexity. *Discrete Applied Mathematics*, 5(1):11 – 24. (cité pages 16 et 137)

BODLAENDER, H. L., KLOKS, T., TAN, R. B. et van LEEUWEN, J. (2004). Approximations for lambda-Colorings of Graphs. *Comput. J.*, 47(2):193–204. (cité page 37)

BODLAENDER, H. L., LOKSHTANOV, D. et PENNINKX, E. (2009). Planar Capacitated Dominating Set Is *W*[1]-Hard. *In* Chen et Fomin (2009), pages 50–60. (cité page 108)

BOURGEOIS, N. (2009). *Algorithmes exponentiels pour la résolution exacte et approchée de problèmes NP-difficiles*. Thèse de doctorat dirigée par Vangelis Th. Paschos, Université de Paris Dauphine (France). (cité page 2)

BOURGEOIS, N., ESCOFFIER, B., PASCHOS, V. T. et van ROOIJ, J. M. M. (2012). Fast Algorithms for max independent set. *Algorithmica*, 62(1-2):382–415. (cité pages 2 et 20)

BRANDSTÄDT, A., LE, V. et SPINRAD, J. (1999). *Graph Classes : A Survey*. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics. (cité page 122)

BRANDSTÄDT, A., LE, V. B. et SPINRAD, J. P. (1999). *Graph Classes : A Survey*. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA. (cité pages 8, 11 et 13)

BRUCKER, P. (2007). *Scheduling Algorithms*. Springer, Berlin. (cité pages 139 et 165)

CAI, L., CHENG, S.-W. et LAM, T. W., editors (2013). *Algorithms and Computation - 24th International Symposium, ISAAC 2013, Hong Kong, China, December 16-18, 2013, Proceedings*, volume 8283 of *Lecture Notes in Computer Science*. Springer. (cité pages 182 et 186)

CALAMONERI, T. (2011). The $L(h,k)$-Labelling Problem : An Updated Survey and Annotated Bibliography. *Comput. J.*, 54(8):1344–1371. (cité page 38)

CHANG, G. J. et KUO, D. (1996). The $L(2,1)$-Labeling Problem on Graphs. *SIAM J. Discrete Math.*, 9(2):309–316. (cité page 37)

CHEN, J. et FOMIN, F. V., editors (2009). *Parameterized and Exact Computation, 4th International Workshop, IWPEC 2009, Copenhagen, Denmark, September 10-11, 2009, Revised Selected Papers*, volume 5917 of *Lecture Notes in Computer Science*. Springer. (cité pages 176 et 182)

CHU, G., GASPERS, S., NARODYTSKA, N., SCHUTT, A. et WALSH, T. (2013). On the Complexity of Global Scheduling Constraints under Structural Restrictions. *In* ROSSI, F., editor : *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*. IJCAI/AAAI. (cité page 139)

COCHEFERT, M. (2012). *Algorithmes exacts et exponentiels pour les problèmes NP-difficiles sur les graphes et hypergraphes*. Thèse de doctorat dirigée par D. Kratsch, Université de Metz (France), Laboratoire d'Informatique Théorique et Appliquée. (cité page 2)

CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L. et STEIN, C. (2009). *Introduction to Algorithms*. The MIT Press, 3rd edition. (cité pages 8, 10, 27, 89, 148 et 152)

COUTURIER, J. (2012). *Algorithmes exacts et exponentiels sur les graphes : énumération, comptage et optimisation*. Thèse de doctorat dirigée par D. Kratsch, Université de Metz (France), Laboratoire d'Informatique Théorique et Appliquée. (cité page 2)

COUTURIER, J., GOLOVACH, P. A., KRATSCH, D., LIEDLOFF, M. et PYATKIN, A. V. (2013a). Colorings with few Colors : Counting, Enumeration and Combinatorial Bounds. *Theory Comput. Syst.*, 52(4):645–667. (cité pages 39, 52, 53 et 60)

COUTURIER, J., LETOURNEUR, R. et LIEDLOFF, M. (2015). On the number of minimal dominating sets on some graph classes. *Theor. Comput. Sci.*, 562:634–642. (cité pages 107, 120, 121 et 132)

COUTURIER, J.-F., HEGGERNES, P., van 't HOF, P. et KRATSCH, D. (2012). Minimal Dominating Sets in Graph Classes : Combinatorial Bounds and Enumeration. *In* BIELIKOVÁ, M., FRIEDRICH, G., GOTTLOB, G., KATZENBEISSER, S. et TURÁN, G., editors : *SOFSEM*, volume 7147 of *Lecture Notes in Computer Science*, pages 202–213. Springer. (cité pages 103, 120, 121, 125 et 132)

COUTURIER, J.-F., HEGGERNES, P., van 't HOF, P. et KRATSCH, D. (2013b). Minimal dominating sets in graph classes : Combinatorial bounds and enumeration. *Theoretical Computer Science*, 487:82–94. (cité pages 2, 5, 107, 120, 121, 125 et 132)

CYGAN, M. et KOWALIK, L. (2011). Channel assignment via fast zeta transform. *Inf. Process. Lett.*, 111(15):727–730. (cité pages 38 et 102)

CYGAN, M., PILIPCZUK, M., PILIPCZUK, M. et WOJTASZCZYK, J. O. (2014). Scheduling Partially Ordered Jobs Faster than 2^n. *Algorithmica*, 68(3):692–714. (cité pages 139 et 140)

CYGAN, M., PILIPCZUK, M. et WOJTASZCZYK, J. O. (2011). Capacitated domination faster than $O(2^n)$. *Inf. Process. Lett.*, 111(23-24):1099–1103. (cité pages 5, 103, 106, 108, 109, 110, 114 et 132)

DAVIS, M., LOGEMANN, G. et LOVELAND, D. W. (1962). A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397. (cité pages 1 et 18)

DAVIS, M. et PUTNAM, H. (1960). A Computing Procedure for Quantification Theory. *Journal of the ACM*, 7(3):201–215. (cité pages 1 et 18)

de RIDDER, H. N. *et al.* (2015). Information System on Graph Classes and their Inclusions (ISGCI). `http://www.graphclasses.org`. (cité page 11)

DIESTEL, R. (2010). *Graph Theory*. Springer-Verlag, Heidelberg. (cité pages 8 et 11)

DIJKSTRA, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271. (cité page 148)

DOM, M., LOKSHTANOV, D., SAURABH, S. et VILLANGER, Y. (2008). Capacitated Domination and Covering : A Parameterized Perspective. *In* GROHE, M. et NIEDERMEIER, R., editors : *Parameterized and Exact Computation, Third International Workshop, IWPEC 2008, Victoria, Canada, May 14-16, 2008. Proceedings*, volume 5018 of *Lecture Notes in Computer Science*, pages 78–90. Springer. (cité page 108)

DOWNEY, R. G. et FELLOWS, M. R. (1999). *Parameterized Complexity*. Monographs in Computer Science. Springer. (cité pages 19 et 105)

DREYFUS, S. E. et WAGNER, R. A. (1971). The steiner problem in graphs. *Networks*, 1(3):195–207. (cité page 30)

EGGEMANN, N., HAVET, F. et NOBLE, S. D. (2010). k-$L(2,1)$-labelling for planar graphs is NP-complete for $k \geq 4$. *Discrete Applied Mathematics*, 158(16):1777–1788. (cité page 37)

EPPSTEIN, D. (2003a). Small Maximal Independent Sets and Faster Exact Graph Coloring. *J. Graph Algorithms Appl.*, 7(2):131–140. (cité page 93)

EPPSTEIN, D. (2003b). The Traveling Salesman Problem for Cubic Graphs. *In* DEHNE, F. K. H. A., SACK, J. et SMID, M. H. M., editors : *Algorithms and Data Structures, 8th International Workshop, WADS 2003, Ottawa, Ontario, Canada, July 30 - August 1, 2003, Proceedings*, volume 2748 of *Lecture Notes in Computer Science*, pages 307–318. Springer. (cité page 2)

EPPSTEIN, D. (2006). Quasiconvex analysis of multivariate recurrence equations for backtracking algorithms. *ACM Transactions on Algorithms*, 2(4):492–509. (cité pages 2 et 48)

FEIGE, U. (1998). A Threshold of $\ln n$ for Approximating Set Cover. *J. ACM*, 45(4):634–652. (cité page 105)

FERNAU, H., KNEIS, J., KRATSCH, D., LANGER, A., LIEDLOFF, M., RAIBLE, D. et ROSSMANITH, P. (2011). An exact algorithm for the Maximum Leaf Spanning Tree problem. *Theor. Comput. Sci.*, 412(45):6290–6302. (cité page 106)

FIALA, J., GOLOVACH, P. A. et KRATOCHVÍL, J. (2005). Distance Constrained Labelings of Graphs of Bounded Treewidth. *In* CAIRES, L., ITALIANO, G. F., MONTEIRO, L., PALAMIDESSI, C. et YUNG, M., editors : *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 360–372. Springer. (cité page 102)

FIALA, J., GOLOVACH, P. A. et KRATOCHVÍL, J. (2008a). Computational Complexity of the Distance Constrained Labeling Problem for Trees (Extended Abstract). *In* ACETO, L., DAMGÅRD, I., GOLDBERG, L. A., HALLDÓRSSON, M. M., INGÓLFSDÓTTIR, A. et WALUKIEWICZ, I., editors : *Automata, Languages and Programming, 35th International Colloquium, ICALP 2008, Reykjavik, Iceland, July 7-11, 2008, Proceedings, Part I : Tack A : Algorithms, Automata, Complexity, and Games*, volume 5125 of *Lecture Notes in Computer Science*, pages 294–305. Springer. (cité page 102)

FIALA, J., KLOKS, T. et KRATOCHVÍL, J. (2001). Fixed-parameter complexity of lambda-labelings. *Discrete Applied Mathematics*, 113(1):59–72. (cité pages 37 et 39)

FIALA, J. et KRATOCHVÍL, J. (2002). Partial covers of graphs. *Discussiones Mathematicae Graph Theory*, 22(1):89–99. (cité page 38)

FIALA, J., KRATOCHVÍL, J. et PÓR, A. (2008b). On the computational complexity of partial covers of Theta graphs. *Discrete Applied Mathematics*, 156(7):1143–1149. (cité page 37)

FOMIN, F. V., GASPERS, S., PYATKIN, A. V. et RAZGON, I. (2008a). On the Minimum Feedback Vertex Set Problem : Exact and Enumeration Algorithms. *Algorithmica*, 52(2):293–307. (cité pages 2 et 119)

FOMIN, F. V., GASPERS, S., SAURABH, S. et STEPANOV, A. A. (2009a). On Two Techniques of Combining Branching and Treewidth. *Algorithmica*, 54(2):181–207. (cité page 53)

FOMIN, F. V., GOLOVACH, P. A., KRATOCHVÍL, J., KRATSCH, D. et LIEDLOFF, M. (2009b). Sort and Search : Exact algorithms for generalized domination. *Inf. Process. Lett.*, 109(14):795–798. (cité page 153)

FOMIN, F. V., GOLOVACH, P. A., KRATOCHVÍL, J., KRATSCH, D. et LIEDLOFF, M. (2011a). Branch and Recharge : Exact Algorithms for Generalized Domination. *Algorithmica*, 61(2):252–273. (cité page 106)

FOMIN, F. V., GRANDONI, F. et KRATSCH, D. (2005). Some New Techniques in Design and Analysis of Exact (Exponential) Algorithms. *Bulletin of the EATCS*, 87:47–77. (cité page 2)

FOMIN, F. V., GRANDONI, F. et KRATSCH, D. (2009c). A Measure & Conquer Approach for the Analysis of Exact Algorithms. *J. ACM*, 56(5):25 :1–25 :32. (cité pages 2, 25, 48, 105 et 106)

FOMIN, F. V., GRANDONI, F., PYATKIN, A. V. et STEPANOV, A. A. (2008b). Combinatorial bounds via measure and conquer : Bounding minimal dominating sets and applications. *ACM Transactions on Algorithms*, 5(1). (cité pages 5, 120 et 121)

FOMIN, F. V., HEGGERNES, P., KRATSCH, D., PAPADOPOULOS, C. et VILLANGER, Y. (2011b). Enumerating Minimal Subset Feedback Vertex Sets. *In* DEHNE, F., IACONO, J. et SACK, J.-R., editors : *WADS*, volume 6844 of *Lecture Notes in Computer Science*, pages 399–410. Springer. (cité page 119)

FOMIN, F. V., IWAMA, K., KRATSCH, D., KASKI, P., KOIVISTO, M., KOWALIK, L., OKAMOTO, Y., van ROOIJ, J. M. M. et WILLIAMS, R. (2008c). 08431 Open Problems - Moderately Exponential Time Algorithms. *In* FOMIN, F. V., IWAMA, K. et KRATSCH, D., editors : *Moderately Exponential Time Algorithms, 19.10. - 24.10.2008*, volume 08431 of *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany. (cité page 106)

FOMIN, F. V. et KASKI, P. (2013). Exact exponential algorithms. *Commun. ACM*, 56(3):80–88. (cité page 2)

FOMIN, F. V. et KRATSCH, D. (2010). *Exact Exponential Algorithms*. Springer-Verlag. (cité pages 2, 8, 23, 27, 29, 30, 44, 48, 130, 140 et 153)

FOMIN, F. V., KRATSCH, D. et WOEGINGER, G. J. (2004). Exact (Exponential) Algorithms for the Dominating Set Problem. *In* HROMKOVIC, J., NAGL, M. et WESTFECHTEL, B., editors : *Graph-Theoretic Concepts in Computer Science, 30th International Workshop,WG 2004, Bad Honnef, Germany, June 21-23, 2004, Revised Papers*, volume 3353 of *Lecture Notes in Computer Science*, pages 245–256. Springer. (cité page 105)

FOMIN, F. V. et VILLANGER, Y. (2010). Finding Induced Subgraphs via Minimal Triangulations. *In* MARION, J.-Y. et SCHWENTICK, T., editors : *STACS*, volume 5 of *LIPIcs*, pages 383–394. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. (cité pages 2 et 119)

FORD, L. R. (1956). Network Flow Theory. Report P-923, The Rand Corporation. (cité page 148)

GAREY, M. R. et JOHNSON, D. S. (1979). *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W. H. Freeman. (cité pages 8, 11 et 105)

GAREY, M. R., JOHNSON, D. S. et SETHI, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research*, 1(2):117–129. (cité page 144)

GASPERS, S. (2010). *Exponential Time Algorithms : Structures, Measures, and Bounds*. VDM Verlag Dr. Mueller e.K. (cité page 2)

GASPERS, S., KRATSCH, D. et LIEDLOFF, M. (2012). On Independent Sets and Bicliques in Graphs. *Algorithmica*, 62(3-4):637–658. (cité pages 2, 19 et 27)

GASPERS, S. et LIEDLOFF, M. (2012). A Branch-and-Reduce Algorithm for Finding a Minimum Independent Dominating Set. *Discrete Mathematics & Theoretical Computer Science*, 14(1):29–42. (cité page 106)

GASPERS, S. et MACKENZIE, S. (2015). On the Number of Minimal Separators in Graphs. *CoRR*, abs/1503.01203. (cité page 2)

GASPERS, S. et SORKIN, G. B. (2012). A universally fastest algorithm for Max 2-Sat, Max 2-CSP, and everything in between. *J. Comput. Syst. Sci.*, 78(1):305–335. (cité page 170)

GOLOVACH, P., HEGGERNES, P. et KRATSCH, D. (2015). Enumerating minimal connected dominating sets in graphs of bounded chordality. *In Parameterized and Exact Computation - 10th International Symposium, IPEC 2015, Patras, Greece, September 16-18, 2015.* (cité pages 107, 119, 121, 132 et 133)

GOLOVACH, P. A., KRATSCH, D. et COUTURIER, J. (2010). Colorings with Few Colors : Counting, Enumeration and Combinatorial Bounds. *In* THILIKOS, D. M., editor : *Graph Theoretic Concepts in Computer Science - 36th International Workshop, WG 2010, Zarós, Crete, Greece, June 28-30, 2010 Revised Papers*, volume 6410 of *Lecture Notes in Computer Science*, pages 39–50. (cité page 53)

GOLUMBIC, M. (2004). *Algorithmic Graph Theory and Perfect Graphs*. Annals of Discrete Mathematics. Elsevier Science, 2nd edition. (cité pages 11 et 122)

GONÇALVES, D. (2008). On the $L(p, 1)$-labelling of graphs. *Discrete Mathematics*, 308(8):1405–1414. (cité page 37)

GRAHAM, R. L., LAWLER, E. L., LENSTRA, J. K. et RINNOOY KAN, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling : a survey. *Annals of discrete mathematics*, 5(2):287–326. (cité pages 16 et 137)

GRANDONI, F. (2004). *Exact Algorithms for Hard Graph Problems*. Ph.D. Thesis, Università di Roma Tor Vergata, Rome, Italie. (cité page 105)

GRIGGS, J. R. et YEH, R. K. (1992). Labelling Graphs with a Condition at Distance 2. *SIAM J. Discrete Math.*, 5(4):586–595. (cité page 37)

GROMICHO, J. A. S., van HOORN, J. J., SALDANHA-DA-GAMA, F. et TIMMER, G. T. (2012). Solving the job-shop scheduling problem optimally by dynamic programming. *Computers & OR*, 39(12): 2968–2977. (cité pages 139, 140 et 144)

GUREVICH, Y. et SHELAH, S. (1987). Expected Computation Time for Hamiltonian Path Problem. *SIAM J. Comput.*, 16(3):486–502. (cité page 1)

HASUNUMA, T., ISHII, T., ONO, H. et UNO, Y. (2013). A Linear Time Algorithm for $L(2, 1)$-Labeling of Trees. *Algorithmica*, 66(3):654–681. (cité page 37)

HAVET, F., KLAZAR, M., KRATOCHVÍL, J., KRATSCH, D. et LIEDLOFF, M. (2011). Exact Algorithms for $L(2, 1)$-Labeling of Graphs. *Algorithmica*, 59(2):169–194. (cité pages 39, 46, 48, 51, 62, 78, 91 et 101)

HAVET, F., REED, B. A. et SERENI, J. (2008). $L(2, 1)$-labelling of graphs. *In* TENG, S., editor : *Proceedings of the Nineteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2008, San Francisco, California, USA, January 20-22, 2008*, pages 621–630. SIAM. (cité page 37)

HAVET, F., REED, B. A. et SERENI, J. (2012). Griggs and Yeh's Conjecture and $L(p,1)$-labelings. *SIAM J. Discrete Math.*, 26(1):145–168. (cité page 37)

HAYNES, T. W., HEDETNIEMI, S. T. et SLATER, P. J. (1998a). *Domination in graphs : advanced topics*. Marcel Dekker. (cité pages 15 et 105)

HAYNES, T. W., HEDETNIEMI, S. T. et SLATER, P. J. (1998b). *Fundamentals of domination in graphs*. Marcel Dekker. (cité pages 15 et 105)

HELD, M. et KARP, R. M. (1962). A Dynamic Programming Approach to Sequencing Problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210. (cité pages 1, 28, 135, 139, 140 et 169)

HOROWITZ, E. et SAHNI, S. (1974). Computing Partitions with Applications to the Knapsack Problem. *J. ACM*, 21(2):277–292. (cité pages 1, 6, 32, 135, 152, 153, 154 et 160)

IMPAGLIAZZO, R., PATURI, R. et ZANE, F. (2001). Which Problems Have Strongly Exponential Complexity ? *Journal of Computer and System Sciences*, 63(4):512–530. (cité pages 2, 19 et 33)

IWAMA, K. (2004). Worst-Case Upper Bounds for kSAT (Column : Algorithmics). *Bulletin of the EATCS*, 82:61–71. (cité page 2)

IWATA, Y. (2011). A Faster Algorithm for Dominating Set Analyzed by the Potential Method. *In* MARX, D. et ROSSMANITH, P., 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. (cité pages 2, 105 et 133)

JACKSON, J. R. (1955). Scheduling a production line to minimize maximum tardiness. Technical report, University of California, Los Angeles. (cité page 165)

JANSEN, K., LAND, F. et LAND, K. (2013). Bounding the Running Time of Algorithms for Scheduling and Packing Problems. *In* DEHNE, F., SOLIS-OBA, R. et SACK, J., editors : *Algorithms and Data Structures - 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, volume 8037 of *Lecture Notes in Computer Science*, pages 439–450. Springer. (cité pages 139 et 140)

JIAN, T. (1986). An $O(2^{0.304n})$ Algorithm for Solving Maximum Independent Set Problem. *IEEE Trans. Computers*, 35(9):847–851. (cité page 2)

JOHNSON, D. S. (1974). Approximation Algorithms for Combinatorial Problems. *J. Comput. Syst. Sci.*, 9(3):256–278. (cité page 105)

JOHNSON, D. S. (1987). The NP-Completeness Column : An Ongoing Guide. *J. Algorithms*, 8(2):285–303. (cité page 11)

JOHNSON, D. S., PAPADIMITRIOU, C. H. et YANNAKAKIS, M. (1988). On Generating All Maximal Independent Sets. *Inf. Process. Lett.*, 27(3):119–123. (cité pages 19, 29 et 79)

JOHNSON, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68. (cité page 144)

JUNOSZA-SZANIAWSKI, K., KRATOCHVÍL, J., LIEDLOFF, M., ROSSMANITH, P. et RZAZEWSKI, P. (2013a). Fast exact algorithm for $L(2,1)$-labeling of graphs. *Theor. Comput. Sci.*, 505:42–54. (cité pages 39, 62, 80, 86, 89, 91, 97 et 101)

JUNOSZA-SZANIAWSKI, K., KRATOCHVÍL, J., LIEDLOFF, M. et RZAZEWSKI, P. (2013b). Determining the $L(2,1)$-span in polynomial space. *Discrete Applied Mathematics*, 161(13-14):2052–2061. (cité pages 39, 62, 65, 73, 90 et 94)

JUNOSZA-SZANIAWSKI, K. et RZAZEWSKI, P. (2010). On Improved Exact Algorithms for $L(2,1)$-Labeling of Graphs. *In* ILIOPOULOS, C. S. et SMYTH, W. F., editors : *Combinatorial Algorithms - 21st International Workshop, IWOCA 2010, London, UK, July 26-28, 2010, Revised Selected Papers*, volume 6460 of *Lecture Notes in Computer Science*, pages 34–37. Springer. (cité pages 77, 78, 80 et 93)

JUNOSZA-SZANIAWSKI, K. et RZAZEWSKI, P. (2011). On the complexity of exact algorithm for $L(2,1)$-labeling of graphs. *Inf. Process. Lett.*, 111(14):697–701. (cité pages 78 et 80)

KANTÉ, M. M., LIMOUZY, V., MARY, A. et NOURINE, L. (2011). Enumeration of Minimal Dominating Sets and Variants. *In* OWE, O., STEFFEN, M. et TELLE, J. A., editors : *FCT*, volume 6914 of *Lecture Notes in Computer Science*, pages 298–309. Springer. (cité page 120)

KANTÉ, M. M., LIMOUZY, V., MARY, A. et NOURINE, L. (2012). On the Neighbourhood Helly of Some Graph Classes and Applications to the Enumeration of Minimal Dominating Sets. *In* CHAO, K.-M., sheng HSU, T. et LEE, D.-T., editors : *ISAAC*, volume 7676 of *Lecture Notes in Computer Science*, pages 289–298. Springer. (cité page 120)

KANTÉ, M. M., LIMOUZY, V., MARY, A., NOURINE, L. et UNO, T. (2013). On the Enumeration and Counting of Minimal Dominating sets in Interval and Permutation Graphs. *In* Cai *et al.* (2013), pages 339–349. (cité page 120)

KANTÉ, M. M. et NOURINE, L. (2015). Minimal Dominating Set Enumeration. *In* KAO, M., editor : *Encyclopedia of Algorithms*. Springer. (cité page 120)

KARP, R. M. (1982). Dynamic programming meets the principle of inclusion and exclusion. *Operations Research Letters*, 1(2):49 – 51. (cité page 2)

KLEINBERG, J. et TARDOS, E. (2006). *Algorithm Design*. Addison Wesley. (cité page 27)

KNEIS, J., LANGER, A. et ROSSMANITH, P. (2009). A Fine-grained Analysis of a Simple Independent Set Algorithm. *In* KANNAN, R. et KUMAR, K. N., editors : *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, volume 4 of *LIPIcs*, pages 287–298. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik. (cité page 2)

KNUTH, D. (1997). *The Art of Computer Programming, Volume 3 : Sorting and Searching*. Addison Wesley, 2nd edition. (cité page 33)

KOIVISTO, M. (2009). Partitioning into Sets of Bounded Cardinality. *In* Chen et Fomin (2009), pages 258–263. (cité pages 5, 103, 108, 109, 110, 111, 112, 114, 119 et 132)

KOWALIK, L. et SOCALA, A. (2014). Assigning Channels via the Meet-in-the-Middle Approach. *In* RAVI, R. et GØRTZ, I. L., editors : *Algorithm Theory - SWAT 2014 - 14th Scandinavian Symposium and Workshops, Copenhagen, Denmark, July 2-4, 2014. Proceedings*, volume 8503 of *Lecture Notes in Computer Science*, pages 282–293. Springer. (cité page 38)

KRÁL, D. (2005). An exact algorithm for the channel assignment problem. *Discrete Applied Mathematics*, 145(2):326–331. (cité page 38)

KRÁL, D. (2006). The Channel Assignment Problem with Variable Weights. *SIAM J. Discrete Math.*, 20(3):690–704. (cité page 78)

KRATSCH, D. et LIEDLOFF, M. (2007). An exact algorithm for the minimum dominating clique problem. *Theor. Comput. Sci.*, 385(1-3):226–240. (cité page 106)

KRZYWKOWSKI, M. (2013). Trees having many minimal dominating sets. *Information Processing Letters*, 113(8):276–279. (cité pages 120 et 121)

LAWLER, E. L. (1976). A Note on the Complexity of the Chromatic Number Problem. *Inf. Process. Lett.*, 5(3):66–67. (cité pages 28 et 29)

LAWLER, E. L. (1977). A "Pseudopolynomial" Algorithm for Sequencing Jobs to Minimize Total Tardiness. *In* P.L. HAMMER, E.L. Johnson, B. K. et NEMHAUSER, G., editors : *Studies in Integer Programming*, volume 1 of *Annals of Discrete Mathematics*, pages 331 – 342. Elsevier. (cité pages 139 et 140)

LAWLER, E. L. et MOORE, J. M. (1969). A Functional Equation and its Application to Resource Allocation and Sequencing Problems. *Management Science*, 16(1):77–84. (cité page 166)

LE GALL, F. (2014). Algebraic complexity theory and matrix multiplication. *In* NABESHIMA, K., NAGASAKA, K., WINKLER, F. et SZÁNTÓ, Á., editors : *International Symposium on Symbolic and Algebraic Computation, ISSAC '14, Kobe, Japan, July 23-25, 2014*, page 23. ACM. (cité page 82)

LENTÉ, C. (2012). Mathématiques, Ordonnancement et Santé. Habilitation à diriger des recherches. Université François Rabelais, Tours, France. (cité page 140)

LENTÉ, C., LIEDLOFF, M., SOUKHAL, A. et T'KINDT, V. (2011). Exponential-time algorithms for scheduling problems. *In Proceedings of the Workshop on Models and Algorithms for Planning and Scheduling Problems , MAPSP 2011, Nymburk, Czech Republic, 2011*. (cité pages 160 et 165)

LENTÉ, C., LIEDLOFF, M., SOUKHAL, A. et T'KINDT, V. (2013). On an extension of the Sort & Search method with application to scheduling theory. *Theor. Comput. Sci.*, 511:13–22. (cité pages 137, 156, 160, 164, 168 et 169)

LENTÉ, C., LIEDLOFF, M., SOUKHAL, A. et T'KINDT, V. (2014). Exponential Algorithms for Scheduling Problems. Technical report, Laboratoire d'Informatique, Université François Rabelais, Tours, France. (cité page 161)

LENTÉ, C., LIEDLOFF, M., T'KINDT, V. et TODINCA, I. (2015). Algorithmes modérément exponentiels pour problèmes NP-difficiles. *In* OLLINGER, N., editor : *Informatique Mathématique une photographie en 2015*. CNRS Editions. (cité pages 18, 137, 154 et 157)

LETOURNEUR, R. (2015). *Algorithmes exacts et exponentiels pour des problèmes de graphes*. Thèse de doctorat dirigée par I. Todinca et M. Liedloff, Université d'Orléans (France), Laboratoire d'Informatique Fondamentale d'Orléans. (cité page 2)

LIEDLOFF, M. (2007). *Algorithmes exacts et exponentiels pour les problèmes NP-difficiles : domination, variantes et généralisations*. Thèse de doctorat dirigée par D. Kratsch, Université de Metz (France), Laboratoire d'Informatique Théorique et Appliquée. (cité pages 2, 8, 44, 48, 105, 106 et 153)

LIEDLOFF, M. (2008). Finding a dominating set on bipartite graphs. *Inf. Process. Lett.*, 107(5):154–157. (cité page 105)

LIEDLOFF, M., TODINCA, I. et VILLANGER, Y. (2014). Solving Capacitated Dominating Set by using covering by subsets and maximum matching. *Discrete Applied Mathematics*, 168:60–68. (cité pages 107 et 112)

LONC, Z. et TRUSZCZYNSKI, M. (2008). On the number of minimal transversals in 3-uniform hypergraphs. *Discrete Mathematics*, 308(16):3668–3687. (cité pages 119 et 132)

MARX, D. (2011). Fixed-parameter tractable scheduling problems. *In* JANSEN, K., MATHIEU, C., SHACHNAI, H. et YOUNG, N. E., editors : *Packing and Scheduling Algorithms for Information and Communication Services (Dagstuhl Seminar 11091)*, volume 1, page 20, Dagstuhl, Germany. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. (cité page 139)

MNICH, M. et WIESE, A. (2014). Scheduling and Fixed-Parameter Tractability. *In* LEE, J. et VYGEN, J., editors : *Integer Programming and Combinatorial Optimization - 17th International Conference, IPCO 2014, Bonn, Germany, June 23-25, 2014. Proceedings*, volume 8494 of *Lecture Notes in Computer Science*, pages 381–392. Springer. (cité page 139)

MOON, J. et MOSER, L. (1965). On cliques in graphs. *Israel Journal of Mathematics*, 3:23–28. (cité pages 19, 29 et 119)

NEDERLOF, J., van ROOIJ, J. M. M. et van DIJK, T. C. (2014). Inclusion/Exclusion Meets Measure and Conquer. *Algorithmica*, 69(3):685–740. (cité pages 105 et 133)

OKAMOTO, Y., UNO, T. et UEHARA, R. (2008). Counting the number of independent sets in chordal graphs. *J. Discrete Algorithms*, 6(2):229–242. (cité page 19)

PAPADIMITRIOU, C. M. (1994). *Computational complexity*. Addison Wesley. (cité page 8)

PINEDO, M. (2012). *Scheduling : Theory, Algorithms, and Systems*. Springer. (cité pages 139, 144 et 145)

RANDERATH, B. et SCHIERMEYER, I. (2004). Exact algorithms for Minimum Dominating Set. Technical report zaik-469, Zentrum für Angewandte Informatik, Köln, Germany. (cité page 105)

RAZ, R. et SAFRA, S. (1997). A Sub-Constant Error-Probability Low-Degree Test, and a Sub-Constant Error-Probability PCP Characterization of NP. *In* LEIGHTON, F. T. et SHOR, P. W., editors : *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 475–484. ACM. (cité page 105)

ROBERTS, F. (1988). Private communication to J. Griggs. (cité page 37)

ROBERTSON, N. et SEYMOUR, P. D. (1983). Graph minors. I. Excluding a forest. *J. Comb. Theory, Ser. B*, 35(1):39–61. (cité page 11)

ROBERTSON, N. et SEYMOUR, P. D. (1986). Graph Minors. II. Algorithmic Aspects of Tree-Width. *J. Algorithms*, 7(3):309–322. (cité page 11)

ROBSON, J. M. (1986). Algorithms for Maximum Independent Sets. *J. Algorithms*, 7(3):425–440. (cité page 2)

ROTA, G.-C. (1964). On the foundations of combinatorial theory I. Theory of Möbius Functions. *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete*, 2(4):340–368. (cité page 2)

SCHIERMEYER, I. (2008). Efficiency in exponential time for domination-type problems. *Discrete Applied Mathematics*, 156(17):3291–3297. (cité page 105)

SCHÖNING, U. (2005a). Algorithmics in Exponential Time. *In* DIEKERT, V. et DURAND, B., editors : *STACS 2005, 22nd Annual Symposium on Theoretical Aspects of Computer Science, Stuttgart, Germany, February 24-26, 2005, Proceedings*, volume 3404 of *Lecture Notes in Computer Science*, pages 36–43. Springer. (cité page 2)

SCHÖNING, U. (2005b). New Algorithmic Paradigms in Exponential Time Algorithms. *In* COOPER, S. B., LÖWE, B. et TORENVLIET, L., editors : *New Computational Paradigms, First Conference on Computability in Europe, CiE 2005, Amsterdam, The Netherlands, June 8-12, 2005, Proceedings*, volume 3526 of *Lecture Notes in Computer Science*, pages 429–429. Springer. (cité page 2)

SCHROEPPEL, R. et SHAMIR, A. (1981). A $T = O(2^{n/2})$, $S = O(2^{n/4})$ Algorithm for Certain NP-Complete Problems. *SIAM J. Comput.*, 10(3):456–464. (cité pages 32, 152, 154 et 170)

SHANG, L., LENTÉ, C., LIEDLOFF, M. et T'KINDT, V. (2015). An exponential dynamic programming algorithm for the 3-machine flowshop scheduling problem to minimize the makespan. *In Proceedings of the Multidisciplinary International Conference on Scheduling : Theory and Applications, MISTA 2015, Prague, Czech Republic, August 25-28, 2015*. (cité pages 144 et 169)

SHINDO, M. et TOMITA, E. (1990). A Simple Algorithm for Finding a Maximum Clique and Its Worst-Case Time Complexity. *Systems and Computers in Japan*, 21(3):1–13. (cité page 2)

STEPANOV, A. (2008). *Exact Algorithms for Hard Listing, Counting and Decision Problems*. Thèse dirigée par F.V. Fomin, Université de Bergen (Norvège). (cité page 2)

STRASSEN, V. (1969). Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356. (cité pages 80 et 81)

TARJAN, R. et TROJANOWSKI, A. (1977). Finding a maximum independent set. *SIAM Journal on Computing*, 6:537–546. (cité pages 2 et 20)

TELLE, J. A. (1994). Complexity of Domination-Type Problems in Graphs. *Nord. J. Comput.*, 1(1):157–171. (cité page 106)

VALIANT, L. G. (1979). The Complexity of Computing the Permanent. *Theor. Comput. Sci.*, 8:189–201. (cité page 114)

van ROOIJ, J. M. (2011). *Exact exponential-time algorithms for domination problems in graphs*. Ph.D. Thesis, Department of Information and Computing Sciences, Utrecht University, Utrecht, Pays-Bas. (cité pages 2, 81 et 105)

van ROOIJ, J. M. M. et BODLAENDER, H. L. (2011). Exact algorithms for dominating set. *Discrete Applied Mathematics*, 159(17):2147–2164. (cité page 105)

van ROOIJ, J. M. M., BODLAENDER, H. L. et ROSSMANITH, P. (2009). Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. *In* FIAT, A. et SANDERS, P., editors : *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, volume 5757 of *Lecture Notes in Computer Science*, pages 566–577. Springer. (cité page 81)

WEST, D. B. (2001). *Introduction to Graph Theory*. Prentice Hall, 2nd edition. (cité pages 8 et 11)

WOEGINGER, G. J. (2001). Exact Algorithms for NP-Hard Problems : A Survey. *In* JÜNGER, M., REINELT, G. et RINALDI, G., editors : *Combinatorial Optimization - Eureka, You Shrink !, Papers Dedicated to Jack Edmonds, 5th International Workshop, Aussois, France, March 5-9, 2001, Revised Papers*, volume 2570 of *Lecture Notes in Computer Science*, pages 185–208. Springer. (cité pages 2, 8, 139, 140, 153 et 169)

WOEGINGER, G. J. (2004). Space and Time Complexity of Exact Algorithms : Some Open Problems (Invited Talk). *In* DOWNEY, R. G., FELLOWS, M. R. et DEHNE, F. K. H. A., editors : *Parameterized and Exact Computation, First International Workshop, IWPEC 2004, Bergen, Norway, September 14-17, 2004, Proceedings*, volume 3162 of *Lecture Notes in Computer Science*, pages 281–290. Springer. (cité page 2)

WOOD, D. (2011). On the number of maximal independent sets in a graph. *Discrete Mathematics & Theoretical Computer Science*, 13(3):17–20. (cité page 95)

XIAO, M. et NAGAMOCHI, H. (2013). Exact Algorithms for Maximum Independent Set. *In* Cai *et al.* (2013), pages 328–338. (cité pages 2 et 20)

XIAO, M. et NAGAMOCHI, H. (2015). An improved exact algorithm for undirected feedback vertex set. *Journal of Combinatorial Optimization*, 30(2):214–241. (cité page 2)

YATES, F. (1937). The design and analysis of factorial experiments. *Technical Communication no. 35 of the Commonwealth Bureau of Soils*. (cité page 2)